

glucat

0.12.0

Generated by Doxygen 1.12.0

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
5.1 cga3 Namespace Reference	9
5.1.1 Detailed Description	9
5.1.2 Function Documentation	9
5.1.2.1 agc3()	9
5.1.2.2 cga3()	10
5.1.2.3 cga3std()	10
5.2 glucat Namespace Reference	10
5.2.1 Typedef Documentation	22
5.2.1.1 index_t	22
5.2.1.2 intfn	22
5.2.1.3 intintfn	22
5.2.1.4 set_value_t	23
5.2.1.5 tuning_fast	23
5.2.1.6 tuning_naive	23
5.2.1.7 tuning_slow	24
5.2.2 Function Documentation	24
5.2.2.1 _GLUCAT_CTAssert() [1/3]	24
5.2.2.2 _GLUCAT_CTAssert() [2/3]	24
5.2.2.3 _GLUCAT_CTAssert() [3/3]	24
5.2.2.4 abs()	25
5.2.2.5 acos() [1/2]	25
5.2.2.6 acos() [2/2]	25
5.2.2.7 acosh() [1/2]	25
5.2.2.8 acosh() [2/2]	26
5.2.2.9 approx_equal() [1/2]	26
5.2.2.10 approx_equal() [2/2]	26
5.2.2.11 asin() [1/2]	27
5.2.2.12 asin() [2/2]	27
5.2.2.13 asinh() [1/2]	27
5.2.2.14 asinh() [2/2]	27
5.2.2.15 atan() [1/2]	28

5.2.2.16 atan() [2/2]	28
5.2.2.17 atanh() [1/2]	28
5.2.2.18 atanh() [2/2]	28
5.2.2.19 cascade_log()	29
5.2.2.20 check_complex()	29
5.2.2.21 clifford_exp()	29
5.2.2.22 compare()	30
5.2.2.23 complexifier()	30
5.2.2.24 conj()	30
5.2.2.25 cos() [1/2]	30
5.2.2.26 cos() [2/2]	31
5.2.2.27 cosh()	31
5.2.2.28 cr_sqrt()	31
5.2.2.29 crd_of_mult() [1/2]	32
5.2.2.30 crd_of_mult() [2/2]	32
5.2.2.31 db_sqrt()	32
5.2.2.32 db_step()	32
5.2.2.33 elliptic()	33
5.2.2.34 error_squared()	33
5.2.2.35 error_squared_tol()	33
5.2.2.36 even()	34
5.2.2.37 exp() [1/2]	34
5.2.2.38 exp() [2/2]	34
5.2.2.39 fast()	34
5.2.2.40 folded_dim()	35
5.2.2.41 imag()	35
5.2.2.42 inv()	35
5.2.2.43 inverse_gray()	35
5.2.2.44 inverse_reversed_gray()	36
5.2.2.45 involute()	36
5.2.2.46 log() [1/4]	36
5.2.2.47 log() [2/4]	36
5.2.2.48 log() [3/4]	37
5.2.2.49 log() [4/4]	37
5.2.2.50 log2()	37
5.2.2.51 matrix_log()	37
5.2.2.52 matrix_sqrt()	38
5.2.2.53 max_abs()	38
5.2.2.54 max_pos()	38
5.2.2.55 min_neg()	38
5.2.2.56 norm()	39
5.2.2.57 odd()	39

5.2.2.58 <code>offset_level()</code>	39
5.2.2.59 <code>operator"!=()</code> [1/3]	39
5.2.2.60 <code>operator"!=()</code> [2/3]	40
5.2.2.61 <code>operator"!=()</code> [3/3]	40
5.2.2.62 <code>operator%()</code> [1/3]	40
5.2.2.63 <code>operator%()</code> [2/3]	40
5.2.2.64 <code>operator%()</code> [3/3]	41
5.2.2.65 <code>operator&()</code> [1/4]	41
5.2.2.66 <code>operator&()</code> [2/4]	41
5.2.2.67 <code>operator&()</code> [3/4]	41
5.2.2.68 <code>operator&()</code> [4/4]	42
5.2.2.69 <code>operator*()</code> [1/6]	42
5.2.2.70 <code>operator*()</code> [2/6]	42
5.2.2.71 <code>operator*()</code> [3/6]	42
5.2.2.72 <code>operator*()</code> [4/6]	43
5.2.2.73 <code>operator*()</code> [5/6]	43
5.2.2.74 <code>operator*()</code> [6/6]	43
5.2.2.75 <code>operator+()</code> [1/3]	43
5.2.2.76 <code>operator+()</code> [2/3]	44
5.2.2.77 <code>operator+()</code> [3/3]	44
5.2.2.78 <code>operator-()</code> [1/3]	44
5.2.2.79 <code>operator-()</code> [2/3]	44
5.2.2.80 <code>operator-()</code> [3/3]	45
5.2.2.81 <code>operator/()</code> [1/5]	45
5.2.2.82 <code>operator/()</code> [2/5]	45
5.2.2.83 <code>operator/()</code> [3/5]	45
5.2.2.84 <code>operator/()</code> [4/5]	46
5.2.2.85 <code>operator/()</code> [5/5]	46
5.2.2.86 <code>operator<<()</code> [1/5]	46
5.2.2.87 <code>operator<<()</code> [2/5]	46
5.2.2.88 <code>operator<<()</code> [3/5]	46
5.2.2.89 <code>operator<<()</code> [4/5]	47
5.2.2.90 <code>operator<<()</code> [5/5]	47
5.2.2.91 <code>operator>>()</code> [1/3]	47
5.2.2.92 <code>operator>>()</code> [2/3]	47
5.2.2.93 <code>operator>>()</code> [3/3]	47
5.2.2.94 <code>operator^()</code> [1/4]	48
5.2.2.95 <code>operator^()</code> [2/4]	48
5.2.2.96 <code>operator^()</code> [3/4]	48
5.2.2.97 <code>operator^()</code> [4/4]	48
5.2.2.98 <code>operator" ()</code> [1/4]	49
5.2.2.99 <code>operator" ()</code> [2/4]	49

5.2.2.100 operator" () [3/4]	49
5.2.2.101 operator" () [4/4]	49
5.2.2.102 outer_pow()	50
5.2.2.103 pade_approx()	50
5.2.2.104 pade_log()	50
5.2.2.105 pos_mod()	50
5.2.2.106 pow() [1/2]	51
5.2.2.107 pow() [2/2]	51
5.2.2.108 pure()	51
5.2.2.109 quad()	51
5.2.2.110 real()	52
5.2.2.111 reframe()	52
5.2.2.112 reverse()	52
5.2.2.113 scalar()	52
5.2.2.114 sign_of_square()	53
5.2.2.115 sin() [1/2]	53
5.2.2.116 sin() [2/2]	53
5.2.2.117 sinh()	53
5.2.2.118 sqrt() [1/4]	54
5.2.2.119 sqrt() [2/4]	54
5.2.2.120 sqrt() [3/4]	54
5.2.2.121 sqrt() [4/4]	54
5.2.2.122 star() [1/3]	55
5.2.2.123 star() [2/3]	55
5.2.2.124 star() [3/3]	55
5.2.2.125 tan() [1/2]	55
5.2.2.126 tan() [2/2]	56
5.2.2.127 tanh()	56
5.2.2.128 to_demote()	56
5.2.2.129 to_promote()	56
5.2.2.130 try_catch() [1/2]	57
5.2.2.131 try_catch() [2/2]	57
5.2.2.132 vector_part()	57
5.2.3 Variable Documentation	57
5.2.3.1 BITS_PER_SET_VALUE	57
5.2.3.2 DEFAULT_HI	57
5.2.3.3 l_ln2	58
5.2.3.4 l_pi	58
5.2.3.5 MS_PER_S	58
5.2.3.6 Tuning_Fast_Basis_Max_Count	58
5.2.3.7 Tuning_Fast_CR_Sqrt_Max_Steps	58
5.2.3.8 Tuning_Fast_DB_Sqrt_Max_Steps	58

5.2.3.9	Tuning_Fast_Div_Max_Steps	58
5.2.3.10	Tuning_Fast_Fast_Size_Threshold	59
5.2.3.11	Tuning_Fast_Inv_Fast_Dim_Threshold	59
5.2.3.12	Tuning_Fast_Log_Max_Inner_Steps	59
5.2.3.13	Tuning_Fast_Log_Max_Outer_Steps	59
5.2.3.14	Tuning_Fast_Mult_Matrix_Threshold	59
5.2.3.15	Tuning_Fast_Products_Size_Threshold	59
5.2.3.16	Tuning_Int_Digits	59
5.2.3.17	Tuning_Max_Threshold	59
5.2.3.18	Tuning_Naive_Basis_Max_Count	60
5.2.3.19	Tuning_Naive_Fast_Size_Threshold	60
5.2.3.20	Tuning_Naive_Inv_Fast_Dim_Threshold	60
5.2.3.21	Tuning_Naive_Mult_Matrix_Threshold	60
5.2.3.22	Tuning_Slow_Basis_Max_Count	60
5.2.3.23	Tuning_Slow_Fast_Size_Threshold	60
5.2.3.24	Tuning_Slow_Inv_Fast_Dim_Threshold	60
5.2.3.25	Tuning_Slow_Mult_Matrix_Threshold	60
5.2.3.26	Tuning_Slow_Products_Size_Threshold	61
5.3	glucat::gen Namespace Reference	61
5.3.1	Typedef Documentation	61
5.3.1.1	signature_t	61
5.3.2	Variable Documentation	61
5.3.2.1	offset_to_super	61
5.4	glucat::matrix Namespace Reference	62
5.4.1	Typedef Documentation	63
5.4.1.1	eig_case_t	63
5.4.2	Function Documentation	63
5.4.2.1	classify_eigenvalues()	63
5.4.2.2	eigenvalues()	64
5.4.2.3	inner()	64
5.4.2.4	isinf()	64
5.4.2.5	isnan()	64
5.4.2.6	kron()	65
5.4.2.7	mono_kron()	65
5.4.2.8	mono_prod()	65
5.4.2.9	nnz()	65
5.4.2.10	nork()	66
5.4.2.11	nork_range()	66
5.4.2.12	norm_frob2()	66
5.4.2.13	prod()	66
5.4.2.14	signed_perm_nork()	67
5.4.2.15	sparse_prod()	67

5.4.2.16 to_lapack()	67
5.4.2.17 trace()	67
5.4.2.18 unit()	68
5.5 glucat::timing Namespace Reference	68
5.5.1 Function Documentation	68
5.5.1.1 elapsed()	68
5.5.2 Variable Documentation	68
5.5.2.1 EXTRA_TRIALS	68
5.5.2.2 MS_PER_CLOCK	69
5.5.2.3 MS_PER_SEC	69
5.6 pade Namespace Reference	69
5.7 PyClical Namespace Reference	70
5.7.1 Function Documentation	70
5.7.1.1 _test()	70
5.7.1.2 clifford_hidden_doctests()	71
5.7.1.3 e()	72
5.7.1.4 index_set_hidden_doctests()	72
5.7.1.5 istpq()	73
5.7.2 Variable Documentation	74
5.7.2.1 __version__	74
5.7.2.2 cl	74
5.7.2.3 fill	74
5.7.2.4 i	74
5.7.2.5 ist	74
5.7.2.6 ixt	74
5.7.2.7 lhs	74
5.7.2.8 nbar3	75
5.7.2.9 ninf3	75
5.7.2.10 None	75
5.7.2.11 obj	75
5.7.2.12 pi	75
5.7.2.13 rhs	75
5.7.2.14 scalar_epsilon	75
5.7.2.15 tau	75
5.7.2.16 threshold	76
5.7.2.17 tol	76
5.8 std Namespace Reference	76
6 Class Documentation	77
6.1 glucat::basis_table< Scalar_T, LO, HI, Matrix_T > Class Template Reference	77
6.1.1 Detailed Description	78
6.1.2 Constructor & Destructor Documentation	78

6.1.2.1 basis_table() [1/2]	78
6.1.2.2 ~basis_table()	78
6.1.2.3 basis_table() [2/2]	78
6.1.3 Member Function Documentation	79
6.1.3.1 basis()	79
6.1.3.2 operator=()	79
6.1.4 Friends And Related Symbol Documentation	79
6.1.4.1 friend_for_private_destructor	79
6.2 glucat::bool_to_type< truth_value > Class Template Reference	79
6.2.1 Detailed Description	80
6.2.2 Member Enumeration Documentation	80
6.2.2.1 anonymous enum	80
6.3 PyClical.clifford Class Reference	80
6.3.1 Detailed Description	82
6.3.2 Member Function Documentation	82
6.3.2.1 __add__()	82
6.3.2.2 __and__()	82
6.3.2.3 __call__()	83
6.3.2.4 __cinit__()	83
6.3.2.5 __contains__()	84
6.3.2.6 __dealloc__()	84
6.3.2.7 __getitem__()	84
6.3.2.8 __iadd__()	85
6.3.2.9 __iand__()	85
6.3.2.10 __idiv__()	85
6.3.2.11 __imod__()	86
6.3.2.12 __imul__()	86
6.3.2.13 __ior__()	86
6.3.2.14 __isub__()	87
6.3.2.15 __iter__()	87
6.3.2.16 __ixor__()	87
6.3.2.17 __mod__()	88
6.3.2.18 __mul__()	88
6.3.2.19 __neg__()	88
6.3.2.20 __or__()	89
6.3.2.21 __pos__()	89
6.3.2.22 __pow__()	89
6.3.2.23 __repr__()	90
6.3.2.24 __richcmp__()	90
6.3.2.25 __str__()	90
6.3.2.26 __sub__()	91
6.3.2.27 __truediv__()	91

6.3.2.28 <code>__xor__()</code>	91
6.3.2.29 <code>abs()</code>	92
6.3.2.30 <code>conj()</code>	92
6.3.2.31 <code>even()</code>	92
6.3.2.32 <code>frame()</code>	93
6.3.2.33 <code>inv()</code>	93
6.3.2.34 <code>involute()</code>	93
6.3.2.35 <code>isinf()</code>	94
6.3.2.36 <code>isnan()</code>	94
6.3.2.37 <code>max_abs()</code>	94
6.3.2.38 <code>norm()</code>	95
6.3.2.39 <code>odd()</code>	95
6.3.2.40 <code>outer_pow()</code>	95
6.3.2.41 <code>pow()</code>	96
6.3.2.42 <code>pure()</code>	96
6.3.2.43 <code>quad()</code>	96
6.3.2.44 <code>reframe()</code>	97
6.3.2.45 <code>reverse()</code>	97
6.3.2.46 <code>scalar()</code>	97
6.3.2.47 <code>truncated()</code>	98
6.3.2.48 <code>vector_part()</code>	98
6.3.3 Member Data Documentation	98
6.3.3.1 instance	98
6.4 <code>glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T ></code> Class Template Reference	99
6.4.1 Detailed Description	101
6.4.2 Member Typedef Documentation	101
6.4.2.1 <code>index_set_t</code>	101
6.4.2.2 <code>multivector_t</code>	101
6.4.2.3 <code>pair_t</code>	101
6.4.2.4 <code>scalar_t</code>	101
6.4.2.5 <code>vector_t</code>	102
6.4.3 Constructor & Destructor Documentation	102
6.4.3.1 <code>~clifford_algebra()</code>	102
6.4.4 Member Function Documentation	102
6.4.4.1 <code>classname()</code>	102
6.4.4.2 <code>conj()</code>	102
6.4.4.3 <code>even()</code>	102
6.4.4.4 <code>frame()</code>	102
6.4.4.5 <code>grade()</code>	103
6.4.4.6 <code>inv()</code>	103
6.4.4.7 <code>involute()</code>	103
6.4.4.8 <code>isinf()</code>	103

6.4.4.9 isnan()	103
6.4.4.10 max_abs()	103
6.4.4.11 norm()	104
6.4.4.12 odd()	104
6.4.4.13 operator%=()	104
6.4.4.14 operator&=()	104
6.4.4.15 operator()()	104
6.4.4.16 operator*=() [1/2]	104
6.4.4.17 operator*=() [2/2]	105
6.4.4.18 operator+=() [1/2]	105
6.4.4.19 operator+=() [2/2]	105
6.4.4.20 operator-()	105
6.4.4.21 operator-=() [1/2]	105
6.4.4.22 operator-=() [2/2]	105
6.4.4.23 operator/=() [1/2]	106
6.4.4.24 operator/=() [2/2]	106
6.4.4.25 operator==() [1/2]	106
6.4.4.26 operator==() [2/2]	106
6.4.4.27 operator[]()	106
6.4.4.28 operator^()	106
6.4.4.29 operator" =()	107
6.4.4.30 outer_pow()	107
6.4.4.31 pow()	107
6.4.4.32 pure()	107
6.4.4.33 quad()	107
6.4.4.34 reverse()	107
6.4.4.35 scalar()	108
6.4.4.36 truncated()	108
6.4.4.37 vector_part() [1/2]	108
6.4.4.38 vector_part() [2/2]	108
6.4.4.39 write() [1/2]	108
6.4.4.40 write() [2/2]	109
6.4.5 Member Data Documentation	109
6.4.5.1 default_truncation	109
6.4.5.2 v_hi	109
6.4.5.3 v_lo	109
6.5 glucat::compare_types< LHS_T, RHS_T > Class Template Reference	109
6.5.1 Detailed Description	110
6.5.2 Member Enumeration Documentation	110
6.5.2.1 anonymous enum	110
6.6 glucat::compare_types< T, T > Class Template Reference	110
6.6.1 Detailed Description	110

6.6.2 Member Enumeration Documentation	110
6.6.2.1 anonymous enum	110
6.7 glucat::control_t Class Reference	111
6.7.1 Detailed Description	112
6.7.2 Constructor & Destructor Documentation	112
6.7.2.1 control_t() [1/3]	112
6.7.2.2 control_t() [2/3]	112
6.7.2.3 ~control_t()	112
6.7.2.4 control_t() [3/3]	112
6.7.3 Member Function Documentation	113
6.7.3.1 call() [1/2]	113
6.7.3.2 call() [2/2]	113
6.7.3.3 catch_exceptions()	113
6.7.3.4 control()	113
6.7.3.5 operator=()	113
6.7.3.6 valid()	114
6.7.3.7 verbose()	114
6.7.4 Friends And Related Symbol Documentation	114
6.7.4.1 friend_for_private_destructor	114
6.7.5 Member Data Documentation	114
6.7.5.1 m_catch_exceptions	114
6.7.5.2 m_valid	114
6.7.5.3 m_verbose_output	115
6.8 glucat::CTAssertion< bool > Struct Template Reference	115
6.8.1 Detailed Description	115
6.9 glucat::CTAssertion< true > Struct Reference	115
6.9.1 Detailed Description	115
6.10 glucat::numeric_traits< Scalar_T >::demoted Struct Reference	116
6.10.1 Detailed Description	116
6.10.2 Member Typedef Documentation	116
6.10.2.1 type	116
6.11 glucat::matrix::eig_genus< Matrix_T > Struct Template Reference	116
6.11.1 Detailed Description	117
6.11.2 Member Typedef Documentation	117
6.11.2.1 Scalar_T	117
6.11.3 Member Data Documentation	117
6.11.3.1 m_eig_case	117
6.11.3.2 m_is_singular	117
6.11.3.3 m_safe_arg	118
6.12 glucat::error< Class_T > Class Template Reference	118
6.12.1 Detailed Description	119
6.12.2 Constructor & Destructor Documentation	120

6.12.2.1 error() [1/2]	120
6.12.2.2 error() [2/2]	120
6.12.3 Member Function Documentation	120
6.12.3.1 classname()	120
6.12.3.2 heading()	120
6.12.3.3 print_error_msg()	120
6.13 glucat::framed_multi< Scalar_T, LO, HI, Tune_P > Class Template Reference	121
6.13.1 Detailed Description	126
6.13.2 Member Typedef Documentation	126
6.13.2.1 const_iterator	126
6.13.2.2 error_t	126
6.13.2.3 framed_multi_t	126
6.13.2.4 framed_pair_t	126
6.13.2.5 index_set_t	127
6.13.2.6 iterator	127
6.13.2.7 map_t	127
6.13.2.8 matrix_multi_t	127
6.13.2.9 matrix_t	127
6.13.2.10 multivector_t	127
6.13.2.11 scalar_t	128
6.13.2.12 size_type	128
6.13.2.13 sorted_map_t	128
6.13.2.14 term_t	128
6.13.2.15 tune_p	128
6.13.2.16 var_term_t	128
6.13.2.17 vector_t	129
6.13.3 Constructor & Destructor Documentation	129
6.13.3.1 ~framed_multi()	129
6.13.3.2 framed_multi() [1/15]	129
6.13.3.3 framed_multi() [2/15]	129
6.13.3.4 framed_multi() [3/15]	129
6.13.3.5 framed_multi() [4/15]	130
6.13.3.6 framed_multi() [5/15]	130
6.13.3.7 framed_multi() [6/15]	130
6.13.3.8 framed_multi() [7/15]	130
6.13.3.9 framed_multi() [8/15]	131
6.13.3.10 framed_multi() [9/15]	131
6.13.3.11 framed_multi() [10/15]	131
6.13.3.12 framed_multi() [11/15]	131
6.13.3.13 framed_multi() [12/15]	132
6.13.3.14 framed_multi() [13/15]	132
6.13.3.15 framed_multi() [14/15]	132

6.13.3.16 framed_multi() [15/15]	132
6.13.4 Member Function Documentation	133
6.13.4.1 centre_pm4_qp4()	133
6.13.4.2 centre_pp4_qm4()	133
6.13.4.3 centre_qp1_pm1()	133
6.13.4.4 classname()	133
6.13.4.5 divide()	134
6.13.4.6 fast()	134
6.13.4.7 fast_framed_multi()	134
6.13.4.8 fast_matrix_multi()	134
6.13.4.9 fold()	135
6.13.4.10 nbr_terms()	135
6.13.4.11 operator+=()	135
6.13.4.12 random()	135
6.13.4.13 unfold()	136
6.13.5 Friends And Related Symbol Documentation	136
6.13.5.1 exp	136
6.13.5.2 framed_multi	136
6.13.5.3 matrix_multi	136
6.13.5.4 operator%	136
6.13.5.5 operator&	137
6.13.5.6 operator*	137
6.13.5.7 operator/	137
6.13.5.8 operator<< [1/2]	137
6.13.5.9 operator<< [2/2]	137
6.13.5.10 operator>>	137
6.13.5.11 operator^	137
6.13.5.12 operator"	138
6.13.5.13 star	138
6.14 glucat::generator_table< Matrix_T > Class Template Reference	138
6.14.1 Detailed Description	140
6.14.2 Constructor & Destructor Documentation	140
6.14.2.1 generator_table() [1/2]	140
6.14.2.2 ~generator_table()	140
6.14.2.3 generator_table() [2/2]	140
6.14.3 Member Function Documentation	140
6.14.3.1 gen_from_pm1_qm1()	140
6.14.3.2 gen_from_pm4_qp4()	141
6.14.3.3 gen_from_pp4_qm4()	141
6.14.3.4 gen_from_qp1_pm1()	141
6.14.3.5 gen_vector()	141
6.14.3.6 generator()	142

6.14.3.7 operator>()	142
6.14.3.8 operator=()	142
6.14.4 Friends And Related Symbol Documentation	142
6.14.4.1 friend_for_private_destructor	142
6.15 glucat::glucat_error Class Reference	143
6.15.1 Detailed Description	144
6.15.2 Constructor & Destructor Documentation	144
6.15.2.1 glucat_error()	144
6.15.2.2 ~glucat_error()	144
6.15.3 Member Function Documentation	144
6.15.3.1 classname()	144
6.15.3.2 heading()	144
6.15.3.3 print_error_msg()	144
6.15.4 Member Data Documentation	145
6.15.4.1 name	145
6.16 glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t Class Reference	145
6.16.1 Detailed Description	145
6.16.2 Constructor & Destructor Documentation	145
6.16.2.1 hash_size_t()	145
6.16.3 Member Function Documentation	146
6.16.3.1 operator>()	146
6.16.4 Member Data Documentation	146
6.16.4.1 n	146
6.17 glucat::index_set< LO, HI > Class Template Reference	146
6.17.1 Detailed Description	149
6.17.2 Member Typedef Documentation	149
6.17.2.1 bitset_t	149
6.17.2.2 error_t	150
6.17.2.3 index_pair_t	150
6.17.2.4 index_set_t	150
6.17.3 Constructor & Destructor Documentation	150
6.17.3.1 index_set() [1/6]	150
6.17.3.2 index_set() [2/6]	150
6.17.3.3 index_set() [3/6]	150
6.17.3.4 index_set() [4/6]	151
6.17.3.5 index_set() [5/6]	151
6.17.3.6 index_set() [6/6]	151
6.17.4 Member Function Documentation	151
6.17.4.1 BOOST_STATIC_ASSERT()	151
6.17.4.2 classname()	151
6.17.4.3 count()	152
6.17.4.4 count_neg()	152

6.17.4.5	count_pos()	152
6.17.4.6	flip() [1/2]	152
6.17.4.7	flip() [2/2]	152
6.17.4.8	fold() [1/2]	153
6.17.4.9	fold() [2/2]	153
6.17.4.10	hash_fn()	153
6.17.4.11	is_contiguous()	153
6.17.4.12	lex_less_than()	153
6.17.4.13	max()	154
6.17.4.14	min()	154
6.17.4.15	operator!=()	154
6.17.4.16	operator&=()	154
6.17.4.17	operator<()	155
6.17.4.18	operator==()	155
6.17.4.19	operator[]() [1/2]	155
6.17.4.20	operator[]() [2/2]	155
6.17.4.21	operator^=()	155
6.17.4.22	operator" =()	156
6.17.4.23	operator~()	156
6.17.4.24	reset() [1/2]	156
6.17.4.25	reset() [2/2]	156
6.17.4.26	set() [1/3]	156
6.17.4.27	set() [2/3]	157
6.17.4.28	set() [3/3]	157
6.17.4.29	sign_of_mult()	157
6.17.4.30	sign_of_square()	157
6.17.4.31	test()	157
6.17.4.32	unfold()	158
6.17.4.33	value_of_fold()	158
6.17.5	Friends And Related Symbol Documentation	158
6.17.5.1	compare	158
6.17.5.2	operator&	158
6.17.5.3	operator^	158
6.17.5.4	operator" 	159
6.17.5.5	reference	159
6.17.6	Member Data Documentation	159
6.17.6.1	v_hi	159
6.17.6.2	v_lo	159
6.18	PyClical.index_set Class Reference	160
6.18.1	Detailed Description	161
6.18.2	Member Function Documentation	161
6.18.2.1	__and__()	161

6.18.2.2	__cinit__()	161
6.18.2.3	__contains__()	162
6.18.2.4	__dealloc__()	162
6.18.2.5	__getitem__()	162
6.18.2.6	__iand__()	163
6.18.2.7	__invert__()	163
6.18.2.8	__ior__()	163
6.18.2.9	__iter__()	163
6.18.2.10	__ixor__()	164
6.18.2.11	__or__()	164
6.18.2.12	__repr__()	164
6.18.2.13	__richcmp__()	165
6.18.2.14	__setitem__()	165
6.18.2.15	__str__()	165
6.18.2.16	__xor__()	166
6.18.2.17	count()	166
6.18.2.18	count_neg()	166
6.18.2.19	count_pos()	167
6.18.2.20	hash_fn()	167
6.18.2.21	max()	167
6.18.2.22	min()	168
6.18.2.23	sign_of_mult()	168
6.18.2.24	sign_of_square()	168
6.18.3	Member Data Documentation	169
6.18.3.1	instance [1/2]	169
6.18.3.2	instance [2/2]	169
6.19	glucat::index_set_hash< LO, HI > Class Template Reference	169
6.19.1	Detailed Description	170
6.19.2	Member Typedef Documentation	170
6.19.2.1	index_set_t	170
6.19.3	Member Function Documentation	170
6.19.3.1	operator()()	170
6.20	glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > Class Template Reference	170
6.20.1	Detailed Description	175
6.20.2	Member Typedef Documentation	176
6.20.2.1	basis_matrix_t	176
6.20.2.2	error_t	176
6.20.2.3	framed_multi_t	176
6.20.2.4	index_set_t	176
6.20.2.5	matrix_index_t	176
6.20.2.6	matrix_multi_t	176
6.20.2.7	matrix_t	177

6.20.2.8 multivector_t	177
6.20.2.9 orientation_t	177
6.20.2.10 scalar_t	177
6.20.2.11 term_t	177
6.20.2.12 tune_p	177
6.20.2.13 vector_t	178
6.20.3 Constructor & Destructor Documentation	178
6.20.3.1 ~matrix_multi()	178
6.20.3.2 matrix_multi() [1/17]	178
6.20.3.3 matrix_multi() [2/17]	178
6.20.3.4 matrix_multi() [3/17]	178
6.20.3.5 matrix_multi() [4/17]	179
6.20.3.6 matrix_multi() [5/17]	179
6.20.3.7 matrix_multi() [6/17]	179
6.20.3.8 matrix_multi() [7/17]	179
6.20.3.9 matrix_multi() [8/17]	180
6.20.3.10 matrix_multi() [9/17]	180
6.20.3.11 matrix_multi() [10/17]	180
6.20.3.12 matrix_multi() [11/17]	180
6.20.3.13 matrix_multi() [12/17]	181
6.20.3.14 matrix_multi() [13/17]	181
6.20.3.15 matrix_multi() [14/17]	181
6.20.3.16 matrix_multi() [15/17]	181
6.20.3.17 matrix_multi() [16/17]	182
6.20.3.18 matrix_multi() [17/17]	182
6.20.4 Member Function Documentation	182
6.20.4.1 basis_element()	182
6.20.4.2 classname()	182
6.20.4.3 fast_framed_multi()	183
6.20.4.4 fast_matrix_multi()	183
6.20.4.5 operator+=()	183
6.20.4.6 operator=()	183
6.20.4.7 random()	184
6.20.5 Friends And Related Symbol Documentation	184
6.20.5.1 framed_multi	184
6.20.5.2 matrix_log	184
6.20.5.3 matrix_multi	184
6.20.5.4 matrix_sqrt	185
6.20.5.5 operator%	185
6.20.5.6 operator&	185
6.20.5.7 operator*	185
6.20.5.8 operator/	185

6.20.5.9 operator<<	185
6.20.5.10 operator>>	186
6.20.5.11 operator^	186
6.20.5.12 operator"	186
6.20.5.13 reframe	186
6.20.5.14 star	186
6.20.6 Member Data Documentation	187
6.20.6.1 m_frame	187
6.20.6.2 m_matrix	187
6.21 std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > > Struct Template Reference	187
6.21.1 Detailed Description	188
6.22 std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > > Struct Template Reference	189
6.22.1 Detailed Description	189
6.23 glucat::numeric_traits< Scalar_T > Class Template Reference	190
6.23.1 Detailed Description	192
6.23.2 Member Function Documentation	192
6.23.2.1 abs()	192
6.23.2.2 acos()	192
6.23.2.3 asin()	192
6.23.2.4 atan()	193
6.23.2.5 conj()	193
6.23.2.6 cos()	193
6.23.2.7 cosh()	193
6.23.2.8 exp()	193
6.23.2.9 fmod()	194
6.23.2.10 imag()	194
6.23.2.11 isInf() [1/3]	194
6.23.2.12 isInf() [2/3]	194
6.23.2.13 isInf() [3/3]	195
6.23.2.14 isNaN() [1/3]	195
6.23.2.15 isNaN() [2/3]	195
6.23.2.16 isNaN() [3/3]	195
6.23.2.17 isNaN_or_isInf()	196
6.23.2.18 ln_2() [1/2]	196
6.23.2.19 ln_2() [2/2]	196
6.23.2.20 log()	196
6.23.2.21 log2()	197
6.23.2.22 NaN()	197
6.23.2.23 pi() [1/2]	197
6.23.2.24 pi() [2/2]	197
6.23.2.25 pow()	198
6.23.2.26 real()	198

6.23.2.27 <code>sin()</code>	198
6.23.2.28 <code>sinh()</code>	198
6.23.2.29 <code>sqrt()</code>	198
6.23.2.30 <code>tan()</code>	199
6.23.2.31 <code>tanh()</code>	199
6.23.2.32 <code>to_double()</code>	199
6.23.2.33 <code>to_int()</code>	199
6.23.2.34 <code>to_scalar_t()</code> [1/9]	199
6.23.2.35 <code>to_scalar_t()</code> [2/9]	200
6.23.2.36 <code>to_scalar_t()</code> [3/9]	200
6.23.2.37 <code>to_scalar_t()</code> [4/9]	200
6.23.2.38 <code>to_scalar_t()</code> [5/9]	200
6.23.2.39 <code>to_scalar_t()</code> [6/9]	200
6.23.2.40 <code>to_scalar_t()</code> [7/9]	201
6.23.2.41 <code>to_scalar_t()</code> [8/9]	201
6.23.2.42 <code>to_scalar_t()</code> [9/9]	201
6.24 <code>pade::pade_log_denom< Scalar_T ></code> Struct Template Reference	201
6.24.1 Detailed Description	202
6.24.2 Member Typedef Documentation	202
6.24.2.1 <code>array</code>	202
6.24.3 Member Data Documentation	202
6.24.3.1 <code>denom</code>	202
6.25 <code>pade::pade_log_denom< dd_real ></code> Struct Reference	202
6.25.1 Detailed Description	203
6.25.2 Member Typedef Documentation	203
6.25.2.1 <code>array</code>	203
6.25.3 Member Data Documentation	203
6.25.3.1 <code>denom</code>	203
6.26 <code>pade::pade_log_denom< float ></code> Struct Reference	203
6.26.1 Detailed Description	204
6.26.2 Member Typedef Documentation	204
6.26.2.1 <code>array</code>	204
6.26.3 Member Data Documentation	204
6.26.3.1 <code>denom</code>	204
6.27 <code>pade::pade_log_denom< long double ></code> Struct Reference	204
6.27.1 Detailed Description	205
6.27.2 Member Typedef Documentation	205
6.27.2.1 <code>array</code>	205
6.27.3 Member Data Documentation	205
6.27.3.1 <code>denom</code>	205
6.28 <code>pade::pade_log_denom< qd_real ></code> Struct Reference	205
6.28.1 Detailed Description	205

6.28.2 Member Typedef Documentation	206
6.28.2.1 array	206
6.28.3 Member Data Documentation	206
6.28.3.1 denom	206
6.29 <code>pade::pade_log_numer< Scalar_T ></code> Struct Template Reference	206
6.29.1 Detailed Description	207
6.29.2 Member Typedef Documentation	207
6.29.2.1 array	207
6.29.3 Member Data Documentation	207
6.29.3.1 numer	207
6.30 <code>pade::pade_log_numer< dd_real ></code> Struct Reference	208
6.30.1 Detailed Description	208
6.30.2 Member Typedef Documentation	208
6.30.2.1 array	208
6.30.3 Member Data Documentation	208
6.30.3.1 numer	208
6.31 <code>pade::pade_log_numer< float ></code> Struct Reference	209
6.31.1 Detailed Description	209
6.31.2 Member Typedef Documentation	209
6.31.2.1 array	209
6.31.3 Member Data Documentation	209
6.31.3.1 numer	209
6.32 <code>pade::pade_log_numer< long double ></code> Struct Reference	209
6.32.1 Detailed Description	210
6.32.2 Member Typedef Documentation	210
6.32.2.1 array	210
6.32.3 Member Data Documentation	210
6.32.3.1 numer	210
6.33 <code>pade::pade_log_numer< qd_real ></code> Struct Reference	210
6.33.1 Detailed Description	211
6.33.2 Member Typedef Documentation	211
6.33.2.1 array	211
6.33.3 Member Data Documentation	211
6.33.3.1 numer	211
6.34 <code>pade::pade_sqrt_denom< Scalar_T ></code> Struct Template Reference	212
6.34.1 Detailed Description	212
6.34.2 Member Typedef Documentation	212
6.34.2.1 array	212
6.34.3 Member Data Documentation	212
6.34.3.1 denom	212
6.35 <code>pade::pade_sqrt_denom< dd_real ></code> Struct Reference	213
6.35.1 Detailed Description	213

6.35.2 Member Typedef Documentation	213
6.35.2.1 array	213
6.35.3 Member Data Documentation	213
6.35.3.1 denom	213
6.36 pade::pade_sqrt_denom< float > Struct Reference	214
6.36.1 Detailed Description	214
6.36.2 Member Typedef Documentation	214
6.36.2.1 array	214
6.36.3 Member Data Documentation	214
6.36.3.1 denom	214
6.37 pade::pade_sqrt_denom< long double > Struct Reference	214
6.37.1 Detailed Description	215
6.37.2 Member Typedef Documentation	215
6.37.2.1 array	215
6.37.3 Member Data Documentation	215
6.37.3.1 denom	215
6.38 pade::pade_sqrt_denom< qd_real > Struct Reference	215
6.38.1 Detailed Description	216
6.38.2 Member Typedef Documentation	216
6.38.2.1 array	216
6.38.3 Member Data Documentation	216
6.38.3.1 denom	216
6.39 pade::pade_sqrt_numer< Scalar_T > Struct Template Reference	216
6.39.1 Detailed Description	217
6.39.2 Member Typedef Documentation	217
6.39.2.1 array	217
6.39.3 Member Data Documentation	217
6.39.3.1 numer	217
6.40 pade::pade_sqrt_numer< dd_real > Struct Reference	218
6.40.1 Detailed Description	218
6.40.2 Member Typedef Documentation	218
6.40.2.1 array	218
6.40.3 Member Data Documentation	218
6.40.3.1 numer	218
6.41 pade::pade_sqrt_numer< float > Struct Reference	219
6.41.1 Detailed Description	219
6.41.2 Member Typedef Documentation	219
6.41.2.1 array	219
6.41.3 Member Data Documentation	219
6.41.3.1 numer	219
6.42 pade::pade_sqrt_numer< long double > Struct Reference	219
6.42.1 Detailed Description	220

6.42.2 Member Typedef Documentation	220
6.42.2.1 array	220
6.42.3 Member Data Documentation	220
6.42.3.1 numer	220
6.43 pade::pade_sqrt_numer< qd_real > Struct Reference	220
6.43.1 Detailed Description	221
6.43.2 Member Typedef Documentation	221
6.43.2.1 array	221
6.43.3 Member Data Documentation	221
6.43.3.1 numer	221
6.44 glucat::numeric_traits< Scalar_T >::promoted Struct Reference	221
6.44.1 Detailed Description	222
6.44.2 Member Typedef Documentation	222
6.44.2.1 type [1/2]	222
6.44.2.2 type [2/2]	222
6.45 glucat::random_generator< Scalar_T > Class Template Reference	222
6.45.1 Detailed Description	223
6.45.2 Constructor & Destructor Documentation	223
6.45.2.1 random_generator() [1/2]	223
6.45.2.2 random_generator() [2/2]	224
6.45.2.3 ~random_generator()	224
6.45.3 Member Function Documentation	224
6.45.3.1 generator()	224
6.45.3.2 normal()	224
6.45.3.3 operator=()	224
6.45.3.4 uniform()	224
6.45.4 Friends And Related Symbol Documentation	225
6.45.4.1 friend_for_private_destructor	225
6.45.5 Member Data Documentation	225
6.45.5.1 normal_dist	225
6.45.5.2 seed	225
6.45.5.3 uint_gen	225
6.45.5.4 uniform_dist	225
6.46 glucat::index_set< LO, HI >::reference Class Reference	226
6.46.1 Detailed Description	227
6.46.2 Constructor & Destructor Documentation	227
6.46.2.1 reference() [1/2]	227
6.46.2.2 reference() [2/2]	227
6.46.2.3 ~reference()	227
6.46.3 Member Function Documentation	227
6.46.3.1 flip()	227
6.46.3.2 operator bool()	228

6.46.3.3 operator=() [1/2]	228
6.46.3.4 operator=() [2/2]	228
6.46.3.5 operator==()	228
6.46.3.6 operator~()	228
6.46.4 Friends And Related Symbol Documentation	229
6.46.4.1 index_set	229
6.46.5 Member Data Documentation	229
6.46.5.1 m_idx	229
6.46.5.2 m_pst	229
6.47 glucat::sorted_range< Map_T, Sorted_Map_T > Class Template Reference	229
6.47.1 Detailed Description	230
6.47.2 Member Typedef Documentation	230
6.47.2.1 map_t	230
6.47.2.2 sorted_iterator	230
6.47.2.3 sorted_map_t	230
6.47.3 Constructor & Destructor Documentation	230
6.47.3.1 sorted_range()	230
6.47.4 Member Data Documentation	231
6.47.4.1 sorted_begin	231
6.47.4.2 sorted_end	231
6.48 glucat::sorted_range< Sorted_Map_T, Sorted_Map_T > Class Template Reference	231
6.48.1 Detailed Description	232
6.48.2 Member Typedef Documentation	232
6.48.2.1 map_t	232
6.48.2.2 sorted_iterator	232
6.48.2.3 sorted_map_t	232
6.48.3 Constructor & Destructor Documentation	232
6.48.3.1 sorted_range()	232
6.48.4 Member Data Documentation	232
6.48.4.1 sorted_begin	232
6.48.4.2 sorted_end	233
6.49 glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term Class Reference	233
6.49.1 Detailed Description	234
6.49.2 Member Typedef Documentation	234
6.49.2.1 var_pair_t	234
6.49.3 Constructor & Destructor Documentation	234
6.49.3.1 ~var_term()	234
6.49.3.2 var_term() [1/2]	235
6.49.3.3 var_term() [2/2]	235
6.49.4 Member Function Documentation	235
6.49.4.1 classname()	235
6.49.4.2 operator*==()	235

7 File Documentation	237
7.1 glucat/clifford_algebra.h File Reference	237
7.1.1 Macro Definition Documentation	245
7.1.1.1 _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS	245
7.2 clifford_algebra.h	245
7.3 glucat/clifford_algebra_imp.h File Reference	254
7.4 clifford_algebra_imp.h	261
7.5 glucat/errors.h File Reference	273
7.6 errors.h	275
7.7 glucat/errors_imp.h File Reference	275
7.8 errors_imp.h	276
7.9 glucat/framed_multi.h File Reference	277
7.10 framed_multi.h	280
7.11 glucat/framed_multi_imp.h File Reference	284
7.11.1 Macro Definition Documentation	286
7.11.1.1 _GLUCAT_HASH_N	286
7.11.1.2 _GLUCAT_HASH_SIZE_T	286
7.12 framed_multi_imp.h	286
7.13 glucat/generation.h File Reference	307
7.14 generation.h	308
7.15 glucat/generation_imp.h File Reference	309
7.16 generation_imp.h	310
7.17 glucat/global.h File Reference	313
7.17.1 Macro Definition Documentation	314
7.17.1.1 _GLUCAT_CTAssert	314
7.18 global.h	315
7.19 glucat/glucat.h File Reference	316
7.20 glucat.h	317
7.21 glucat/glucat_config.h File Reference	318
7.21.1 Macro Definition Documentation	319
7.21.1.1 GLUCAT_HAVE_CXX11	319
7.21.1.2 GLUCAT_HAVE_INTTYPES_H	319
7.21.1.3 GLUCAT_HAVE_STDINT_H	319
7.21.1.4 GLUCAT_HAVE_STDIO_H	319
7.21.1.5 GLUCAT_HAVE_STDLIB_H	319
7.21.1.6 GLUCAT_HAVE_STRING_H	319
7.21.1.7 GLUCAT_HAVE_STRINGS_H	320
7.21.1.8 GLUCAT_HAVE_SYS_STAT_H	320
7.21.1.9 GLUCAT_HAVE_SYS_TYPES_H	320
7.21.1.10 GLUCAT_HAVE_UNISTD_H	320
7.21.1.11 GLUCAT_PACKAGE	320
7.21.1.12 GLUCAT_PACKAGE_BUGREPORT	320

7.21.1.13 GLUCAT_PACKAGE_NAME	320
7.21.1.14 GLUCAT_PACKAGE_STRING	321
7.21.1.15 GLUCAT_PACKAGE_TARNAME	321
7.21.1.16 GLUCAT_PACKAGE_URL	321
7.21.1.17 GLUCAT_PACKAGE_VERSION	321
7.21.1.18 GLUCAT_STDC_HEADERS	321
7.21.1.19 GLUCAT_VERSION	321
7.22 glucat_config.h	322
7.23 glucat/glucat_imp.h File Reference	323
7.24 glucat_imp.h	324
7.25 glucat/index_set.h File Reference	325
7.26 index_set.h	326
7.27 glucat/index_set_imp.h File Reference	329
7.28 index_set_imp.h	330
7.29 glucat/long_double.h File Reference	342
7.30 long_double.h	343
7.31 glucat/matrix.h File Reference	344
7.32 matrix.h	346
7.33 glucat/matrix_imp.h File Reference	347
7.34 matrix_imp.h	349
7.35 glucat/matrix_multi.h File Reference	357
7.36 matrix_multi.h	359
7.37 glucat/matrix_multi_imp.h File Reference	362
7.38 matrix_multi_imp.h	366
7.39 glucat/portability.h File Reference	390
7.39.1 Macro Definition Documentation	391
7.39.1.1 _GLUCAT_ISINF	391
7.39.1.2 _GLUCAT_ISNAN	391
7.39.1.3 UBLAS_ABS	391
7.39.1.4 UBLAS_SQRT	392
7.40 portability.h	392
7.41 glucat/promotion.h File Reference	392
7.42 promotion.h	394
7.43 glucat/qd.h File Reference	396
7.44 qd.h	397
7.45 glucat/random.h File Reference	401
7.46 random.h	402
7.47 glucat/scalar.h File Reference	403
7.48 scalar.h	404
7.49 glucat/scalar_imp.h File Reference	407
7.50 scalar_imp.h	408
7.51 glucat/tuning.h File Reference	410

7.51.1 Function Documentation	411
7.51.1.1 _GLUCAT_CTAssert()	411
7.52 tuning.h	412
7.53 test/tuning.h File Reference	414
7.54 tuning.h	415
7.55 pyclical/glucat.pxd File Reference	416
7.56 glucat.pxd	416
7.57 pyclical/PyClical.h File Reference	418
7.57.1 Typedef Documentation	419
7.57.1.1 Clifford	419
7.57.1.2 IndexSet	419
7.57.1.3 scalar_t	420
7.57.1.4 String	420
7.57.2 Function Documentation	420
7.57.2.1 clifford_to_repr()	420
7.57.2.2 clifford_to_str()	420
7.57.2.3 index_set_to_repr()	420
7.57.2.4 index_set_to_str()	421
7.57.2.5 PyFloat_FromDouble()	421
7.57.3 Variable Documentation	421
7.57.3.1 epsilon	421
7.57.3.2 glucat_package_version	421
7.57.3.3 hi_ndx	421
7.57.3.4 lo_ndx	421
7.58 PyClical.h	422
7.59 pyclical/PyClical.pxd File Reference	423
7.60 PyClical.pxd	424
7.61 pyclical/PyClical.pyx File Reference	424
7.62 PyClical.pyx	425
7.63 pyclical/PyClical_nocython.cpp File Reference	448
7.63.1 Macro Definition Documentation	448
7.63.1.1 PY_SSIZE_T_CLEAN	448
7.64 PyClical_nocython.cpp	448
7.65 test/control.h File Reference	781
7.66 control.h	782
7.67 test/driver.h File Reference	783
7.68 driver.h	784
7.69 test/timing.h File Reference	784
7.70 timing.h	785
7.71 test/try_catch.h File Reference	786
7.72 try_catch.h	786

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

cga3	Definitions for 3D Conformal Geometric Algebra [DL]	9
glucat		10
glucat::gen		61
glucat::matrix		62
glucat::timing		68
pade		69
PyClical		70
std		76

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

std::bitset	
glucat::index_set< DEFAULT_LO, DEFAULT_HI >	146
glucat::index_set< LO, HI >	146
glucat::bool_to_type< truth_value >	79
cdef	
PyClical.clifford	80
PyClical.index_set	160
Clifford	
PyClical.clifford	80
glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >	99
glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, matrix_multi< double, DEFAULT_LO, DEFAULT_HI, tuning<> > >	99
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >	170
glucat::clifford_algebra< Scalar_T, index_set< LO, HI >, framed_multi< Scalar_T, LO, HI, Tune_P > >	99
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >	121
glucat::compare_types< LHS_T, RHS_T >	109
glucat::compare_types< T, T >	110
glucat::control_t	111
glucat::CTAssertion< bool >	115
glucat::CTAssertion< true >	115
glucat::numeric_traits< Scalar_T >::demoted	116
glucat::matrix::eig_genus< Matrix_T >	116
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t	145
glucat::index_set_hash< LO, HI >	169
IndexSet	
PyClical.index_set	160
inline	
PyClical.clifford	80
PyClical.index_set	160
std::logic_error	
glucat::glucat_error	143
glucat::error< Class_T >	118
std::map	
glucat::basis_table< Scalar_T, LO, HI, Matrix_T >	77
glucat::gen::generator_table< Matrix_T >	138

numeric_limits	
std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > >	187
std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > >	189
glucat::numeric_traits< Scalar_T >	190
obj	
PyClical.clifford	80
PyClical.index_set	160
pade::pade_log_denom< Scalar_T >	201
pade::pade_log_denom< dd_real >	202
pade::pade_log_denom< float >	203
pade::pade_log_denom< long double >	204
pade::pade_log_denom< qd_real >	205
pade::pade_log_numer< Scalar_T >	206
pade::pade_log_numer< dd_real >	208
pade::pade_log_numer< float >	209
pade::pade_log_numer< long double >	209
pade::pade_log_numer< qd_real >	210
pade::pade_sqrt_denom< Scalar_T >	212
pade::pade_sqrt_denom< dd_real >	213
pade::pade_sqrt_denom< float >	214
pade::pade_sqrt_denom< long double >	214
pade::pade_sqrt_denom< qd_real >	215
pade::pade_sqrt_numer< Scalar_T >	216
pade::pade_sqrt_numer< dd_real >	218
pade::pade_sqrt_numer< float >	219
pade::pade_sqrt_numer< long double >	219
pade::pade_sqrt_numer< qd_real >	220
std::pair	
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term	233
glucat::numeric_traits< Scalar_T >::promoted	221
glucat::random_generator< Scalar_T >	222
glucat::index_set< LO, HI >::reference	226
glucat::sorted_range< Map_T, Sorted_Map_T >	229
glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >	231
toClifford	
PyClical.clifford	80
toIndexSet	
PyClical.index_set	160
std::unordered_map	
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >	121

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

glucat::basis_table< Scalar_T, LO, HI, Matrix_T >	77
Table of basis elements used as a cache by basis_element()	
glucat::bool_to_type< truth_value >	79
Bool to type	
PyClical.clifford	80
glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >	
Clifford_algebra<> declares the operations of a Clifford algebra	99
glucat::compare_types< LHS_T, RHS_T >	
Type comparison	109
glucat::compare_types< T, T >	110
glucat::control_t	
Parameters to control tests	111
glucat::CTAssertion< bool >	
Compile time assertion	115
glucat::CTAssertion< true >	115
glucat::numeric_traits< Scalar_T >::demoted	
Demoted type for long double	116
glucat::matrix::eig_genus< Matrix_T >	
Structure containing classification of eigenvalues	116
glucat::error< Class_T >	
Specific exception class	118
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >	
A framed_multi<Scalar_T,LO,HI,Tune_P> is a framed approximation to a multivector	121
glucat::gen::generator_table< Matrix_T >	
Table of generators for specific signatures	138
glucat::glucat_error	
Abstract exception class	143
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t	145
glucat::index_set< LO, HI >	
Index set class based on std::bitset<> in Gnu standard C++ library	146
PyClical.index_set	160
glucat::index_set_hash< LO, HI >	169
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >	
A matrix_multi<Scalar_T,LO,HI,Tune_P> is a matrix approximation to a multivector	170
std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > >	
Numeric limits for framed_multi inherit limits for the corresponding scalar type	187

<code>std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > ></code>	
Numeric limits for <code>matrix_multi</code> inherit limits for the corresponding scalar type	189
<code>glucat::numeric_traits< Scalar_T ></code>	
Extra traits which extend numeric limits	190
<code>pade::pade_log_denom< Scalar_T ></code>	
Coefficients of denominator polynomials of Pade approximations produced by <code>Pade1(log(1+x),x,n,n)</code>	
201	
<code>pade::pade_log_denom< dd_real ></code>	202
<code>pade::pade_log_denom< float ></code>	203
<code>pade::pade_log_denom< long double ></code>	204
<code>pade::pade_log_denom< qd_real ></code>	205
<code>pade::pade_log_numer< Scalar_T ></code>	
Coefficients of numerator polynomials of Pade approximations produced by <code>Pade1(log(1+x),x,n,n)</code>	
206	
<code>pade::pade_log_numer< dd_real ></code>	208
<code>pade::pade_log_numer< float ></code>	209
<code>pade::pade_log_numer< long double ></code>	209
<code>pade::pade_log_numer< qd_real ></code>	210
<code>pade::pade_sqrt_denom< Scalar_T ></code>	
Coefficients of denominator polynomials of Pade approximations produced by <code>Pade1(sqrt(1+x),x,n,n)</code>	
212	
<code>pade::pade_sqrt_denom< dd_real ></code>	213
<code>pade::pade_sqrt_denom< float ></code>	214
<code>pade::pade_sqrt_denom< long double ></code>	214
<code>pade::pade_sqrt_denom< qd_real ></code>	215
<code>pade::pade_sqrt_numer< Scalar_T ></code>	
Coefficients of numerator polynomials of Pade approximations produced by <code>Pade1(sqrt(1+x),x,n,n)</code>	
216	
<code>pade::pade_sqrt_numer< dd_real ></code>	218
<code>pade::pade_sqrt_numer< float ></code>	219
<code>pade::pade_sqrt_numer< long double ></code>	219
<code>pade::pade_sqrt_numer< qd_real ></code>	220
<code>glucat::numeric_traits< Scalar_T >::promoted</code>	
Extra traits which extend numeric limits	221
<code>glucat::random_generator< Scalar_T ></code>	
Random number generator with single instance per <code>Scalar_T</code>	222
<code>glucat::index_set< LO, HI >::reference</code>	
Index set member reference	226
<code>glucat::sorted_range< Map_T, Sorted_Map_T ></code>	
Sorted range for use with output	229
<code>glucat::sorted_range< Sorted_Map_T, Sorted_Map_T ></code>	231
<code>glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term</code>	
Variable term	233

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

glucat/clifford_algebra.h	237
glucat/clifford_algebra_imp.h	254
glucat/errors.h	273
glucat/errors_imp.h	275
glucat/framed_multi.h	277
glucat/framed_multi_imp.h	284
glucat/generation.h	307
glucat/generation_imp.h	309
glucat/global.h	313
glucat/glucat.h	316
glucat/glucat_config.h	318
glucat/glucat_imp.h	323
glucat/index_set.h	325
glucat/index_set_imp.h	329
glucat/long_double.h	342
glucat/matrix.h	344
glucat/matrix_imp.h	347
glucat/matrix_multi.h	357
glucat/matrix_multi_imp.h	362
glucat/portability.h	390
glucat/promotion.h	392
glucat/qd.h	396
glucat/random.h	401
glucat/scalar.h	403
glucat/scalar_imp.h	407
glucat/tuning.h	410
pyclical/glucat.pxd	416
pyclical/PyClical.h	418
pyclical/PyClical.pxd	423
pyclical/PyClical.pyx	424
pyclical/PyClical_nocython.cpp	448
test/control.h	781
test/driver.h	783
test/timing.h	784
test/try_catch.h	786
test/tuning.h	414

Chapter 5

Namespace Documentation

5.1 cga3 Namespace Reference

Definitions for 3D Conformal Geometric Algebra [DL].

Functions

- `template<typename Multivector_T >`
`Multivector_T cga3 (const Multivector_T &x)`
Convert Euclidean 3D vector to Conformal Geometric Algebra null vector [DL (10.50)].
- `template<typename Multivector_T >`
`Multivector_T cga3std (const Multivector_T &X)`
Convert CGA3 null vector to standard Conformal Geometric Algebra null vector [DL (10.52)].
- `template<typename Multivector_T >`
`Multivector_T agc3 (const Multivector_T &X)`
Convert CGA3 null vector to Euclidean 3D vector [DL (10.50)].

5.1.1 Detailed Description

Definitions for 3D Conformal Geometric Algebra [DL].

5.1.2 Function Documentation

5.1.2.1 agc3()

```
template<typename Multivector_T >
Multivector_T cga3::agc3 (
    const Multivector_T & X) [inline]
```

Convert CGA3 null vector to Euclidean 3D vector [DL (10.50)].

Definition at line 126 of file [PyClical.h](#).

References [cga3std\(\)](#).

5.1.2.2 cga3()

```
template<typename Multivector_T >
Multivector_T cga3::cga3 (
    const Multivector_T & x) [inline]
```

Convert Euclidean 3D vector to Conformal Geometric Algebra null vector [DL (10.50)].

Definition at line 103 of file [PyClical.h](#).

5.1.2.3 cga3std()

```
template<typename Multivector_T >
Multivector_T cga3::cga3std (
    const Multivector_T & X) [inline]
```

Convert CGA3 null vector to standard Conformal Geometric Algebra null vector [DL (10.52)].

Definition at line 114 of file [PyClical.h](#).

Referenced by [agc3\(\)](#).

5.2 glucat Namespace Reference

Namespaces

- namespace [gen](#)
- namespace [matrix](#)
- namespace [timing](#)

Classes

- class [basis_table](#)
Table of basis elements used as a cache by basis_element()
- class [bool_to_type](#)
Bool to type.
- class [clifford_algebra](#)
clifford_algebra<> declares the operations of a Clifford algebra
- class [compare_types](#)
Type comparison.
- class [compare_types< T, T >](#)
- class [control_t](#)
Parameters to control tests.
- struct [CTAssertion](#)
Compile time assertion.
- struct [CTAssertion< true >](#)
- class [error](#)
Specific exception class.
- class [framed_multi](#)
A framed_multi<Scalar_T,LO,HI,Tune_P> is a framed approximation to a multivector.

- class [glucat_error](#)
Abstract exception class.
- class [index_set](#)
Index set class based on `std::bitset<>` in Gnu standard C++ library.
- class [index_set_hash](#)
- class [matrix_multi](#)
A `matrix_multi<Scalar_T,LO,HI,Tune_P>` is a matrix approximation to a multivector.
- class [numeric_traits](#)
Extra traits which extend numeric limits.
- class [random_generator](#)
Random number generator with single instance per `Scalar_T`.
- class [sorted_range](#)
Sorted range for use with output.
- class [sorted_range<Sorted_Map_T,Sorted_Map_T>](#)

Typedefs

- using [index_t](#) = int
Size of `index_t` should be enough to represent LO, HI.
- using [set_value_t](#) = unsigned long
Size of `set_value_t` should be enough to contain `index_set<LO,HI>`
- typedef int(* [intfn](#)) ()
For exception catching: pointer to function returning int.
- typedef int(* [intintfn](#)) (int)
For exception catching: pointer to function of int returning int.
- using [tuning_slow](#)
- using [tuning_naive](#)
- using [tuning_fast](#)

Functions

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, template< typename, const [index_t](#), const [index_t](#), typename > class RHS, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [operator!=](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> bool
Test for inequality of multivectors.
- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [operator!=](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> bool
Test for inequality of multivector and scalar.
- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [operator!=](#) (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> bool
Test for inequality of scalar and multivector.
- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [error_squared_tol](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T
Quadratic norm error tolerance relative to a specific multivector.
- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, template< typename, const [index_t](#), const [index_t](#), typename > class RHS, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [error_squared](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs, const Scalar_T threshold) -> Scalar_T

Relative or absolute error using the quadratic norm.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto approx_equal (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs, const Scalar_T threshold, const Scalar_T tolerance) -> bool`

Test for approximate equality of multivectors.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto approx_equal (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> bool`

Test for approximate equality of multivectors.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator+ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric sum of multivector and scalar.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator+ (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric sum of scalar and multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator+ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric sum.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator- (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric difference of multivector and scalar.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator- (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric difference of scalar and multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator- (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric difference.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator* (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Product of multivector and scalar.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator* (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Product of scalar and multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator* (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric product.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator^ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Outer product.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator& (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Inner product.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator% (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Left contraction.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto star (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> Scalar_T`

Hestenes scalar product.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator/ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Quotient of multivector and scalar.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator/ (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Quotient of scalar and multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator/ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric quotient.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator| (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Transformation via twisted adjoint action.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto inv (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric multiplicative inverse.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, int rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Integer power of multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Multivector power of multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto outer_pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, int rhs) -> const Multivector< Scalar_←
_T, LO, HI, Tune_P >`

Outer product power of multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto scalar (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

Scalar part.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto real (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

Real part: synonym for scalar part.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto imag (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

Imaginary part: deprecated (always 0)

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto pure (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Pure part.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto even (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Even part.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto odd (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Odd part.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto vector_part (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const std::vector< Scalar_T >`

Vector part of multivector, as a vector_t with respect to frame()

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto involute (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Main involution, each {i} is replaced by -{i} in each term, eg. {1}{2} -> (-{2})*(-{1})*

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto reverse (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Reversion, eg. {1}{2} -> {2}*{1}.*

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto conj (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Conjugation, rev o invo == invo o rev.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto quad (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

*Scalar_T quadratic form == (rev(x)*x)(0)*

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [norm](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

Scalar_T norm == sum of norm of coordinates.

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [abs](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

Absolute value == sqrt(norm)

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [max_abs](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T

Maximum of absolute values of components of multivector: multivector infinity norm.

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [complexifier](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Square root of -1 which commutes with all members of the frame of the given multivector.

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [elliptic](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [sqrt](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Square root of multivector with specified complexifier.

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [sqrt](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Square root of multivector.

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [clifford_exp](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Exponential of multivector.

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [log](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Natural logarithm of multivector with specified complexifier.

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [log](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Natural logarithm of multivector.

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [cos](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Cosine of multivector with specified complexifier.

- template<template< typename, const [index_t](#), const [index_t](#), typename > class Multivector, typename Scalar_T , const [index_t](#) LO, const [index_t](#) HI, typename Tune_P >
auto [cos](#) (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >

Inverse hyperbolic sine of multivector with specified complexifier.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto asinh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Inverse hyperbolic sine of multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto tan (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Tangent of multivector with specified complexifier.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto tan (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Tangent of multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto atan (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Inverse tangent of multivector with specified complexifier.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto atan (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Inverse tangent of multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto tanh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Hyperbolic tangent of multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto atanh (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Inverse hyperbolic tangent of multivector with specified complexifier.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto atanh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Inverse hyperbolic tangent of multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`static void check_complex (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false)`

Check that i is a valid complexifier for val.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator* (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`

Geometric product.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator^ (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`

Outer product.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator& (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO,`
`HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`
Inner product.
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator% (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO,`
`HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`
Left contraction.
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto star (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO, HI,`
`Tune_P > &rhs) -> Scalar_T`
Hestenes scalar product.
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator/ (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO,`
`HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`
Geometric quotient.
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator| (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO,`
`HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`
Transformation via twisted adjoint action.
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator>> (std::istream &s, framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::istream &`
Read multivector from input.
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator<< (std::ostream &os, const framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::ostream`
`&`
Write multivector to output.
- `template<typename Scalar_T , const index_t LO, const index_t HI>`
`auto operator<< (std::ostream &os, const std::pair< const index_set< LO, HI >, Scalar_T > &term) ->`
`std::ostream &`
Write term to output.
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto exp (const framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> const framed_multi< Scalar_T, LO, HI,`
`Tune_P >`
Exponential of multivector.
- `template<typename Scalar_T , const index_t LO, const index_t HI>`
`static auto crd_of_mult (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std::pair< const`
`index_set< LO, HI >, Scalar_T > &rhs) -> Scalar_T`
Coordinate of product of terms.
- `template<typename Scalar_T , const index_t LO, const index_t HI>`
`auto operator* (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std::pair< const`
`index_set< LO, HI >, Scalar_T > &rhs) -> const std::pair< const index_set< LO, HI >, Scalar_T >`
Product of terms.
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto sqrt (const framed_multi< Scalar_T, LO, HI, Tune_P > &val, const framed_multi< Scalar_T, LO, HI,`
`Tune_P > &i, bool prechecked) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`
Square root of multivector with specified complexifier.
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto log (const framed_multi< Scalar_T, LO, HI, Tune_P > &val, const framed_multi< Scalar_T, LO, HI,`
`Tune_P > &i, bool prechecked) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`
Natural logarithm of multivector with specified complexifier.
- `template<typename Scalar_T , const index_t LO, const index_t HI>`
`static auto crd_of_mult (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std::pair< const`
`index_set< LO, HI >, Scalar_T > &rhs) -> Scalar_T`

- Coordinate of product of terms.*

 - `_GLUCAT_CTAssert` (std::numeric_limits< unsigned char >::radix==2, CannotDetermineBitsPerChar) const `index_t` BITS_PER_CHAR

If radix of unsigned char is not 2, we can't easily determine number of bits from sizeof.

 - `_GLUCAT_CTAssert` (_GLUCAT_BITS_PER_ULONG==BITS_PER_SET_VALUE, BitsPerULongDoesNotMatchSetValueT) const `index_t` DEFAULT_LO

Default lowest index in an index set.

 - template<typename LHS_T, typename RHS_T >
auto `pos_mod` (LHS_T lhs, RHS_T rhs) -> LHS_T

Modulo function which works reliably for lhs < 0.

 - template<const `index_t` LO, const `index_t` HI>
auto `operator^` (const `index_set`< LO, HI > &lhs, const `index_set`< LO, HI > &rhs) -> const `index_set`< LO, HI >

Symmetric set difference: exclusive or.

 - template<const `index_t` LO, const `index_t` HI>
auto `operator&` (const `index_set`< LO, HI > &lhs, const `index_set`< LO, HI > &rhs) -> const `index_set`< LO, HI >

Set intersection: and.

 - template<const `index_t` LO, const `index_t` HI>
auto `operator|` (const `index_set`< LO, HI > &lhs, const `index_set`< LO, HI > &rhs) -> const `index_set`< LO, HI >

Set union: or.

 - template<const `index_t` LO, const `index_t` HI>
auto `compare` (const `index_set`< LO, HI > &a, const `index_set`< LO, HI > &b) -> int

"lexicographic compare" eg. {3,4,5} is less than {3,7,8}

 - `_GLUCAT_CTAssert` (sizeof(set_value_t) >=sizeof(std::bitset< DEFAULT_HI-DEFAULT_LO >), Default_index_set_too_big_for_value) template< const `index_t` LO

Size of set_value_t should be enough to contain bitset<DEFAULT_HI-DEFAULT_LO>

 - const `index_t` HI auto `operator<<` (std::ostream &os, const `index_set`< LO, HI > &ist) -> std::ostream &
 - template<const `index_t` LO, const `index_t` HI>
auto `operator>>` (std::istream &s, `index_set`< LO, HI > &ist) -> std::istream &

Read in index set.

 - auto `sign_of_square` (`index_t` j) -> int

Square of generator {j}.

 - template<const `index_t` LO, const `index_t` HI>
auto `min_neg` (const `index_set`< LO, HI > &ist) -> `index_t`

Minimum negative index, or 0 if none.

 - template<const `index_t` LO, const `index_t` HI>
auto `max_pos` (const `index_set`< LO, HI > &ist) -> `index_t`

Maximum positive index, or 0 if none.

 - template<const `index_t` LO, const `index_t` HI>
auto `operator<<` (std::ostream &os, const `index_set`< LO, HI > &ist) -> std::ostream &

Write out index set.

 - static auto `inverse_reversed_gray` (unsigned long x) -> unsigned long

Inverse reversed Gray code.

 - static auto `inverse_gray` (unsigned long x) -> unsigned long

Inverse Gray code.

 - template<typename Scalar_T, const `index_t` LO, const `index_t` HI, typename Tune_P >
auto `operator*` (const `matrix_multi`< Scalar_T, LO, HI, Tune_P > &lhs, const `matrix_multi`< Scalar_T, LO, HI, Tune_P > &rhs) -> const `matrix_multi`< Scalar_T, LO, HI, Tune_P >

Geometric product.

 - template<typename Scalar_T, const `index_t` LO, const `index_t` HI, typename Tune_P >
auto `operator^` (const `matrix_multi`< Scalar_T, LO, HI, Tune_P > &lhs, const `matrix_multi`< Scalar_T, LO, HI, Tune_P > &rhs) -> const `matrix_multi`< Scalar_T, LO, HI, Tune_P >

Outer product.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator& (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO,`
`HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >`

Inner product.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator% (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO,`
`HI, Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >`

Left contraction.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto star (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI,`
`Tune_P > &rhs) -> Scalar_T`

Hestenes scalar product.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator/ (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI,`
`Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >`

Geometric quotient.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator| (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI,`
`Tune_P > &rhs) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >`

Transformation via twisted adjoint action.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator>> (std::istream &s, matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::istream &`

Read multivector from input.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto operator<< (std::ostream &os, const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::ostream`
`&`

Write multivector to output.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto reframe (const matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs, const matrix_multi< Scalar_T, LO, HI,`
`Tune_P > &rhs, matrix_multi< Scalar_T, LO, HI, Tune_P > &lhs_reframed, matrix_multi< Scalar_T, LO, HI,`
`Tune_P > &rhs_reframed) -> const index_set< LO, HI >`

Find a common frame for operands of a binary operator.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI,`
`Tune_P > &i, bool prechecked) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >`

Square root of multivector with specified complexifier.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto matrix_sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO,`
`HI, Tune_P > &i, const index_t level) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >`

Square root of multivector with specified complexifier.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO, HI,`
`Tune_P > &i, bool prechecked) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >`

Natural logarithm of multivector with specified complexifier.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto matrix_log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO,`
`HI, Tune_P > &i, const index_t level) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >`

Natural logarithm of multivector with specified complexifier.

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto exp (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> const matrix_multi< Scalar_T, LO, HI,`
`Tune_P >`

Exponential of multivector.

- `auto offset_level (const index_t p, const index_t q) -> index_t`

Determine the log2 dim corresponding to signature p, q.

- template<typename Matrix_Index_T, const [index_t](#) LO, const [index_t](#) HI>
static auto [folded_dim](#) (const [index_set](#)< LO, HI > &sub) -> Matrix_Index_T

Determine the matrix dimension of the fold of a subalegebra.

- template<typename Multivector_T, typename Matrix_T, typename Basis_Matrix_T>
static auto [fast](#) (const Matrix_T &X, [index_t](#) level) -> Multivector_T

Inverse generalized Fast Fourier Transform.

- template<typename Scalar_T, const [index_t](#) LO, const [index_t](#) HI, typename Tune_P, const size_t Size>
static auto [pade_approx](#) (const std::array< Scalar_T, Size > &numer, const std::array< Scalar_T, Size > &denom, const [matrix_multi](#)< Scalar_T, LO, HI, Tune_P > &X) -> const [matrix_multi](#)< Scalar_T, LO, HI, Tune_P >

Pade' approximation.

- template<typename Scalar_T, const [index_t](#) LO, const [index_t](#) HI, typename Tune_P>
static void [db_step](#) ([matrix_multi](#)< Scalar_T, LO, HI, Tune_P > &M, [matrix_multi](#)< Scalar_T, LO, HI, Tune_P > &Y)

Single step of product form of Denman-Beavers square root iteration.

- template<typename Scalar_T, const [index_t](#) LO, const [index_t](#) HI, typename Tune_P>
static auto [db_sqrt](#) (const [matrix_multi](#)< Scalar_T, LO, HI, Tune_P > &val, Scalar_T norm_tol=std::pow(std::numeric_limits< Scalar_T >::epsilon(), 4)) -> const [matrix_multi](#)< Scalar_T, LO, HI, Tune_P >

Product form of Denman-Beavers square root iteration.

- template<typename Scalar_T, const [index_t](#) LO, const [index_t](#) HI, typename Tune_P>
static auto [cr_sqrt](#) (const [matrix_multi](#)< Scalar_T, LO, HI, Tune_P > &val, Scalar_T norm_Y_tol=std::pow(std::numeric_limits< Scalar_T >::epsilon(), 1)) -> const [matrix_multi](#)< Scalar_T, LO, HI, Tune_P >

Cyclic reduction square root iteration.

- template<typename Scalar_T, const [index_t](#) LO, const [index_t](#) HI, typename Tune_P>
static auto [pade_log](#) (const [matrix_multi](#)< Scalar_T, LO, HI, Tune_P > &val) -> const [matrix_multi](#)< Scalar_T, LO, HI, Tune_P >

Pade' approximation of log.

- template<typename Scalar_T, const [index_t](#) LO, const [index_t](#) HI, typename Tune_P>
static auto [cascade_log](#) (const [matrix_multi](#)< Scalar_T, LO, HI, Tune_P > &val) -> const [matrix_multi](#)< Scalar_T, LO, HI, Tune_P >

Incomplete square root cascade and Pade' approximation of log.

- template<typename Scalar_T>
auto [log2](#) (const Scalar_T &x) -> Scalar_T

Log base 2 of scalar.

- template<typename Scalar_T>
auto [to_promote](#) (const Scalar_T &val) -> typename [numeric_traits](#)< Scalar_T >::promoted::type

Cast to promote.

- template<typename Scalar_T>
auto [to_demote](#) (const Scalar_T &val) -> typename [numeric_traits](#)< Scalar_T >::demoted::type

Cast to demote.

- int [try_catch](#) (intfn f)

Exception catching for functions returning int.

- int [try_catch](#) (intintfn f, int arg)

Exception catching for functions of int returning int.

Variables

- const double [MS_PER_S](#) = 1000.0

Timing constant: deprecated here - moved to [test/timing.h](#).

- const [index_t](#) [BITS_PER_SET_VALUE](#) = std::numeric_limits<[set_value_t](#)>::digits

Number of bits in [set_value_t](#).

- const [index_t](#) [DEFAULT_HI](#) = [index_t](#)([BITS_PER_SET_VALUE](#) / 2)
Default highest index in an index set.
- static const long double [I_pi](#) = 3.1415926535897932384626433832795029L
- static const long double [I_ln2](#) = 0.6931471805599453094172321214581766L
- const unsigned int [Tuning_Int_Digits](#) = [std::numeric_limits<int>::digits](#)
- const unsigned int [Tuning_Max_Threshold](#) = 1 << [Tuning_Int_Digits](#)
- const unsigned int [Tuning_Slow_Mult_Matrix_Threshold](#) = [Tuning_Max_Threshold](#)
- const unsigned int [Tuning_Slow_Basis_Max_Count](#) = 0
- const unsigned int [Tuning_Slow_Fast_Size_Threshold](#) = [Tuning_Max_Threshold](#)
- const unsigned int [Tuning_Slow_Inv_Fast_Dim_Threshold](#) = [Tuning_Max_Threshold](#)
- const unsigned int [Tuning_Slow_Products_Size_Threshold](#) = [Tuning_Max_Threshold](#)
- const unsigned int [Tuning_Naive_Mult_Matrix_Threshold](#) = 0
- const unsigned int [Tuning_Naive_Basis_Max_Count](#) = [Tuning_Max_Threshold](#)
- const unsigned int [Tuning_Naive_Fast_Size_Threshold](#) = [Tuning_Max_Threshold](#)
- const unsigned int [Tuning_Naive_Inv_Fast_Dim_Threshold](#) = [Tuning_Max_Threshold](#)
- const unsigned int [Tuning_Fast_Mult_Matrix_Threshold](#) = 0
- const unsigned int [Tuning_Fast_Div_Max_Steps](#) = 0
- const unsigned int [Tuning_Fast_CR_Sqrt_Max_Steps](#) = 256
- const unsigned int [Tuning_Fast_DB_Sqrt_Max_Steps](#) = 256
- const unsigned int [Tuning_Fast_Log_Max_Outer_Steps](#) = 16
- const unsigned int [Tuning_Fast_Log_Max_Inner_Steps](#) = 8
- const unsigned int [Tuning_Fast_Basis_Max_Count](#) = 1
- const unsigned int [Tuning_Fast_Fast_Size_Threshold](#) = 0
- const unsigned int [Tuning_Fast_Inv_Fast_Dim_Threshold](#) = 0
- const unsigned int [Tuning_Fast_Products_Size_Threshold](#) = 0

5.2.1 Typedef Documentation

5.2.1.1 [index_t](#)

using [glucat::index_t](#) = int

Size of [index_t](#) should be enough to represent LO, HI.

Definition at line 77 of file [global.h](#).

5.2.1.2 [intfn](#)

```
typedef int(* glucat::intfn) ()
```

For exception catching: pointer to function returning int.

Definition at line 37 of file [try_catch.h](#).

5.2.1.3 [intintfn](#)

```
typedef int(* glucat::intintfn) (int)
```

For exception catching: pointer to function of int returning int.

Definition at line 40 of file [try_catch.h](#).

5.2.1.4 set_value_t

using `glucat::set_value_t` = unsigned long

Size of `set_value_t` should be enough to contain `index_set<LO,HI>`

Definition at line 79 of file [global.h](#).

5.2.1.5 tuning_fast

using `glucat::tuning_fast`

Initial value:

```
tuning
<
    Tuning_Fast_Mult_Matrix_Threshold,
    Tuning_Fast_Div_Max_Steps,
    Tuning_Fast_CR_Sqrt_Max_Steps,
    Tuning_Fast_DB_Sqrt_Max_Steps,
    Tuning_Fast_Log_Max_Outer_Steps,
    Tuning_Fast_Log_Max_Inner_Steps,
    Tuning_Fast_Basis_Max_Count,
    Tuning_Fast_Fast_Size_Threshold,
    Tuning_Fast_Inv_Fast_Dim_Threshold,
    Tuning_Fast_Products_Size_Threshold,
    Tuning_Default_Denom_Different_Bits,
    Tuning_Default_Extra_Different_Bits,
    Tuning_Default_Function_Precision
>
```

Definition at line 97 of file [tuning.h](#).

5.2.1.6 tuning_naive

using `glucat::tuning_naive`

Initial value:

```
tuning
<
    Tuning_Naive_Mult_Matrix_Threshold,
    Tuning_Default_Div_Max_Steps,
    Tuning_Default_CR_Sqrt_Max_Steps,
    Tuning_Default_DB_Sqrt_Max_Steps,
    Tuning_Default_Log_Max_Outer_Steps,
    Tuning_Default_Log_Max_Inner_Steps,
    Tuning_Naive_Basis_Max_Count,
    Tuning_Naive_Fast_Size_Threshold,
    Tuning_Naive_Inv_Fast_Dim_Threshold,
    Tuning_Default_Products_Size_Threshold,
    Tuning_Default_Denom_Different_Bits,
    Tuning_Default_Extra_Different_Bits,
    Tuning_Default_Function_Precision
>
```

Definition at line 69 of file [tuning.h](#).

5.2.1.7 tuning_slow

using `glucat::tuning_slow`

Initial value:

```
tuning
<
    Tuning_Slow_Mult_Matrix_Threshold,
    Tuning_Default_Div_Max_Steps,
    Tuning_Default_CR_Sqrt_Max_Steps,
    Tuning_Default_DB_Sqrt_Max_Steps,
    Tuning_Default_Log_Max_Outer_Steps,
    Tuning_Default_Log_Max_Inner_Steps,
    Tuning_Slow_Basis_Max_Count,
    Tuning_Slow_Fast_Size_Threshold,
    Tuning_Slow_Inv_Fast_Dim_Threshold,
    Tuning_Slow_Products_Size_Threshold,
    Tuning_Default_Denom_Different_Bits,
    Tuning_Default_Extra_Different_Bits,
    Tuning_Default_Function_Precision
>
```

Definition at line 47 of file `tuning.h`.

5.2.2 Function Documentation

5.2.2.1 _GLUCAT_CTAssert() [1/3]

```
glucat::_GLUCAT_CTAssert (
    _GLUCAT_BITS_PER_ULONG ==BITS_PER_SET_VALUE,
    BitsPerUlongDoesNotMatchSetValueT ) const
```

Default lowest index in an index set.

5.2.2.2 _GLUCAT_CTAssert() [2/3]

```
glucat::_GLUCAT_CTAssert (
    sizeof(set_value_t) >=sizeof(std::bitset< DEFAULT_HI-DEFAULT_LO > ) ,
    Default_index_set_too_big_for_value ) const
```

Size of `set_value_t` should be enough to contain `bitset<DEFAULT_HI-DEFAULT_LO>`

Write out index set

5.2.2.3 _GLUCAT_CTAssert() [3/3]

```
glucat::_GLUCAT_CTAssert (
    std::numeric_limits< unsigned char >::radix ==2,
    CannotDetermineBitsPerChar ) const
```

If radix of unsigned char is not 2, we can't easily determine number of bits from `sizeof`.

Number of bits per char is used to determine number of bits in `set_value_t`

5.2.2.4 abs()

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::abs (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T [inline]
```

Absolute value == sqrt(norm)

Definition at line 577 of file [clifford_algebra_imp.h](#).

References [glucat::numeric_traits< Scalar_T >::sqrt\(\)](#).

Referenced by [PyClical.clifford::abs\(\)](#), [clifford_to_str\(\)](#), [matrix_log\(\)](#), and [matrix_sqrt\(\)](#).

5.2.2.5 acos() [1/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::acos (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P> [inline]
```

Inverse cosine of multivector.

Definition at line 903 of file [clifford_algebra_imp.h](#).

References [acos\(\)](#), and [complexifier\(\)](#).

5.2.2.6 acos() [2/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::acos (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val,
    const Multivector< Scalar_T, LO, HI, Tune_P > & i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Inverse cosine of multivector with specified complexifier.

Definition at line 883 of file [clifford_algebra_imp.h](#).

References [acosh\(\)](#), and [check_complex\(\)](#).

Referenced by [acos\(\)](#).

5.2.2.7 acosh() [1/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::acosh (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P> [inline]
```

Inverse hyperbolic cosine of multivector.

Definition at line 844 of file [clifford_algebra_imp.h](#).

References [acosh\(\)](#), and [complexifier\(\)](#).

5.2.2.8 `acosh()` [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::acosh (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val,
    const Multivector< Scalar_T, LO, HI, Tune_P > & i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Inverse hyperbolic cosine of multivector with specified complexifier.

Definition at line 825 of file [clifford_algebra_imp.h](#).

References [check_complex\(\)](#), [log\(\)](#), [norm\(\)](#), and [sqrt\(\)](#).

Referenced by [acos\(\)](#), and [acosh\(\)](#).

5.2.2.9 `approx_equal()` [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::approx_equal (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> bool [inline]
```

Test for approximate equality of multivectors.

Definition at line 169 of file [clifford_algebra_imp.h](#).

References [approx_equal\(\)](#), and [error_squared_tol\(\)](#).

5.2.2.10 `approx_equal()` [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::approx_equal (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs,
    const Scalar_T threshold,
    const Scalar_T tolerance) -> bool [inline]
```

Test for approximate equality of multivectors.

Definition at line 154 of file [clifford_algebra_imp.h](#).

References [error_squared\(\)](#).

Referenced by [approx_equal\(\)](#), and [matrix_sqrt\(\)](#).

5.2.2.11 asin() [1/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::asin (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P> [inline]
```

Inverse sine of multivector.

Definition at line 1008 of file [clifford_algebra_imp.h](#).

References [asin\(\)](#), and [complexifier\(\)](#).

5.2.2.12 asin() [2/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::asin (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val,
    const Multivector< Scalar_T, LO, HI, Tune_P > & i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Inverse sine of multivector with specified complexifier.

Definition at line 988 of file [clifford_algebra_imp.h](#).

References [asinh\(\)](#), and [check_complex\(\)](#).

Referenced by [asin\(\)](#).

5.2.2.13 asinh() [1/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::asinh (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P> [inline]
```

Inverse hyperbolic sine of multivector.

Definition at line 949 of file [clifford_algebra_imp.h](#).

References [asinh\(\)](#), and [complexifier\(\)](#).

5.2.2.14 asinh() [2/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::asinh (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val,
    const Multivector< Scalar_T, LO, HI, Tune_P > & i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Inverse hyperbolic sine of multivector with specified complexifier.

Definition at line 930 of file [clifford_algebra_imp.h](#).

References [check_complex\(\)](#), [log\(\)](#), [norm\(\)](#), and [sqrt\(\)](#).

Referenced by [asin\(\)](#), and [asinh\(\)](#).

5.2.2.15 atan() [1/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::atan (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P> [inline]
```

Inverse tangent of multivector.

Definition at line 1108 of file [clifford_algebra_imp.h](#).

References [atan\(\)](#), and [complexifier\(\)](#).

5.2.2.16 atan() [2/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::atan (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val,
    const Multivector< Scalar_T, LO, HI, Tune_P > & i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Inverse tangent of multivector with specified complexifier.

Definition at line 1088 of file [clifford_algebra_imp.h](#).

References [atanh\(\)](#), and [check_complex\(\)](#).

Referenced by [atan\(\)](#).

5.2.2.17 atanh() [1/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::atanh (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P> [inline]
```

Inverse hyperbolic tangent of multivector.

Definition at line 1052 of file [clifford_algebra_imp.h](#).

References [atanh\(\)](#), and [complexifier\(\)](#).

5.2.2.18 atanh() [2/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::atanh (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val,
    const Multivector< Scalar_T, LO, HI, Tune_P > & i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Inverse hyperbolic tangent of multivector with specified complexifier.

Definition at line 1035 of file [clifford_algebra_imp.h](#).

References [check_complex\(\)](#), [log\(\)](#), and [norm\(\)](#).

Referenced by [atan\(\)](#), and [atanh\(\)](#).

5.2.2.19 cascade_log()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
static auto glucat::cascade_log (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & val) -> const matrix\_multi<Scalar←
_T,LO,HI,Tune_P>    [static]
```

Incomplete square root cascade and Pade' approximation of log.

Definition at line [1920](#) of file [matrix_multi_imp.h](#).

References [db_step\(\)](#), [epsilon](#), [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan\(\)](#), [norm\(\)](#), and [pade_log\(\)](#).

Referenced by [matrix_log\(\)](#).

5.2.2.20 check_complex()

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
static void glucat::check_complex (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val,
    const Multivector< Scalar_T, LO, HI, Tune_P > & i,
    const bool prechecked = false)    [inline], [static]
```

Check that i is a valid complexifier for val.

Definition at line [652](#) of file [clifford_algebra_imp.h](#).

References [complexifier\(\)](#).

Referenced by [acos\(\)](#), [acosh\(\)](#), [asin\(\)](#), [asinh\(\)](#), [atan\(\)](#), [atanh\(\)](#), [cos\(\)](#), [log\(\)](#), [log\(\)](#), [sin\(\)](#), [sqrt\(\)](#), [sqrt\(\)](#), and [tan\(\)](#).

5.2.2.21 clifford_exp()

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::clifford_exp (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P>
```

Exponential of multivector.

Definition at line [690](#) of file [clifford_algebra_imp.h](#).

References [log2\(\)](#).

Referenced by [exp\(\)](#), and [exp\(\)](#).

5.2.2.22 compare()

```
template<const index_t LO, const index_t HI>
auto glucat::compare (
    const index_set< LO, HI > & a,
    const index_set< LO, HI > & b) -> int [inline]
```

"lexicographic compare" eg. {3,4,5} is less than {3,7,8}

Lexicographic ordering of two sets: -1 if $a < b$, +1 if $a > b$, 0 if $a == b$.

Definition at line 574 of file [index_set_imp.h](#).

5.2.2.23 complexifier()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::complexifier (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P>
```

Square root of -1 which commutes with all members of the frame of the given multivector.

Definition at line 592 of file [clifford_algebra_imp.h](#).

References [pos_mod\(\)](#).

Referenced by [acos\(\)](#), [acosh\(\)](#), [asin\(\)](#), [asinh\(\)](#), [atan\(\)](#), [atanh\(\)](#), [check_complex\(\)](#), [cos\(\)](#), [elliptic\(\)](#), [log\(\)](#), [sin\(\)](#), [sqrt\(\)](#), and [tan\(\)](#).

5.2.2.24 conj()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::conj (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P> [inline]
```

Conjugation, rev o invo == invo o rev.

Definition at line 553 of file [clifford_algebra_imp.h](#).

5.2.2.25 cos() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::cos (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P> [inline]
```

Cosine of multivector.

Definition at line 874 of file [clifford_algebra_imp.h](#).

References [complexifier\(\)](#), and [cos\(\)](#).

5.2.2.26 cos() [2/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::cos (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val,
    const Multivector< Scalar_T, LO, HI, Tune_P > & i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>
```

Cosine of multivector with specified complexifier.

Definition at line 851 of file [clifford_algebra_imp.h](#).

References [check_complex\(\)](#), and [exp\(\)](#).

Referenced by [cos\(\)](#), and [tan\(\)](#).

5.2.2.27 cosh()

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::cosh (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar_←
_T,LO,HI,Tune_P> [inline]
```

Hyperbolic cosine of multivector.

Definition at line 807 of file [clifford_algebra_imp.h](#).

References [exp\(\)](#).

Referenced by [tanh\(\)](#).

5.2.2.28 cr_sqrt()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
static auto glucat::cr_sqrt (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & val,
    Scalar_T norm_Y_tol = std::pow(std::numeric_limits<Scalar_T>::epsilon(), 1)) ->
const matrix\_multi<Scalar_T,LO,HI,Tune_P> [static]
```

Cyclic reduction square root iteration.

Definition at line 1349 of file [matrix_multi_imp.h](#).

References [glucat::numeric_traits< Scalar_T >::NaN\(\)](#), and [norm\(\)](#).

Referenced by [matrix_sqrt\(\)](#).

5.2.2.29 crd_of_mult() [1/2]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI>
static auto glucat::crd_of_mult (
    const std::pair< const index\_set< LO, HI >, Scalar_T > & lhs,
    const std::pair< const index\_set< LO, HI >, Scalar_T > & rhs) -> Scalar_T [inline],
[static]
```

Coordinate of product of terms.

Referenced by [operator%\(\)](#), [operator&\(\)](#), [operator*\(\)](#), and [operator^\(\)](#).

5.2.2.30 crd_of_mult() [2/2]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI>
static auto glucat::crd_of_mult (
    const std::pair< const index\_set< LO, HI >, Scalar_T > & lhs,
    const std::pair< const index\_set< LO, HI >, Scalar_T > & rhs) -> Scalar_T [inline],
[static]
```

Coordinate of product of terms.

Definition at line 1709 of file [framed_multi_imp.h](#).

5.2.2.31 db_sqrt()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
static auto glucat::db_sqrt (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & val,
    Scalar_T norm_tol = std::pow(std::numeric_limits<Scalar_T>::epsilon(), 4)) ->
const matrix\_multi<Scalar_T,LO,HI,Tune_P> [static]
```

Product form of Denman-Beavers square root iteration.

Definition at line 1320 of file [matrix_multi_imp.h](#).

References [db_step\(\)](#), [glucat::numeric_traits< Scalar_T >::NaN\(\)](#), and [norm\(\)](#).

Referenced by [matrix_sqrt\(\)](#).

5.2.2.32 db_step()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
static void glucat::db_step (
    matrix\_multi< Scalar_T, LO, HI, Tune_P > & M,
    matrix\_multi< Scalar_T, LO, HI, Tune_P > & Y) [inline], [static]
```

Single step of product form of Denman-Beavers square root iteration.

Definition at line 1308 of file [matrix_multi_imp.h](#).

References [inv\(\)](#).

Referenced by [cascade_log\(\)](#), and [db_sqrt\(\)](#).

5.2.2.33 elliptic()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::elliptic (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar_←
_T,LO,HI,Tune_P> [inline]
```

Square root of -1 which commutes with all members of the frame of the given multivector The name "elliptic" is now deprecated: use "complexifier" instead.

Definition at line 643 of file [clifford_algebra_imp.h](#).

References [complexifier\(\)](#).

5.2.2.34 error_squared()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::error_squared (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs,
    const Scalar_T threshold) -> Scalar_T [inline]
```

Relative or absolute error using the quadratic norm.

Definition at line 134 of file [clifford_algebra_imp.h](#).

References [norm\(\)](#).

Referenced by [approx_equal\(\)](#).

5.2.2.35 error_squared_tol()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::error_squared_tol (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T
```

Quadratic norm error tolerance relative to a specific multivector.

Definition at line 112 of file [clifford_algebra_imp.h](#).

References [glucat::numeric_traits< Scalar_T >::pow\(\)](#).

Referenced by [approx_equal\(\)](#).

5.2.2.36 even()

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::even (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↵
_T,LO,HI,Tune_P> [inline]
```

Even part.

Definition at line 513 of file [clifford_algebra_imp.h](#).

5.2.2.37 exp() [1/2]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::exp (
    const framed\_multi< Scalar_T, LO, HI, Tune_P > & val) -> const framed\_multi<Scalar↵
_T,LO,HI,Tune_P>
```

Exponential of multivector.

Definition at line 1750 of file [framed_multi_imp.h](#).

References [clifford_exp\(\)](#), [exp\(\)](#), and [scalar\(\)](#).

Referenced by [cos\(\)](#), [cosh\(\)](#), [exp\(\)](#), [matrix_log\(\)](#), [matrix_sqrt\(\)](#), [pow\(\)](#), [sin\(\)](#), and [sinh\(\)](#).

5.2.2.38 exp() [2/2]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::exp (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & val) -> const matrix\_multi<Scalar↵
_T,LO,HI,Tune_P>
```

Exponential of multivector.

Definition at line 2086 of file [matrix_multi_imp.h](#).

References [clifford_exp\(\)](#), [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan\(\)](#), and [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::scalar\(\)](#).

5.2.2.39 fast()

```
template<typename Multivector_T , typename Matrix_T , typename Basis_Matrix_T >
static auto glucat::fast (
    const Matrix_T & X,
    index\_t level) -> Multivector_T [static]
```

Inverse generalized Fast Fourier Transform.

Definition at line 1027 of file [matrix_multi_imp.h](#).

References [fast\(\)](#), [glucat::matrix::signed_perm_nork\(\)](#), and [glucat::matrix::unit\(\)](#).

Referenced by [fast\(\)](#), and [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi\(\)](#).

5.2.2.40 folded_dim()

```
template<typename Matrix_Index_T , const index_t LO, const index_t HI>
static auto glucat::folded_dim (
    const index_set< LO, HI > & sub) -> Matrix_Index_T    [inline], [static]
```

Determine the matrix dimension of the fold of a subalgebra.

Definition at line 101 of file [matrix_multi_imp.h](#).

References [offset_level\(\)](#).

Referenced by [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), and [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#).

5.2.2.41 imag()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::imag (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T    [inline]
```

Imaginary part: deprecated (always 0)

Definition at line 497 of file [clifford_algebra_imp.h](#).

5.2.2.42 inv()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::inv (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P>    [inline]
```

Geometric multiplicative inverse.

Definition at line 400 of file [clifford_algebra_imp.h](#).

Referenced by [db_step\(\)](#), and [matrix_log\(\)](#).

5.2.2.43 inverse_gray()

```
static auto glucat::inverse_gray (
    unsigned long x) -> unsigned long    [inline], [static]
```

Inverse Gray code.

Definition at line 863 of file [index_set_imp.h](#).

Referenced by [glucat::index_set< LO, HI >::sign_of_mult\(\)](#).

5.2.2.44 `inverse_reversed_gray()`

```
static auto glucat::inverse_reversed_gray (
    unsigned long x) -> unsigned long    [inline], [static]
```

Inverse reversed Gray code.

Definition at line 846 of file [index_set_imp.h](#).

Referenced by [glucat::index_set< LO, HI >::sign_of_mult\(\)](#).

5.2.2.45 `involute()`

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::involute (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P>    [inline]
```

Main involution, each {i} is replaced by -{i} in each term, eg. {1}*{2} -> (-{2})*(-{1})

Main involution, each {i} is replaced by -{i} in each term, eg. {1} -> -{1}.

Definition at line 537 of file [clifford_algebra_imp.h](#).

5.2.2.46 `log()` [1/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::log (
    const framed_multi< Scalar_T, LO, HI, Tune_P > & val,
    const framed_multi< Scalar_T, LO, HI, Tune_P > & i,
    bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
```

Natural logarithm of multivector with specified complexifier.

Definition at line 1800 of file [framed_multi_imp.h](#).

References [check_complex\(\)](#), and [log\(\)](#).

5.2.2.47 `log()` [2/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::log (
    const matrix_multi< Scalar_T, LO, HI, Tune_P > & val,
    const matrix_multi< Scalar_T, LO, HI, Tune_P > & i,
    bool prechecked) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
```

Natural logarithm of multivector with specified complexifier.

Definition at line 2045 of file [matrix_multi_imp.h](#).

References [check_complex\(\)](#), [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan\(\)](#), and [matrix_log\(\)](#).

5.2.2.48 log() [3/4]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::log (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar_
_T,LO,HI,Tune_P> [inline]
```

Natural logarithm of multivector.

Definition at line 799 of file [clifford_algebra_imp.h](#).

References [complexifier\(\)](#), and [log\(\)](#).

5.2.2.49 log() [4/4]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::log (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val,
    const Multivector< Scalar_T, LO, HI, Tune_P > & i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Natural logarithm of multivector with specified complexifier.

Definition at line 791 of file [clifford_algebra_imp.h](#).

References [log\(\)](#).

Referenced by [acosh\(\)](#), [asinh\(\)](#), [atanh\(\)](#), [log\(\)](#), [log\(\)](#), [log\(\)](#), and [pow\(\)](#).

5.2.2.50 log2()

```
template<typename Scalar_T >
auto glucat::log2 (
    const Scalar_T & x) -> Scalar_T [inline]
```

Log base 2 of scalar.

Definition at line 303 of file [scalar.h](#).

References [glucat::numeric_traits< Scalar_T >::log2\(\)](#).

Referenced by [clifford_exp\(\)](#).

5.2.2.51 matrix_log()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::matrix_log (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & val,
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & i,
    const index\_t level) -> const matrix\_multi<Scalar_T,LO,HI,Tune_P>
```

Natural logarithm of multivector with specified complexifier.

Definition at line 1967 of file [matrix_multi_imp.h](#).

References [abs\(\)](#), [cascade_log\(\)](#), [glucat::matrix::classify_eigenvalues\(\)](#), [exp\(\)](#), [inv\(\)](#), [glucat::clifford_algebra< Scalar_T, Index_Set_T, matrix_log\(\), norm\(\), and glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::scalar\(\)](#).

Referenced by [log\(\)](#), and [matrix_log\(\)](#).

5.2.2.52 `matrix_sqrt()`

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::matrix_sqrt (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & val,
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & i,
    const index\_t level) -> const matrix\_multi<Scalar_T,LO,HI,Tune_P>
```

Square root of multivector with specified complexifier.

Definition at line [1571](#) of file [matrix_multi_imp.h](#).

References [abs\(\)](#), [approx_equal\(\)](#), [glucat::matrix::classify_eigenvalues\(\)](#), [cr_sqrt\(\)](#), [db_sqrt\(\)](#), [exp\(\)](#), [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::matrix_sqrt\(\)](#), [norm\(\)](#), [pade_approx\(\)](#), [pow\(\)](#), and [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::scalar\(\)](#).

Referenced by [matrix_sqrt\(\)](#), and [sqrt\(\)](#).

5.2.2.53 `max_abs()`

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::max_abs (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T [inline]
```

Maximum of absolute values of components of multivector: multivector infinity norm.

Definition at line [585](#) of file [clifford_algebra_imp.h](#).

5.2.2.54 `max_pos()`

```
template<const index\_t LO, const index\_t HI>
auto glucat::max_pos (
    const index\_set< LO, HI > & ist) -> index\_t [inline]
```

Maximum positive index, or 0 if none.

Definition at line [977](#) of file [index_set_imp.h](#).

5.2.2.55 `min_neg()`

```
template<const index\_t LO, const index\_t HI>
auto glucat::min_neg (
    const index\_set< LO, HI > & ist) -> index\_t [inline]
```

Minimum negative index, or 0 if none.

Definition at line [970](#) of file [index_set_imp.h](#).

5.2.2.56 norm()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::norm (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T [inline]
```

Scalar_T norm == sum of norm of coordinates.

Definition at line 569 of file [clifford_algebra_imp.h](#).

Referenced by [acosh\(\)](#), [asinh\(\)](#), [atanh\(\)](#), [cascade_log\(\)](#), [cr_sqrt\(\)](#), [db_sqrt\(\)](#), [error_squared\(\)](#), [matrix_log\(\)](#), and [matrix_sqrt\(\)](#).

5.2.2.57 odd()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::odd (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P> [inline]
```

Odd part.

Definition at line 521 of file [clifford_algebra_imp.h](#).

Referenced by [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast\(\)](#).

5.2.2.58 offset_level()

```
auto glucat::offset_level (
    const index_t p,
    const index_t q) -> index_t [inline]
```

Determine the log2 dim corresponding to signature p, q.

Definition at line 86 of file [matrix_multi_imp.h](#).

References [pos_mod\(\)](#).

Referenced by [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_element\(\)](#), and [folded_dim\(\)](#).

5.2.2.59 operator!=() [1/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator!=(
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> bool [inline]
```

Test for inequality of multivectors.

Definition at line 86 of file [clifford_algebra_imp.h](#).

5.2.2.60 operator"!="() [2/3]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator!=(
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const Scalar_T & scr) -> bool [inline]
```

Test for inequality of multivector and scalar.

Definition at line 94 of file [clifford_algebra_imp.h](#).

5.2.2.61 operator"!="() [3/3]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator!=(
    const Scalar_T & scr,
    const Multivector< Scalar_T, LO, HI, Tune_P > & rhs) -> bool [inline]
```

Test for inequality of scalar and multivector.

Definition at line 102 of file [clifford_algebra_imp.h](#).

5.2.2.62 operator%() [1/3]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator% (
    const framed\_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const framed\_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const framed\_multi<Scalar_T,LO,HI,Tune_P>
```

Left contraction.

Definition at line 597 of file [framed_multi_imp.h](#).

References [_GLUCAT_HASH_SIZE_T](#), and [crd_of_mult\(\)](#).

5.2.2.63 operator%() [2/3]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator% (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const matrix\_multi<Scalar_T,LO,HI,Tune_P> [inline]
```

Left contraction.

Definition at line 581 of file [matrix_multi_imp.h](#).

5.2.2.64 operator%() [3/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator% (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Left contraction.

Definition at line 322 of file [clifford_algebra_imp.h](#).

5.2.2.65 operator&() [1/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator& (
    const framed_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const framed_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
```

Inner product.

Definition at line 495 of file [framed_multi_imp.h](#).

References [_GLUCAT_HASH_SIZE_T](#), and [crd_of_mult\(\)](#).

5.2.2.66 operator&() [2/4]

```
template<const index_t LO, const index_t HI>
auto glucat::operator& (
    const index_set< LO, HI > & lhs,
    const index_set< LO, HI > & rhs) -> const index_set<LO,HI> [inline]
```

Set intersection: and.

Definition at line 186 of file [index_set_imp.h](#).

5.2.2.67 operator&() [3/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator& (
    const matrix_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const matrix_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P> [inline]
```

Inner product.

Definition at line 562 of file [matrix_multi_imp.h](#).

5.2.2.68 operator&() [4/4]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator& (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↔
T,LO,HI,Tune_P> [inline]
```

Inner product.

Definition at line 307 of file [clifford_algebra_imp.h](#).

5.2.2.69 operator*() [1/6]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (
    const framed_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const framed_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const framed_multi<Scalar_↔
_T,LO,HI,Tune_P>
```

Geometric product.

Definition at line 374 of file [framed_multi_imp.h](#).

References [_GLUCAT_HASH_SIZE_T](#).

5.2.2.70 operator*() [2/6]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (
    const matrix_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const matrix_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const matrix_multi<Scalar_↔
_T,LO,HI,Tune_P> [inline]
```

Geometric product.

Definition at line 502 of file [matrix_multi_imp.h](#).

References [glucat::numeric_traits< Scalar_T >::NaN\(\)](#), and [reframe\(\)](#).

5.2.2.71 operator*() [3/6]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↔
T,LO,HI,Tune_P> [inline]
```

Geometric product.

Definition at line 277 of file [clifford_algebra_imp.h](#).

5.2.2.72 operator*() [4/6]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const Scalar_T & scr) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Product of multivector and scalar.

Definition at line 251 of file [clifford_algebra_imp.h](#).

5.2.2.73 operator*() [5/6]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator* (
    const Scalar_T & scr,
    const Multivector< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↔
_T,LO,HI,Tune_P> [inline]
```

Product of scalar and multivector.

Definition at line 262 of file [clifford_algebra_imp.h](#).

5.2.2.74 operator*() [6/6]

```
template<typename Scalar_T , const index_t LO, const index_t HI>
auto glucat::operator* (
    const std::pair< const index_set< LO, HI >, Scalar_T > & lhs,
    const std::pair< const index_set< LO, HI >, Scalar_T > & rhs) -> const std::↔
::pair<const index_set<LO,HI>, Scalar_T> [inline]
```

Product of terms.

Definition at line 1717 of file [framed_multi_imp.h](#).

References [crd_of_mult\(\)](#).

5.2.2.75 operator+() [1/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator+ (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↔
_T,LO,HI,Tune_P> [inline]
```

Geometric sum.

Definition at line 206 of file [clifford_algebra_imp.h](#).

5.2.2.76 operator+() [2/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator+ (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const Scalar_T & scr) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Geometric sum of multivector and scalar.

Definition at line 181 of file [clifford_algebra_imp.h](#).

5.2.2.77 operator+() [3/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator+ (
    const Scalar_T & scr,
    const Multivector< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_←
_T,LO,HI,Tune_P> [inline]
```

Geometric sum of scalar and multivector.

Definition at line 192 of file [clifford_algebra_imp.h](#).

5.2.2.78 operator-() [1/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator- (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_←
T,LO,HI,Tune_P> [inline]
```

Geometric difference.

Definition at line 240 of file [clifford_algebra_imp.h](#).

5.2.2.79 operator-() [2/3]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator- (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const Scalar_T & scr) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Geometric difference of multivector and scalar.

Definition at line 217 of file [clifford_algebra_imp.h](#).

5.2.2.80 operator-() [3/3]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator- (
    const Scalar_T & scr,
    const Multivector< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↵
_T,LO,HI,Tune_P> [inline]
```

Geometric difference of scalar and multivector.

Definition at line 228 of file [clifford_algebra_imp.h](#).

5.2.2.81 operator/() [1/5]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator/ (
    const framed\_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const framed\_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const framed\_multi<Scalar_↵
_T,LO,HI,Tune_P> [inline]
```

Geometric quotient.

Definition at line 734 of file [framed_multi_imp.h](#).

5.2.2.82 operator/() [2/5]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator/ (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const matrix\_multi<Scalar_↵
_T,LO,HI,Tune_P>
```

Geometric quotient.

Definition at line 614 of file [matrix_multi_imp.h](#).

References [glucat::matrix::isnan\(\)](#), and [reframe\(\)](#).

5.2.2.83 operator/() [3/5]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
template< typename, const index\_t, const index\_t, typename > class RHS, typename Scalar_T ,
const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator/ (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↵
_T,LO,HI,Tune_P> [inline]
```

Geometric quotient.

Definition at line 374 of file [clifford_algebra_imp.h](#).

5.2.2.84 operator/() [4/5]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator/ (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const Scalar_T & scr) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Quotient of multivector and scalar.

Definition at line 348 of file [clifford_algebra_imp.h](#).

5.2.2.85 operator/() [5/5]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator/ (
    const Scalar_T & scr,
    const Multivector< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_←
_T,LO,HI,Tune_P> [inline]
```

Quotient of scalar and multivector.

Definition at line 359 of file [clifford_algebra_imp.h](#).

5.2.2.86 operator<<() [1/5]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator<< (
    std::ostream & os,
    const framed\_multi< Scalar_T, LO, HI, Tune_P > & val) -> std::ostream&
```

Write multivector to output.

Definition at line 1148 of file [framed_multi_imp.h](#).

References [scalar\(\)](#), and [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::truncated\(\)](#).

5.2.2.87 operator<<() [2/5]

```
const index\_t HI auto glucat::operator<< (
    std::ostream & os,
    const index\_set< LO, HI > & ist) -> std::ostream &
```

5.2.2.88 operator<<() [3/5]

```
template<const index\_t LO, const index\_t HI>
auto glucat::operator<< (
    std::ostream & os,
    const index\_set< LO, HI > & ist) -> std::ostream&
```

Write out index set.

Definition at line 611 of file [index_set_imp.h](#).

5.2.2.89 operator<<() [4/5]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator<< (
    std::ostream & os,
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & val) -> std::ostream& [inline]
```

Write multivector to output.

Definition at line 955 of file [matrix_multi_imp.h](#).

5.2.2.90 operator<<() [5/5]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI>
auto glucat::operator<< (
    std::ostream & os,
    const std::pair< const index\_set< LO, HI >, Scalar_T > & term) -> std::ostream&
```

Write term to output.

Definition at line 1209 of file [framed_multi_imp.h](#).

References [glucat::numeric_traits< Scalar_T >::to_double\(\)](#).

5.2.2.91 operator>>() [1/3]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator>> (
    std::istream & s,
    framed\_multi< Scalar_T, LO, HI, Tune_P > & val) -> std::istream&
```

Read multivector from input.

Definition at line 1248 of file [framed_multi_imp.h](#).

5.2.2.92 operator>>() [2/3]

```
template<const index\_t LO, const index\_t HI>
auto glucat::operator>> (
    std::istream & s,
    index\_set< LO, HI > & ist) -> std::istream&
```

Read in index set.

Definition at line 634 of file [index_set_imp.h](#).

5.2.2.93 operator>>() [3/3]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator>> (
    std::istream & s,
    matrix\_multi< Scalar_T, LO, HI, Tune_P > & val) -> std::istream& [inline]
```

Read multivector from input.

Definition at line 966 of file [matrix_multi_imp.h](#).

5.2.2.94 operator[^]() [1/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator^ (
    const framed_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const framed_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
```

Outer product.

Definition at line 416 of file [framed_multi_imp.h](#).

References [_GLUCAT_HASH_SIZE_T](#), and [crd_of_mult\(\)](#).

5.2.2.95 operator[^]() [2/4]

```
template<const index_t LO, const index_t HI>
auto glucat::operator^ (
    const index_set< LO, HI > & lhs,
    const index_set< LO, HI > & rhs) -> const index_set<LO,HI> [inline]
```

Symmetric set difference: exclusive or.

Definition at line 161 of file [index_set_imp.h](#).

5.2.2.96 operator[^]() [3/4]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator^ (
    const matrix_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const matrix_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P> [inline]
```

Outer product.

Definition at line 543 of file [matrix_multi_imp.h](#).

5.2.2.97 operator[^]() [4/4]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::operator^ (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Outer product.

Definition at line 292 of file [clifford_algebra_imp.h](#).

5.2.2.98 operator" | () [1/4]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator| (
    const framed\_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const framed\_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const framed\_multi<Scalar_↵
_T,LO,HI,Tune_P> [inline]
```

Transformation via twisted adjoint action.

Definition at line 760 of file [framed_multi_imp.h](#).

5.2.2.99 operator" | () [2/4]

```
template<const index\_t LO, const index\_t HI>
auto glucat::operator| (
    const index\_set< LO, HI > & lhs,
    const index\_set< LO, HI > & rhs) -> const index\_set<LO,HI> [inline]
```

Set union: or.

Definition at line 211 of file [index_set_imp.h](#).

5.2.2.100 operator" | () [3/4]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator| (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> const matrix\_multi<Scalar_↵
_T,LO,HI,Tune_P> [inline]
```

Transformation via twisted adjoint action.

Definition at line 717 of file [matrix_multi_imp.h](#).

5.2.2.101 operator" | () [4/4]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
template< typename, const index\_t, const index\_t, typename > class RHS, typename Scalar_T ,
const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::operator| (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_↵
_T,LO,HI,Tune_P> [inline]
```

Transformation via twisted adjoint action.

Definition at line 389 of file [clifford_algebra_imp.h](#).

5.2.2.102 `outer_pow()`

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::outer_pow (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    int rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
```

Outer product power of multivector.

Definition at line 470 of file [clifford_algebra_imp.h](#).

5.2.2.103 `pade_approx()`

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P , const size_t Size>
static auto glucat::pade_approx (
    const std::array< Scalar_T, Size > & numer,
    const std::array< Scalar_T, Size > & denom,
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & X) -> const matrix\_multi<Scalar_T,LO,HI,Tune_P> [inline], [static]
```

Pade' approximation.

Definition at line 1245 of file [matrix_multi_imp.h](#).

Referenced by [matrix_sqrt\(\)](#), and [pade_log\(\)](#).

5.2.2.104 `pade_log()`

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
static auto glucat::pade_log (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & val) -> const matrix\_multi<Scalar_T,LO,HI,Tune_P> [static]
```

Pade' approximation of log.

Definition at line 1900 of file [matrix_multi_imp.h](#).

References [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan\(\)](#), and [pade_approx\(\)](#).

Referenced by [cascade_log\(\)](#).

5.2.2.105 `pos_mod()`

```
template<typename LHS_T , typename RHS_T >
auto glucat::pos_mod (
    LHS_T lhs,
    RHS_T rhs) -> LHS_T [inline]
```

Modulo function which works reliably for lhs < 0.

Definition at line 117 of file [global.h](#).

Referenced by [complexifier\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi\(\)](#), [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi\(\)](#), [glucat::gen::generator_table< Matrix_T >::gen_vector\(\)](#), [offset_level\(\)](#), and [glucat::gen::generator_table< Matrix_T >::operator\(\)\(\)](#).

5.2.2.106 pow() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T ,
const index_t LO, const index_t HI, typename Tune_P >
auto glucat::pow (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> const Multivector<Scalar_T,
T,LO,HI,Tune_P> [inline]
```

Multivector power of multivector.

Definition at line 446 of file [clifford_algebra_imp.h](#).

References [exp\(\)](#), and [log\(\)](#).

5.2.2.107 pow() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::pow (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    int rhs) -> const Multivector<Scalar_T,LO,HI,Tune_P>
```

Integer power of multivector.

Definition at line 407 of file [clifford_algebra_imp.h](#).

Referenced by [matrix_sqrt\(\)](#).

5.2.2.108 pure()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::pure (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar_
_T,LO,HI,Tune_P> [inline]
```

Pure part.

Definition at line 505 of file [clifford_algebra_imp.h](#).

5.2.2.109 quad()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::quad (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T [inline]
```

Scalar_T quadratic form == (rev(x)*x)(0)

Definition at line 561 of file [clifford_algebra_imp.h](#).

5.2.2.110 real()

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::real (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T [inline]
```

Real part: synonym for scalar part.

Definition at line [486](#) of file [clifford_algebra_imp.h](#).

5.2.2.111 reframe()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::reframe (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & rhs,
    matrix\_multi< Scalar_T, LO, HI, Tune_P > & lhs_reframed,
    matrix\_multi< Scalar_T, LO, HI, Tune_P > & rhs_reframed) -> const index\_set<LO,HI>
[inline]
```

Find a common frame for operands of a binary operator.

Definition at line [345](#) of file [matrix_multi_imp.h](#).

Referenced by [operator*\(\)](#), and [operator/\(\)](#).

5.2.2.112 reverse()

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::reverse (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar↔
_T,LO,HI,Tune_P> [inline]
```

Reversion, eg. $\{1\}*\{2\} \rightarrow \{2\}*\{1\}$.

Definition at line [545](#) of file [clifford_algebra_imp.h](#).

5.2.2.113 scalar()

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::scalar (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> Scalar_T [inline]
```

Scalar part.

Definition at line [478](#) of file [clifford_algebra_imp.h](#).

Referenced by [exp\(\)](#), [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast\(\)](#), and [operator<<\(\)](#).

5.2.2.114 sign_of_square()

```
auto glucat::sign_of_square (
    index_t j) -> int [inline]
```

Square of generator {j}.

Square of generator index j.

Definition at line 963 of file [index_set_imp.h](#).

5.2.2.115 sin() [1/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sin (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P> [inline]
```

Sine of multivector.

Definition at line 979 of file [clifford_algebra_imp.h](#).

References [complexifier\(\)](#), and [sin\(\)](#).

5.2.2.116 sin() [2/2]

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sin (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val,
    const Multivector< Scalar_T, LO, HI, Tune_P > & i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>
```

Sine of multivector with specified complexifier.

Definition at line 956 of file [clifford_algebra_imp.h](#).

References [check_complex\(\)](#), and [exp\(\)](#).

Referenced by [sin\(\)](#), and [tan\(\)](#).

5.2.2.117 sinh()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::sinh (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar←
_T,LO,HI,Tune_P> [inline]
```

Hyperbolic sine of multivector.

Definition at line 911 of file [clifford_algebra_imp.h](#).

References [exp\(\)](#).

Referenced by [tanh\(\)](#).

5.2.2.118 sqrt() [1/4]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::sqrt (
    const framed\_multi< Scalar_T, LO, HI, Tune_P > & val,
    const framed\_multi< Scalar_T, LO, HI, Tune_P > & i,
    bool prechecked) -> const framed\_multi<Scalar_T,LO,HI,Tune_P>
```

Square root of multivector with specified complexifier.

Definition at line 1727 of file [framed_multi_imp.h](#).

References [check_complex\(\)](#), and [sqrt\(\)](#).

5.2.2.119 sqrt() [2/4]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::sqrt (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & val,
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & i,
    bool prechecked) -> const matrix\_multi<Scalar_T,LO,HI,Tune_P>
```

Square root of multivector with specified complexifier.

Definition at line 1667 of file [matrix_multi_imp.h](#).

References [check_complex\(\)](#), [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan\(\)](#), and [matrix_sqrt\(\)](#).

5.2.2.120 sqrt() [3/4]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::sqrt (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar_←
_T,LO,HI,Tune_P> [inline]
```

Square root of multivector.

Definition at line 683 of file [clifford_algebra_imp.h](#).

References [complexifier\(\)](#), and [sqrt\(\)](#).

5.2.2.121 sqrt() [4/4]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::sqrt (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val,
    const Multivector< Scalar_T, LO, HI, Tune_P > & i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Square root of multivector with specified complexifier.

Definition at line 675 of file [clifford_algebra_imp.h](#).

References [sqrt\(\)](#).

Referenced by [acosh\(\)](#), [asinh\(\)](#), [sqrt\(\)](#), [sqrt\(\)](#), and [sqrt\(\)](#).

5.2.2.122 star() [1/3]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::star (
    const framed\_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const framed\_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> Scalar_T
```

Hestenes scalar product.

Definition at line [684](#) of file [framed_multi_imp.h](#).

5.2.2.123 star() [2/3]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::star (
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & lhs,
    const matrix\_multi< Scalar_T, LO, HI, Tune_P > & rhs) -> Scalar_T [inline]
```

Hestenes scalar product.

Definition at line [600](#) of file [matrix_multi_imp.h](#).

5.2.2.124 star() [3/3]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
template< typename, const index\_t, const index\_t, typename > class RHS, typename Scalar_T ,
const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::star (
    const Multivector< Scalar_T, LO, HI, Tune_P > & lhs,
    const RHS< Scalar_T, LO, HI, Tune_P > & rhs) -> Scalar_T [inline]
```

Hestenes scalar product.

Definition at line [337](#) of file [clifford_algebra_imp.h](#).

References [star\(\)](#).

Referenced by [star\(\)](#).

5.2.2.125 tan() [1/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::tan (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar_←
_T,LO,HI,Tune_P> [inline]
```

Tangent of multivector.

Definition at line [1079](#) of file [clifford_algebra_imp.h](#).

References [complexifier\(\)](#), and [tan\(\)](#).

5.2.2.126 tan() [2/2]

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::tan (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val,
    const Multivector< Scalar_T, LO, HI, Tune_P > & i,
    const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P> [inline]
```

Tangent of multivector with specified complexifier.

Definition at line 1060 of file [clifford_algebra_imp.h](#).

References [check_complex\(\)](#), [cos\(\)](#), and [sin\(\)](#).

Referenced by [tan\(\)](#).

5.2.2.127 tanh()

```
template<template< typename, const index\_t, const index\_t, typename > class Multivector,
typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::tanh (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const Multivector<Scalar_←
_T,LO,HI,Tune_P> [inline]
```

Hyperbolic tangent of multivector.

Definition at line 1016 of file [clifford_algebra_imp.h](#).

References [cosh\(\)](#), and [sinh\(\)](#).

5.2.2.128 to_demote()

```
template<typename Scalar_T >
auto glucat::to_demote (
    const Scalar_T & val) -> typename numeric\_traits<Scalar_T>::demoted::type [inline]
```

Cast to demote.

Definition at line 135 of file [scalar_imp.h](#).

References [glucat::numeric_traits< Scalar_T >::to_scalar_t\(\)](#).

5.2.2.129 to_promote()

```
template<typename Scalar_T >
auto glucat::to_promote (
    const Scalar_T & val) -> typename numeric\_traits<Scalar_T>::promoted::type [inline]
```

Cast to promote.

Definition at line 125 of file [scalar_imp.h](#).

References [glucat::numeric_traits< Scalar_T >::to_scalar_t\(\)](#).

5.2.2.130 try_catch() [1/2]

```
int glucat::try_catch (
    intfn f)
```

Exception catching for functions returning int.

Definition at line 49 of file [try_catch.h](#).

Referenced by [glucat::control_t::call\(\)](#), and [glucat::control_t::call\(\)](#).

5.2.2.131 try_catch() [2/2]

```
int glucat::try_catch (
    intintfn f,
    int arg)
```

Exception catching for functions of int returning int.

Definition at line 64 of file [try_catch.h](#).

5.2.2.132 vector_part()

```
template<template< typename, const index_t, const index_t, typename > class Multivector,
typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::vector_part (
    const Multivector< Scalar_T, LO, HI, Tune_P > & val) -> const std::vector<Scalar_←
_T> [inline]
```

Vector part of multivector, as a vector_t with respect to frame()

Definition at line 529 of file [clifford_algebra_imp.h](#).

5.2.3 Variable Documentation**5.2.3.1 BITS_PER_SET_VALUE**

```
const index_t glucat::BITS_PER_SET_VALUE = std::numeric_limits<set_value_t>::digits
```

Number of bits in set_value_t.

Definition at line 103 of file [global.h](#).

5.2.3.2 DEFAULT_HI

```
const index_t glucat::DEFAULT_HI = index_t(BITS_PER_SET_VALUE / 2)
```

Default highest index in an index set.

Definition at line 111 of file [global.h](#).

5.2.3.3 l_ln2

```
const long double glucat::l_ln2 = 0.6931471805599453094172321214581766L [static]
```

Definition at line 44 of file [long_double.h](#).

Referenced by [glucat::numeric_traits< Scalar_T >::ln_2\(\)](#).

5.2.3.4 l_pi

```
const long double glucat::l_pi = 3.1415926535897932384626433832795029L [static]
```

Definition at line 43 of file [long_double.h](#).

Referenced by [glucat::numeric_traits< Scalar_T >::pi\(\)](#).

5.2.3.5 MS_PER_S

```
const double glucat::MS_PER_S = 1000.0
```

Timing constant: deprecated here - moved to [test/timing.h](#).

Definition at line 83 of file [global.h](#).

5.2.3.6 Tuning_Fast_Basis_Max_Count

```
const unsigned int glucat::Tuning_Fast_Basis_Max_Count = 1
```

Definition at line 92 of file [tuning.h](#).

5.2.3.7 Tuning_Fast_CR_Sqrt_Max_Steps

```
const unsigned int glucat::Tuning_Fast_CR_Sqrt_Max_Steps = 256
```

Definition at line 88 of file [tuning.h](#).

5.2.3.8 Tuning_Fast_DB_Sqrt_Max_Steps

```
const unsigned int glucat::Tuning_Fast_DB_Sqrt_Max_Steps = 256
```

Definition at line 89 of file [tuning.h](#).

5.2.3.9 Tuning_Fast_Div_Max_Steps

```
const unsigned int glucat::Tuning_Fast_Div_Max_Steps = 0
```

Definition at line 87 of file [tuning.h](#).

5.2.3.10 Tuning_Fast_Fast_Size_Threshold

```
const unsigned int glucat::Tuning_Fast_Fast_Size_Threshold = 0
```

Definition at line 93 of file [tuning.h](#).

5.2.3.11 Tuning_Fast_Inv_Fast_Dim_Threshold

```
const unsigned int glucat::Tuning_Fast_Inv_Fast_Dim_Threshold = 0
```

Definition at line 94 of file [tuning.h](#).

5.2.3.12 Tuning_Fast_Log_Max_Inner_Steps

```
const unsigned int glucat::Tuning_Fast_Log_Max_Inner_Steps = 8
```

Definition at line 91 of file [tuning.h](#).

5.2.3.13 Tuning_Fast_Log_Max_Outer_Steps

```
const unsigned int glucat::Tuning_Fast_Log_Max_Outer_Steps = 16
```

Definition at line 90 of file [tuning.h](#).

5.2.3.14 Tuning_Fast_Mult_Matrix_Threshold

```
const unsigned int glucat::Tuning_Fast_Mult_Matrix_Threshold = 0
```

Definition at line 86 of file [tuning.h](#).

5.2.3.15 Tuning_Fast_Products_Size_Threshold

```
const unsigned int glucat::Tuning_Fast_Products_Size_Threshold = 0
```

Definition at line 95 of file [tuning.h](#).

5.2.3.16 Tuning_Int_Digits

```
const unsigned int glucat::Tuning_Int_Digits = std::numeric_limits<int>::digits
```

Definition at line 36 of file [tuning.h](#).

5.2.3.17 Tuning_Max_Threshold

```
const unsigned int glucat::Tuning_Max_Threshold = 1 << Tuning_Int_Digits
```

Definition at line 37 of file [tuning.h](#).

5.2.3.18 Tuning_Naive_Basis_Max_Count

```
const unsigned int glucat::Tuning_Naive_Basis_Max_Count = Tuning_Max_Threshold
```

Definition at line 65 of file [tuning.h](#).

5.2.3.19 Tuning_Naive_Fast_Size_Threshold

```
const unsigned int glucat::Tuning_Naive_Fast_Size_Threshold = Tuning_Max_Threshold
```

Definition at line 66 of file [tuning.h](#).

5.2.3.20 Tuning_Naive_Inv_Fast_Dim_Threshold

```
const unsigned int glucat::Tuning_Naive_Inv_Fast_Dim_Threshold = Tuning_Max_Threshold
```

Definition at line 67 of file [tuning.h](#).

5.2.3.21 Tuning_Naive_Mult_Matrix_Threshold

```
const unsigned int glucat::Tuning_Naive_Mult_Matrix_Threshold = 0
```

Definition at line 64 of file [tuning.h](#).

5.2.3.22 Tuning_Slow_Basis_Max_Count

```
const unsigned int glucat::Tuning_Slow_Basis_Max_Count = 0
```

Definition at line 42 of file [tuning.h](#).

5.2.3.23 Tuning_Slow_Fast_Size_Threshold

```
const unsigned int glucat::Tuning_Slow_Fast_Size_Threshold = Tuning_Max_Threshold
```

Definition at line 43 of file [tuning.h](#).

5.2.3.24 Tuning_Slow_Inv_Fast_Dim_Threshold

```
const unsigned int glucat::Tuning_Slow_Inv_Fast_Dim_Threshold = Tuning_Max_Threshold
```

Definition at line 44 of file [tuning.h](#).

5.2.3.25 Tuning_Slow_Mult_Matrix_Threshold

```
const unsigned int glucat::Tuning_Slow_Mult_Matrix_Threshold = Tuning_Max_Threshold
```

Definition at line 41 of file [tuning.h](#).

5.2.3.26 Tuning_Slow_Products_Size_Threshold

```
const unsigned int glucat::Tuning_Slow_Products_Size_Threshold = Tuning_Max_Threshold
```

Definition at line 45 of file [tuning.h](#).

5.3 glucat::gen Namespace Reference

Classes

- class [generator_table](#)
Table of generators for specific signatures.

Typedefs

- using [signature_t](#) = std::pair<[index_t](#), [index_t](#)>
A signature is a pair of indices, p, q, with p == frame.max(), q == -frame.min()

Variables

- static const std::array< [index_t](#), 8 > [offset_to_super](#) = {0,-1, 0,-1,-2, 3, 2, 1}
Offsets between the current signature and that of the real superalgebra.

5.3.1 Typedef Documentation

5.3.1.1 signature_t

```
using glucat::gen::signature_t = std::pair<index_t, index_t>
```

A signature is a pair of indices, p, q, with p == frame.max(), q == -frame.min()

Definition at line 48 of file [generation.h](#).

5.3.2 Variable Documentation

5.3.2.1 offset_to_super

```
const std::array<index_t, 8> glucat::gen::offset_to_super = {0,-1, 0,-1,-2, 3, 2, 1} [static]
```

Offsets between the current signature and that of the real superalgebra.

Definition at line 86 of file [generation.h](#).

Referenced by [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi\(\)](#), [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi\(\)](#), and [glucat::gen::generator_table< Matrix_T >::operator\(\)\(\)](#).

5.4 glucat::matrix Namespace Reference

Classes

- struct [eig_genus](#)
Structure containing classification of eigenvalues.

Typedefs

- using [eig_case_t](#)
Classification of eigenvalues of a matrix.

Functions

- template<typename LHS_T , typename RHS_T >
auto [kron](#) (const LHS_T &lhs, const RHS_T &rhs) -> const RHS_T
Kronecker tensor product of matrices - as per Matlab kron.
- template<typename LHS_T , typename RHS_T >
auto [mono_kron](#) (const LHS_T &lhs, const RHS_T &rhs) -> const RHS_T
Sparse Kronecker tensor product of monomial matrices.
- template<typename LHS_T , typename RHS_T >
auto [nork](#) (const LHS_T &lhs, const RHS_T &rhs, const bool mono=true) -> const RHS_T
Left inverse of Kronecker product.
- template<typename LHS_T , typename RHS_T >
auto [signed_perm_nork](#) (const LHS_T &lhs, const RHS_T &rhs) -> const RHS_T
Left inverse of Kronecker product where lhs is a signed permutation matrix.
- template<typename Matrix_T >
auto [nnz](#) (const Matrix_T &m) -> typename Matrix_T::size_type
Number of non-zeros.
- template<typename Matrix_T >
auto [isinf](#) (const Matrix_T &m) -> bool
Infinite.
- template<typename Matrix_T >
auto [isnan](#) (const Matrix_T &m) -> bool
Not a Number.
- template<typename Matrix_T >
auto [unit](#) (const typename Matrix_T::size_type n) -> const Matrix_T
Unit matrix - as per Matlab eye.
- template<typename LHS_T , typename RHS_T >
auto [mono_prod](#) (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_expression< RHS_T > &rhs) -> const typename RHS_T::expression_type
Product of monomial matrices.
- template<typename LHS_T , typename RHS_T >
auto [sparse_prod](#) (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_expression< RHS_T > &rhs) -> const typename RHS_T::expression_type
Product of sparse matrices.
- template<typename LHS_T , typename RHS_T >
auto [prod](#) (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_expression< RHS_T > &rhs) -> const typename RHS_T::expression_type
Product of matrices.

- `template<typename Scalar_T , typename LHS_T , typename RHS_T >`
`auto inner (const LHS_T &lhs, const RHS_T &rhs) -> Scalar_T`
*Inner product: $\sum(x(i,j)*y(i,j))/x.nrows()$*
- `template<typename Matrix_T >`
`auto norm_frob2 (const Matrix_T &val) -> typename Matrix_T::value_type`
Square of Frobenius norm.
- `template<typename Matrix_T >`
`auto trace (const Matrix_T &val) -> typename Matrix_T::value_type`
Matrix trace.
- `template<typename Matrix_T >`
`auto eigenvalues (const Matrix_T &val) -> std::vector< std::complex< double > >`
Eigenvalues of a matrix.
- `template<typename Matrix_T >`
`auto classify_eigenvalues (const Matrix_T &val) -> eig_genus< Matrix_T >`
Classify the eigenvalues of a matrix.
- `template<typename LHS_T , typename RHS_T >`
`void nork_range (RHS_T &result, const typename LHS_T::const_iterator2 lhs_it2, const RHS_T &rhs, const`
`typename RHS_T::size_type res_s1, const typename RHS_T::size_type res_s2)`
Utility routine for nork: calculate result for a range of indices.
- `template<typename Matrix_T >`
`static auto to_lapack (const Matrix_T &val) -> ublas::matrix< double, ublas::column_major >`
Convert matrix to LAPACK format.

5.4.1 Typedef Documentation

5.4.1.1 eig_case_t

using `glucat::matrix::eig_case_t`

Initial value:

```
enum {
    safe_eigs,
    neg_real_eigs,
    both_eigs}
```

Classification of eigenvalues of a matrix.

Definition at line 133 of file `matrix.h`.

5.4.2 Function Documentation

5.4.2.1 classify_eigenvalues()

```
template<typename Matrix_T >
auto glucat::matrix::classify_eigenvalues (
    const Matrix_T & val) -> eig_genus<Matrix_T>
```

Classify the eigenvalues of a matrix.

Definition at line 548 of file `matrix_imp.h`.

References `eigenvalues()`, `epsilon`, `glucat::matrix::eig_genus< Matrix_T >::m_eig_case`, `glucat::matrix::eig_genus< Matrix_T >::m_neg_real_eigs`, `glucat::matrix::eig_genus< Matrix_T >::m_safe_arg`, and `glucat::numeric_traits< Scalar_T >::pi()`.

Referenced by `glucat::matrix_log()`, and `glucat::matrix_sqrt()`.

5.4.2.2 eigenvalues()

```
template<typename Matrix_T >
auto glucat::matrix::eigenvalues (
    const Matrix_T & val) -> std::vector< std::complex<double> >
```

Eigenvalues of a matrix.

Definition at line 500 of file [matrix_imp.h](#).

References [to_lapack\(\)](#).

Referenced by [classify_eigenvalues\(\)](#).

5.4.2.3 inner()

```
template<typename Scalar_T , typename LHS_T , typename RHS_T >
auto glucat::matrix::inner (
    const LHS_T & lhs,
    const RHS_T & rhs) -> Scalar_T
```

Inner product: $\text{sum}(x(i,j)*y(i,j))/x.\text{nrows}()$

Inner product: $\text{sum}(\text{lhs}(i,j)*\text{rhs}(i,j))/\text{lhs}.\text{nrows}()$

Definition at line 373 of file [matrix_imp.h](#).

Referenced by [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi\(\)](#).

5.4.2.4 isinf()

```
template<typename Matrix_T >
auto glucat::matrix::isinf (
    const Matrix_T & m) -> bool
```

Infinite.

Definition at line 275 of file [matrix_imp.h](#).

5.4.2.5 isnan()

```
template<typename Matrix_T >
auto glucat::matrix::isnan (
    const Matrix_T & m) -> bool
```

Not a Number.

Definition at line 292 of file [matrix_imp.h](#).

Referenced by [glucat::operator/\(\)](#).

5.4.2.6 kron()

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::kron (
    const LHS_T & lhs,
    const RHS_T & rhs) -> const RHS_T
```

Kronecker tensor product of matrices - as per Matlab kron.

Definition at line 83 of file [matrix_imp.h](#).

Referenced by [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast\(\)](#).

5.4.2.7 mono_kron()

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::mono_kron (
    const LHS_T & lhs,
    const RHS_T & rhs) -> const RHS_T
```

Sparse Kronecker tensor product of monomial matrices.

Definition at line 119 of file [matrix_imp.h](#).

Referenced by [glucat::gen::generator_table< Matrix_T >::gen_from_pm1_qm1\(\)](#).

5.4.2.8 mono_prod()

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::mono_prod (
    const ublas::matrix_expression< LHS_T > & lhs,
    const ublas::matrix_expression< RHS_T > & rhs) -> const typename RHS_T::expression↵
_type
```

Product of monomial matrices.

Definition at line 320 of file [matrix_imp.h](#).

Referenced by [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_element\(\)](#), [glucat::gen::generator_table< Matrix_T >::gen_f](#), [glucat::gen::generator_table< Matrix_T >::gen_from_pp4_qm4\(\)](#), and [glucat::gen::generator_table< Matrix_T >::gen_from_qp1_pm](#).

5.4.2.9 nnz()

```
template<typename Matrix_T >
auto glucat::matrix::nnz (
    const Matrix_T & m) -> typename Matrix_T::size_type
```

Number of non-zeros.

Definition at line 258 of file [matrix_imp.h](#).

Referenced by [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi\(\)](#).

5.4.2.10 `nork()`

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::nork (
    const LHS_T & lhs,
    const RHS_T & rhs,
    const bool mono = true) -> const RHS_T
```

Left inverse of Kronecker product.

Definition at line 182 of file [matrix_imp.h](#).

References [nork_range\(\)](#), and [norm_frob2\(\)](#).

5.4.2.11 `nork_range()`

```
template<typename LHS_T , typename RHS_T >
void glucat::matrix::nork_range (
    RHS_T & result,
    const typename LHS_T::const_iterator2 lhs_it2,
    const RHS_T & rhs,
    const typename RHS_T::size_type res_s1,
    const typename RHS_T::size_type res_s2)
```

Utility routine for nork: calculate result for a range of indices.

Definition at line 152 of file [matrix_imp.h](#).

References [glucat::numeric_traits< Scalar_T >::to_scalar_t\(\)](#).

Referenced by [nork\(\)](#), and [signed_perm_nork\(\)](#).

5.4.2.12 `norm_frob2()`

```
template<typename Matrix_T >
auto glucat::matrix::norm_frob2 (
    const Matrix_T & val) -> typename Matrix_T::value_type
```

Square of Frobenius norm.

Definition at line 395 of file [matrix_imp.h](#).

References [glucat::numeric_traits< Scalar_T >::NaN\(\)](#).

Referenced by [nork\(\)](#).

5.4.2.13 `prod()`

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::prod (
    const ublas::matrix_expression< LHS_T > & lhs,
    const ublas::matrix_expression< RHS_T > & rhs) -> const typename RHS_T::expression↔
_type [inline]
```

Product of matrices.

Definition at line 361 of file [matrix_imp.h](#).

5.4.2.14 signed_perm_nork()

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::signed_perm_nork (
    const LHS_T & lhs,
    const RHS_T & rhs) -> const RHS_T
```

Left inverse of Kronecker product where lhs is a signed permutation matrix.

Definition at line 228 of file [matrix_imp.h](#).

References [nork_range\(\)](#).

Referenced by [glucat::fast\(\)](#).

5.4.2.15 sparse_prod()

```
template<typename LHS_T , typename RHS_T >
auto glucat::matrix::sparse_prod (
    const ublas::matrix_expression< LHS_T > & lhs,
    const ublas::matrix_expression< RHS_T > & rhs) -> const typename RHS_T::expression↵
_type [inline]
```

Product of sparse matrices.

Definition at line 350 of file [matrix_imp.h](#).

5.4.2.16 to_lapack()

```
template<typename Matrix_T >
static auto glucat::matrix::to_lapack (
    const Matrix_T & val) -> ublas::matrix<double, ublas::column_major> [static]
```

Convert matrix to LAPACK format.

Definition at line 440 of file [matrix_imp.h](#).

Referenced by [eigenvalues\(\)](#).

5.4.2.17 trace()

```
template<typename Matrix_T >
auto glucat::matrix::trace (
    const Matrix_T & val) -> typename Matrix_T::value_type
```

Matrix trace.

Definition at line 416 of file [matrix_imp.h](#).

References [glucat::numeric_traits< Scalar_T >::NaN\(\)](#).

5.4.2.18 unit()

```
template<typename Matrix_T >
auto glucat::matrix::unit (
    const typename Matrix_T::size_type n) -> const Matrix_T [inline]
```

Unit matrix - as per Matlab eye.

Definition at line 310 of file [matrix_imp.h](#).

Referenced by [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_element\(\)](#), [glucat::fast\(\)](#), [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::basis_element\(\)](#), [glucat::gen::generator_table< Matrix_T >::gen_from_pm1_qm1\(\)](#), and [glucat::gen::generator_table< Matrix_T >::gen_vector\(\)](#).

5.5 glucat::timing Namespace Reference

Functions

- static double [elapsed](#) (clock_t cpu_time)
Elapsed time in milliseconds.

Variables

- const double [MS_PER_SEC](#) = 1000.0
Timing constant: milliseconds per second.
- const double [MS_PER_CLOCK](#) = [MS_PER_SEC](#) / double(CLOCKS_PER_SEC)
Timing constant: milliseconds per clock.
- const int [EXTRA_TRIALS](#) = 2
Timing constant: trial expansion factor.

5.5.1 Function Documentation

5.5.1.1 elapsed()

```
static double glucat::timing::elapsed (
    clock_t cpu_time) [inline], [static]
```

Elapsed time in milliseconds.

Definition at line 51 of file [timing.h](#).

References [MS_PER_CLOCK](#).

5.5.2 Variable Documentation

5.5.2.1 EXTRA_TRIALS

```
const int glucat::timing::EXTRA_TRIALS = 2
```

Timing constant: trial expansion factor.

Definition at line 45 of file [timing.h](#).

5.5.2.2 MS_PER_CLOCK

```
const double glucat::timing::MS_PER_CLOCK = MS_PER_SEC / double(CLOCKS_PER_SEC)
```

Timing constant: milliseconds per clock.

Definition at line 42 of file [timing.h](#).

Referenced by [elapsed\(\)](#).

5.5.2.3 MS_PER_SEC

```
const double glucat::timing::MS_PER_SEC = 1000.0
```

Timing constant: milliseconds per second.

Definition at line 39 of file [timing.h](#).

5.6 pade Namespace Reference

Classes

- struct [pade_log_denom](#)

Coefficients of denominator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)

- struct [pade_log_denom](#)< [dd_real](#) >
- struct [pade_log_denom](#)< [float](#) >
- struct [pade_log_denom](#)< [long double](#) >
- struct [pade_log_denom](#)< [qd_real](#) >
- struct [pade_log_numer](#)

Coefficients of numerator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)

- struct [pade_log_numer](#)< [dd_real](#) >
- struct [pade_log_numer](#)< [float](#) >
- struct [pade_log_numer](#)< [long double](#) >
- struct [pade_log_numer](#)< [qd_real](#) >
- struct [pade_sqrt_denom](#)

Coefficients of denominator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)

- struct [pade_sqrt_denom](#)< [dd_real](#) >
- struct [pade_sqrt_denom](#)< [float](#) >
- struct [pade_sqrt_denom](#)< [long double](#) >
- struct [pade_sqrt_denom](#)< [qd_real](#) >
- struct [pade_sqrt_numer](#)

Coefficients of numerator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)

- struct [pade_sqrt_numer](#)< [dd_real](#) >
- struct [pade_sqrt_numer](#)< [float](#) >
- struct [pade_sqrt_numer](#)< [long double](#) >
- struct [pade_sqrt_numer](#)< [qd_real](#) >

5.7 PyClical Namespace Reference

Classes

- class [clifford](#)
- class [index_set](#)

Functions

- [index_set_hidden_doctests](#) ()
- [clifford_hidden_doctests](#) ()
- [e](#) (obj)
- [istpq](#) (p, q)
- [_test](#) ()

Variables

- [__version__](#) = str([glucat_package_version](#), 'utf-8')
- [lhs](#)
- [rhs](#)
- [threshold](#) = error_squared_tol([rhs](#)) if threshold is [None](#) else threshold
- [None](#)
- [tol](#) = error_squared_tol([rhs](#)) if tol is [None](#) else tol
- [obj](#)
- [i](#)
- [ixt](#)
- [fill](#)
- [scalar_epsilon](#) = [epsilon](#)
- float [pi](#) = atan([clifford](#)(1.0)) * 4.0
- float [tau](#) = atan([clifford](#)(1.0)) * 8.0
- [cl](#) = [clifford](#)
- [ist](#) = [index_set](#)
- [ninf3](#) = [e](#)(4) + [e](#)(-1)
- [nbar3](#) = [e](#)(4) - [e](#)(-1)

5.7.1 Function Documentation

5.7.1.1 [_test\(\)](#)

`PyClical._test ()` [protected]

Definition at line [1962](#) of file [PyClical.pyx](#).

References [_test\(\)](#).

Referenced by [_test\(\)](#).

5.7.1.2 clifford_hidden_doctests()

```
PyClical.clifford_hidden_doctests ()
```

Tests for functions that Doctest cannot see.

For clifford.__cinit__: Construct an object of type clifford.

```
>>> print(clifford(2))
2
>>> print(clifford(2.0))
2
>>> print(clifford(1.0e-1))
0.1
>>> print(clifford("2"))
2
>>> print(clifford("2{1,2,3}"))
2{1,2,3}
>>> print(clifford(clifford("2{1,2,3}")))
2{1,2,3}
>>> print(clifford("-{1}"))
-{1}
>>> print(clifford(2, index_set({1,2})))
2{1,2}
>>> print(clifford([2,3], index_set({1,2})))
2{1}+3{2}
>>> print(clifford([1,2]))
Traceback (most recent call last):
...
TypeError: Cannot initialize clifford object from <class 'list'>.
>>> print(clifford(None))
Traceback (most recent call last):
...
TypeError: Cannot initialize clifford object from <class 'NoneType'>.
>>> print(clifford(None, [1,2]))
Traceback (most recent call last):
...
TypeError: Cannot initialize clifford object from (<class 'NoneType'>, <class 'list'>).
>>> print(clifford([1,2], [1,2]))
Traceback (most recent call last):
...
TypeError: Cannot initialize clifford object from (<class 'list'>, <class 'list'>).
>>> print(clifford(""))
Traceback (most recent call last):
...
ValueError: Cannot initialize clifford object from invalid string ''.
>>> print(clifford("{")
Traceback (most recent call last):
...
ValueError: Cannot initialize clifford object from invalid string '{'.
>>> print(clifford("{1")
Traceback (most recent call last):
...
ValueError: Cannot initialize clifford object from invalid string '{1'.
>>> print(clifford("{+")
Traceback (most recent call last):
...
ValueError: Cannot initialize clifford object from invalid string '{+'.
>>> print(clifford("{-")
Traceback (most recent call last):
...
ValueError: Cannot initialize clifford object from invalid string '{-'.
>>> print(clifford("{1}+")
Traceback (most recent call last):
...
ValueError: Cannot initialize clifford object from invalid string '{1}+'.

For clifford.__richcmp__: Compare objects of type clifford.

>>> clifford("{1}") == clifford("1{1}")
True
```

```
>>> clifford("{1}") != clifford("1.0{1}")
False
>>> clifford("{1}") != clifford("1.0")
True
>>> clifford("{1,2}") == None
False
>>> clifford("{1,2}") != None
True
>>> None == clifford("{1,2}")
False
>>> None != clifford("{1,2}")
True
```

Definition at line 1253 of file [PyClicl.pyx](#).

5.7.1.3 e()

```
PyClicl.e (
    obj)
```

Abbreviation for `clifford(index_set(obj))`.

```
>>> print(e(1))
{1}
>>> print(e(-1))
{-1}
>>> print(e(0))
1
```

Definition at line 1936 of file [PyClicl.pyx](#).

5.7.1.4 index_set_hidden_doctests()

```
PyClicl.index_set_hidden_doctests ()
```

Tests for functions that Doctest cannot see.

For `index_set.__cinit__`: Construct `index_set`.

```
>>> print(index_set(1))
{1}
>>> print(index_set({1,2}))
{1,2}
>>> print(index_set(index_set({1,2})))
{1,2}
>>> print(index_set({1,2}))
{1,2}
>>> print(index_set({1,2,1}))
{1,2}
>>> print(index_set({1,2,1}))
{1,2}
>>> print(index_set(""))
{}
>>> print(index_set("{1}")
Traceback (most recent call last):
...
ValueError: Cannot initialize index_set object from invalid string '{1'.
>>> print(index_set("{1}")
Traceback (most recent call last):
...
ValueError: Cannot initialize index_set object from invalid string '{1'.
```

```
>>> print(index_set("{1,2,100}"))
Traceback (most recent call last):
...
ValueError: Cannot initialize index_set object from invalid string '{1,2,100}'.
>>> print(index_set({1,2,100}))
Traceback (most recent call last):
...
IndexError: Cannot initialize index_set object from invalid {1, 2, 100}.
>>> print(index_set([1,2]))
Traceback (most recent call last):
...
TypeError: Cannot initialize index_set object from <class 'list'>.

For index_set.__richcmp__: Compare two objects of class index_set.
```

```
>>> index_set(1) == index_set({1})
True
>>> index_set({1}) != index_set({1})
False
>>> index_set({1}) != index_set({2})
True
>>> index_set({1}) == index_set({2})
False
>>> index_set({1}) < index_set({2})
True
>>> index_set({1}) <= index_set({2})
True
>>> index_set({1}) > index_set({2})
False
>>> index_set({1}) >= index_set({2})
False
>>> None == index_set({1,2})
False
>>> None != index_set({1,2})
True
>>> None < index_set({1,2})
False
>>> None <= index_set({1,2})
False
>>> None > index_set({1,2})
False
>>> None >= index_set({1,2})
False
>>> index_set({1,2}) == None
False
>>> index_set({1,2}) != None
True
>>> index_set({1,2}) < None
False
>>> index_set({1,2}) <= None
False
>>> index_set({1,2}) > None
False
>>> index_set({1,2}) >= None
False
```

Definition at line 406 of file [PyClical.pyx](#).

5.7.1.5 istpq()

```
PyClical.istpq (
    p,
    q)
```

Abbreviation for `index_set({-q,...p})`.

```
>>> print(istpq(2,3))
{-3,-2,-1,1,2}
```

Definition at line 1949 of file [PyClical.pyx](#).

5.7.2 Variable Documentation

5.7.2.1 `__version__`

```
PyClical.__version__ = str(glucat_package_version, 'utf-8') [private]
```

Definition at line 35 of file [PyClical.pyx](#).

5.7.2.2 `cl`

```
PyClical.cl = clifford
```

Definition at line 1910 of file [PyClical.pyx](#).

5.7.2.3 `fill`

```
PyClical.fill
```

Definition at line 1864 of file [PyClical.pyx](#).

5.7.2.4 `i`

```
PyClical.i
```

Definition at line 1591 of file [PyClical.pyx](#).

5.7.2.5 `ist`

```
PyClical.ist = index_set
```

Definition at line 1928 of file [PyClical.pyx](#).

5.7.2.6 `ixt`

```
PyClical.ixt
```

Definition at line 1864 of file [PyClical.pyx](#).

5.7.2.7 `lhs`

```
PyClical.lhs
```

Definition at line 1359 of file [PyClical.pyx](#).

5.7.2.8 nbar3

```
PyClical.nbar3 = e(4) - e(-1)
```

Definition at line 1959 of file [PyClical.pyx](#).

5.7.2.9 ninf3

```
PyClical.ninf3 = e(4) + e(-1)
```

Definition at line 1958 of file [PyClical.pyx](#).

5.7.2.10 None

```
PyClical.None
```

Definition at line 1359 of file [PyClical.pyx](#).

5.7.2.11 obj

```
PyClical.obj
```

Definition at line 1591 of file [PyClical.pyx](#).

5.7.2.12 pi

```
float PyClical.pi = atan(clifford(1.0)) * 4.0
```

Definition at line 1907 of file [PyClical.pyx](#).

5.7.2.13 rhs

```
PyClical.rhs
```

Definition at line 1359 of file [PyClical.pyx](#).

5.7.2.14 scalar_epsilon

```
PyClical.scalar_epsilon = epsilon
```

Definition at line 1905 of file [PyClical.pyx](#).

5.7.2.15 tau

```
float PyClical.tau = atan(clifford(1.0)) * 8.0
```

Definition at line 1908 of file [PyClical.pyx](#).

5.7.2.16 threshold

```
PyClical.threshold = error_squared_tol(rhs) if threshold is None else threshold
```

Definition at line 1359 of file [PyClical.pyx](#).

5.7.2.17 tol

```
PyClical.tol = error_squared_tol(rhs) if tol is None else tol
```

Definition at line 1359 of file [PyClical.pyx](#).

5.8 std Namespace Reference

Classes

- struct [numeric_limits](#)< [glucat::framed_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) > >
Numeric limits for framed_multi inherit limits for the corresponding scalar type.
- struct [numeric_limits](#)< [glucat::matrix_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) > >
Numeric limits for matrix_multi inherit limits for the corresponding scalar type.

Chapter 6

Class Documentation

6.1 glucat::basis_table< Scalar_T, LO, HI, Matrix_T > Class Template Reference

Table of basis elements used as a cache by basis_element()

```
#include <matrix_multi_imp.h>
```

Inheritance diagram for glucat::basis_table< Scalar_T, LO, HI, Matrix_T >:



Collaboration diagram for glucat::basis_table< Scalar_T, LO, HI, Matrix_T >:



Public Member Functions

- [basis_table](#) (const [basis_table](#) &)=delete
- auto [operator=](#) (const [basis_table](#) &) -> [basis_table](#) &=delete

Static Public Member Functions

- static auto [basis](#) () -> [basis_table](#) &
Single instance of basis table.

Private Member Functions

- [basis_table](#) ()=default
- [~basis_table](#) ()=default

Friends

- class [friend_for_private_destructor](#)

6.1.1 Detailed Description

`template<typename Scalar_T, const index_t LO, const index_t HI, typename Matrix_T>`
`class glucat::basis_table< Scalar_T, LO, HI, Matrix_T >`

Table of basis elements used as a cache by [basis_element\(\)](#)

Definition at line [1162](#) of file [matrix_multi_imp.h](#).

6.1.2 Constructor & Destructor Documentation

6.1.2.1 [basis_table\(\)](#) [1/2]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Matrix_T >
glucat::basis_table< Scalar_T, LO, HI, Matrix_T >::basis_table () [private], [default]
```

6.1.2.2 [~basis_table\(\)](#)

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Matrix_T >
glucat::basis_table< Scalar_T, LO, HI, Matrix_T >::~~basis_table () [private], [default]
```

6.1.2.3 [basis_table\(\)](#) [2/2]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Matrix_T >
glucat::basis_table< Scalar_T, LO, HI, Matrix_T >::basis_table (
    const basis\_table< Scalar_T, LO, HI, Matrix_T > & ) [delete]
```

6.1.3 Member Function Documentation

6.1.3.1 basis()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Matrix_T >
static auto glucat::basis_table< Scalar_T, LO, HI, Matrix_T >::basis () -> basis_table&    [inline],
[static]
```

Single instance of basis table.

Definition at line 1168 of file [matrix_multi_imp.h](#).

6.1.3.2 operator=()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Matrix_T >
auto glucat::basis_table< Scalar_T, LO, HI, Matrix_T >::operator= (
    const basis_table< Scalar_T, LO, HI, Matrix_T > & ) -> basis_table &=delete
[delete]
```

6.1.4 Friends And Related Symbol Documentation

6.1.4.1 friend_for_private_destructor

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Matrix_T >
friend class friend_for_private_destructor    [friend]
```

Friend declaration to avoid compiler warning: "... only defines a private destructor and has no friends" Ref: Carlos O'Ryan, ACE <http://doc.ece.uci.edu>

Definition at line 1173 of file [matrix_multi_imp.h](#).

The documentation for this class was generated from the following file:

- [glucat/matrix_multi_imp.h](#)

6.2 glucat::bool_to_type< truth_value > Class Template Reference

Bool to type.

```
#include <global.h>
```

Private Types

- enum { [value](#) = truth_value }

6.2.1 Detailed Description

```
template<bool truth_value>
class glucat::bool_to_type< truth_value >
```

Bool to type.

Definition at line 69 of file [global.h](#).

6.2.2 Member Enumeration Documentation

6.2.2.1 anonymous enum

```
template<bool truth_value>
anonymous enum [private]
```

Enumerator

value	
-------	--

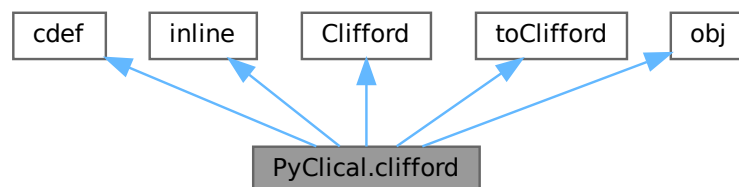
Definition at line 72 of file [global.h](#).

The documentation for this class was generated from the following file:

- [glucat/global.h](#)

6.3 PyClical.clifford Class Reference

Inheritance diagram for PyClical.clifford:



Collaboration diagram for PyClical.clifford:



Public Member Functions

- `__cinit__` (self, other=0, `ixt=None`)
- `__dealloc__` (self)
- `__contains__` (self, x)
- `__iter__` (self)
- `reframe` (self, `ixt`)
- `__richcmp__` (lhs, rhs, int, op)
- `__getitem__` (self, `ixt`)
- `__neg__` (self)
- `__pos__` (self)
- `__add__` (lhs, rhs)
- `__iadd__` (self, rhs)
- `__sub__` (lhs, rhs)
- `__isub__` (self, rhs)
- `__mul__` (lhs, rhs)
- `__imul__` (self, rhs)
- `__mod__` (lhs, rhs)
- `__imod__` (self, rhs)
- `__and__` (lhs, rhs)
- `__iand__` (self, rhs)
- `__xor__` (lhs, rhs)
- `__ixor__` (self, rhs)
- `__truediv__` (lhs, rhs)
- `__idiv__` (self, rhs)
- `inv` (self)
- `__or__` (lhs, rhs)
- `__ior__` (self, rhs)
- `__pow__` (self, m, dummy)
- `pow` (self, m)
- `outer_pow` (self, m)
- `__call__` (self, grade)
- `scalar` (self)
- `pure` (self)
- `even` (self)
- `odd` (self)
- `vector_part` (self, frm=`None`)
- `involute` (self)
- `reverse` (self)

- [conj](#) (self)
- [quad](#) (self)
- [norm](#) (self)
- [abs](#) (self)
- [max_abs](#) (self)
- [truncated](#) (self, limit)
- [isinf](#) (self)
- [isnan](#) (self)
- [frame](#) (self)
- [__repr__](#) (self)
- [__str__](#) (self)

Public Attributes

- [instance](#) = new [Clifford](#)((<[clifford](#)>other).unwrap())

6.3.1 Detailed Description

Python class `clifford` wraps C++ class `Clifford`.

Definition at line 537 of file [PyClical.pyx](#).

6.3.2 Member Function Documentation

6.3.2.1 `__add__()`

```
PyClical.clifford.__add__ (
    lhs,
    rhs)
```

Geometric sum.

```
>>> print(clifford(1) + clifford("{2}"))
1+{2}
>>> print(clifford("{1}") + clifford("{2}"))
{1}+{2}
```

Definition at line 740 of file [PyClical.pyx](#).

6.3.2.2 `__and__()`

```
PyClical.clifford.__and__ (
    lhs,
    rhs)
```

Inner product.

```
>>> print(clifford("{1}") & clifford("{2}"))
0
>>> print(clifford(2) & clifford("{2}"))
0
>>> print(clifford("{1}") & clifford("{1}"))
1
>>> print(clifford("{1}") & clifford("{1,2}"))
{2}
```

Definition at line 836 of file [PyClical.pyx](#).

6.3.2.3 `__call__()`

```
PyClical.clifford.__call__ (
    self,
    grade)
```

Pure grade-vector part.

```
>>> print(clifford("{1}") (1))
{1}
>>> print(clifford("{1}") (0))
0
>>> print(clifford("1+{1}+{1,2}") (0))
1
>>> print(clifford("1+{1}+{1,2}") (1))
{1}
>>> print(clifford("1+{1}+{1,2}") (2))
{1,2}
>>> print(clifford("1+{1}+{1,2}") (3))
0
```

Definition at line 1020 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

6.3.2.4 `__cinit__()`

```
PyClical.clifford.__cinit__ (
    self,
    other = 0,
    ixt = None)
```

Construct an object of type clifford.

```
>>> print(clifford(2))
2
>>> print(clifford(2.0))
2
>>> print(clifford(1.0e-1))
0.1
>>> print(clifford("2"))
2
>>> print(clifford("2{1,2,3}"))
2{1,2,3}
>>> print(clifford(clifford("2{1,2,3}")))
2{1,2,3}
>>> print(clifford("-{1}") )
-{1}
>>> print(clifford(2, index_set ({1,2})))
2{1,2}
>>> print(clifford([2,3], index_set ({1,2})))
2{1}+3{2}
```

Definition at line 565 of file [PyClical.pyx](#).

6.3.2.5 `__contains__()`

```
PyClical.clifford.__contains__ (
    self,
    x)
```

Not applicable.

```
>>> x=clifford(index_set({-3,4,7})); -3 in x
Traceback (most recent call last):
...
TypeError: Not applicable.
```

Definition at line 627 of file [PyClical.pyx](#).

6.3.2.6 `__dealloc__()`

```
PyClical.clifford.__dealloc__ (
    self)
```

Clean up by deallocating the instance of C++ class Clifford.

Definition at line 621 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

6.3.2.7 `__getitem__()`

```
PyClical.clifford.__getitem__ (
    self,
    ixt)
```

Subscripting: map from index set to scalar coordinate.

```
>>> clifford("{1}")[index_set(1)]
1.0
>>> clifford("{1}")[index_set({1})]
1.0
>>> clifford("{1}")[index_set({1,2})]
0.0
>>> clifford("2{1,2}")[index_set({1,2})]
2.0
```

Definition at line 707 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

6.3.2.8 __iadd__()

```
PyClical.clifford.__iadd__ (
    self,
    rhs)
```

Geometric sum.

```
>>> x = clifford(1); x += clifford("{2}"); print(x)
1+{2}
```

Definition at line 751 of file [PyClical.pyx](#).

6.3.2.9 __iand__()

```
PyClical.clifford.__iand__ (
    self,
    rhs)
```

Inner product.

```
>>> x = clifford("{1}"); x &= clifford("{2}"); print(x)
0
>>> x = clifford(2); x &= clifford("{2}"); print(x)
0
>>> x = clifford("{1}"); x &= clifford("{1}"); print(x)
1
>>> x = clifford("{1}"); x &= clifford("{1,2}"); print(x)
{2}
```

Definition at line 851 of file [PyClical.pyx](#).

6.3.2.10 __idiv__()

```
PyClical.clifford.__idiv__ (
    self,
    rhs)
```

Geometric quotient.

```
>>> x = clifford("{1}"); x /= clifford("{2}"); print(x)
{1,2}
>>> x = clifford(2); x /= clifford("{2}"); print(x)
2{2}
>>> x = clifford("{1}"); x /= clifford("{1}"); print(x)
1
>>> x = clifford("{1}"); x /= clifford("{1,2}"); print(x)
-{2}
```

Definition at line 911 of file [PyClical.pyx](#).

6.3.2.11 `__imod__()`

```
PyClical.clifford.__imod__ (
    self,
    rhs)
```

Contraction.

```
>>> x = clifford("{1}"); x %= clifford("{2}"); print(x)
0
>>> x = clifford(2); x %= clifford("{2}"); print(x)
2{2}
>>> x = clifford("{1}"); x %= clifford("{1}"); print(x)
1
>>> x = clifford("{1}"); x %= clifford("{1,2}"); print(x)
{2}
```

Definition at line 821 of file [PyClical.pyx](#).

6.3.2.12 `__imul__()`

```
PyClical.clifford.__imul__ (
    self,
    rhs)
```

Geometric product.

```
>>> x = clifford(2); x *= clifford("{2}"); print(x)
2{2}
>>> x = clifford("{1}"); x *= clifford("{2}"); print(x)
{1,2}
>>> x = clifford("{1}"); x *= clifford("{1,2}"); print(x)
{2}
```

Definition at line 793 of file [PyClical.pyx](#).

6.3.2.13 `__ior__()`

```
PyClical.clifford.__ior__ (
    self,
    rhs)
```

Transform left hand side, using right hand side as a transformation.

```
>>> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); y|=x; print(y)
-{1}
>>> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); y|=exp(x); print(y)
-{1}
```

Definition at line 950 of file [PyClical.pyx](#).

6.3.2.14 `__isub__()`

```
PyClical.clifford.__isub__ (
    self,
    rhs)
```

Geometric difference.

```
>>> x = clifford(1); x -= clifford("{2}"); print(x)
1-{2}
```

Definition at line 771 of file [PyClical.pyx](#).

6.3.2.15 `__iter__()`

```
PyClical.clifford.__iter__ (
    self)
```

Not applicable.

```
>>> for a in clifford(index_set({-3,4,7})):print(a, end=",")
Traceback (most recent call last):
...
TypeError: Not applicable.
```

Definition at line 638 of file [PyClical.pyx](#).

6.3.2.16 `__ixor__()`

```
PyClical.clifford.__ixor__ (
    self,
    rhs)
```

Outer product.

```
>>> x = clifford("{1}"); x ^= clifford("{2}"); print(x)
{1,2}
>>> x = clifford(2); x ^= clifford("{2}"); print(x)
2{2}
>>> x = clifford("{1}"); x ^= clifford("{1}"); print(x)
0
>>> x = clifford("{1}"); x ^= clifford("{1,2}"); print(x)
0
```

Definition at line 881 of file [PyClical.pyx](#).

6.3.2.17 `__mod__()`

```
PyClical.clifford.__mod__ (
    lhs,
    rhs)
```

Contraction.

```
>>> print(clifford("{1}") % clifford("{2}"))
0
>>> print(clifford(2) % clifford("{2}"))
2{2}
>>> print(clifford("{1}") % clifford("{1}"))
1
>>> print(clifford("{1}") % clifford("{1,2}"))
{2}
```

Definition at line 806 of file [PyClical.pyx](#).

6.3.2.18 `__mul__()`

```
PyClical.clifford.__mul__ (
    lhs,
    rhs)
```

Geometric product.

```
>>> print(clifford("{1}") * clifford("{2}"))
{1,2}
>>> print(clifford(2) * clifford("{2}"))
2{2}
>>> print(clifford("{1}") * clifford("{1,2}"))
{2}
```

Definition at line 780 of file [PyClical.pyx](#).

6.3.2.19 `__neg__()`

```
PyClical.clifford.__neg__ (
    self)
```

Unary `-`.

```
>>> print(-clifford("{1}"))
-{1}
```

Definition at line 722 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

6.3.2.20 `__or__()`

```
PyClical.clifford.__or__ (
    lhs,
    rhs)
```

Transform left hand side, using right hand side as a transformation.

```
>>> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); print(y|x)
-{1}
>>> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); print(y|exp(x))
-{1}
```

Definition at line 939 of file [PyClical.pyx](#).

6.3.2.21 `__pos__()`

```
PyClical.clifford.__pos__ (
    self)
```

Unary +.

```
>>> print(+clifford("{1}"))
{1}
```

Definition at line 731 of file [PyClical.pyx](#).

6.3.2.22 `__pow__()`

```
PyClical.clifford.__pow__ (
    self,
    m,
    dummy)
```

Power: self to the m.

```
>>> x=clifford("{1}"); print(x ** 2)
1
>>> x=clifford("2"); print(x ** 2)
4
>>> x=clifford("2+{1}"); print(x ** 0)
1
>>> x=clifford("2+{1}"); print(x ** 1)
2+{1}
>>> x=clifford("2+{1}"); print(x ** 2)
5+4{1}
>>> i=clifford("{1,2}"); print(exp(pi/2) * (i ** i))
1
```

Definition at line 961 of file [PyClical.pyx](#).

References [PyClical.clifford.pow\(\)](#).

6.3.2.23 `__repr__()`

```
PyClical.clifford.__repr__ (
    self)
```

The "official" string representation of self.

```
>>> clifford("1+3{-1}+2{1,2}+4{-2,7}").__repr__()
'clifford("1+3{-1}+2{1,2}+4{-2,7}")'
```

Definition at line 1235 of file [PyClical.pyx](#).

References [clifford_to_repr\(\)](#).

6.3.2.24 `__richcmp__()`

```
PyClical.clifford.__richcmp__ (
    lhs,
    rhs,
    int,
    op)
```

Compare objects of type clifford.

```
>>> clifford("{1}") == clifford("1{1}")
True
>>> clifford("{1}") != clifford("1.0{1}")
False
>>> clifford("{1}") != clifford("1.0")
True
>>> clifford("{1,2}") == None
False
>>> clifford("{1,2}") != None
True
>>> None == clifford("{1,2}")
False
>>> None != clifford("{1,2}")
True
```

Definition at line 672 of file [PyClical.pyx](#).

6.3.2.25 `__str__()`

```
PyClical.clifford.__str__ (
    self)
```

The "informal" string representation of self.

```
>>> clifford("1+3{-1}+2{1,2}+4{-2,7}").__str__()
'1+3{-1}+2{1,2}+4{-2,7}'
```

Definition at line 1244 of file [PyClical.pyx](#).

References [clifford_to_str\(\)](#).

6.3.2.26 `__sub__()`

```
PyClical.clifford.__sub__ (
    lhs,
    rhs)
```

Geometric difference.

```
>>> print(clifford(1) - clifford("{2}"))
1-{2}
>>> print(clifford("{1}") - clifford("{2}"))
{1}-{2}
```

Definition at line 760 of file [PyClical.pyx](#).

6.3.2.27 `__truediv__()`

```
PyClical.clifford.__truediv__ (
    lhs,
    rhs)
```

Geometric quotient.

```
>>> print(clifford("{1}") / clifford("{2}"))
{1,2}
>>> print(clifford(2) / clifford("{2}"))
2{2}
>>> print(clifford("{1}") / clifford("{1}"))
1
>>> print(clifford("{1}") / clifford("{1,2}"))
-{2}
```

Definition at line 896 of file [PyClical.pyx](#).

6.3.2.28 `__xor__()`

```
PyClical.clifford.__xor__ (
    lhs,
    rhs)
```

Outer product.

```
>>> print(clifford("{1}") ^ clifford("{2}"))
{1,2}
>>> print(clifford(2) ^ clifford("{2}"))
2{2}
>>> print(clifford("{1}") ^ clifford("{1}"))
0
>>> print(clifford("{1}") ^ clifford("{1,2}"))
0
```

Definition at line 866 of file [PyClical.pyx](#).

6.3.2.29 abs()

```
PyClical.clifford.abs (
    self)
```

Absolute value: square root of norm.

```
>>> clifford("1+{-1}+{1,2}+{1,2,3}").abs()
2.0
```

Definition at line 1175 of file [PyClical.pyx](#).

References [glucat.abs\(\)](#).

6.3.2.30 conj()

```
PyClical.clifford.conj (
    self)
```

Conjugation, reverse o involute == involute o reverse.

```
>>> print((clifford("{1}")).conj())
-{1}
>>> print((clifford("{2}") * clifford("{1}")).conj())
{1,2}
>>> print((clifford("{1}") * clifford("{2}")).conj())
-{1,2}
>>> print(clifford("1+{1}+{1,2}").conj())
1-{1}-{1,2}
```

Definition at line 1138 of file [PyClical.pyx](#).

References [PyClical.clifford.conj\(\)](#), [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

Referenced by [PyClical.clifford.conj\(\)](#).

6.3.2.31 even()

```
PyClical.clifford.even (
    self)
```

Even part of multivector, sum of even grade terms.

```
>>> print(clifford("1+{1}+{1,2}").even())
1+{1,2}
```

Definition at line 1061 of file [PyClical.pyx](#).

References [PyClical.clifford.even\(\)](#), [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

Referenced by [PyClical.clifford.even\(\)](#).

6.3.2.32 frame()

```
PyClical.clifford.frame (
    self)
```

Subalgebra generated by all generators of terms of given multivector.

```
>>> print(clifford("1+3{-1}+2{1,2}+4{-2,7}").frame())
{-2,-1,1,2,7}
>>> s=clifford("1+3{-1}+2{1,2}+4{-2,7}").frame(); type(s)
<class 'PyClical.index_set'>
```

Definition at line 1224 of file [PyClical.pyx](#).

References [PyClical.clifford.frame\(\)](#), [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

Referenced by [PyClical.clifford.frame\(\)](#).

6.3.2.33 inv()

```
PyClical.clifford.inv (
    self)
```

Geometric multiplicative inverse.

```
>>> x = clifford("{1}"); print(x.inv())
{1}
>>> x = clifford(2); print(x.inv())
0.5
>>> x = clifford("{1,2}"); print(x.inv())
-1,2}
```

Definition at line 926 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.inv\(\)](#).

Referenced by [PyClical.clifford.inv\(\)](#).

6.3.2.34 involute()

```
PyClical.clifford.involute (
    self)
```

Main involution, each {i} is replaced by -{i} in each term,
eg. `clifford("{1}") -> -clifford("{1}")`.

```
>>> print(clifford("{1}").involute())
-1}
>>> print((clifford("{2}") * clifford("{1}")).involute())
-1,2}
>>> print((clifford("{1}") * clifford("{2}")).involute())
1,2}
>>> print(clifford("1+{1}+{1,2}").involute())
1-1}+1,2}
```

Definition at line 1107 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.involute\(\)](#).

Referenced by [PyClical.clifford.involute\(\)](#).

6.3.2.35 isinf()

```
PyClical.clifford.isinf (  
    self)
```

Check if a multivector contains any infinite values.

```
>>> clifford().isinf()  
False
```

Definition at line 1206 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.isnan\(\)](#).

6.3.2.36 isnan()

```
PyClical.clifford.isnan (  
    self)
```

Check if a multivector contains any IEEE NaN values.

```
>>> clifford().isnan()  
False
```

Definition at line 1215 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.isnan\(\)](#).

Referenced by [PyClical.clifford.isinf\(\)](#), and [PyClical.clifford.isnan\(\)](#).

6.3.2.37 max_abs()

```
PyClical.clifford.max_abs (  
    self)
```

Maximum of absolute values of components of multivector: multivector infinity norm.

```
>>> clifford("1+{-1}+{1,2}+{1,2,3}").max_abs()  
1.0  
>>> clifford("3+2{1}+{1,2}").max_abs()  
3.0
```

Definition at line 1184 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.max_abs\(\)](#).

Referenced by [PyClical.clifford.max_abs\(\)](#).

6.3.2.38 norm()

```
PyClical.clifford.norm (  
    self)
```

Norm == sum of squares of coordinates.

```
>>> clifford("1+{1}+{1,2}").norm()  
3.0  
>>> clifford("1+{-1}+{1,2}+{1,2,3}").norm()  
4.0
```

Definition at line 1164 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.norm\(\)](#).

Referenced by [PyClical.clifford.norm\(\)](#).

6.3.2.39 odd()

```
PyClical.clifford.odd (  
    self)
```

Odd part of multivector, sum of odd grade terms.

```
>>> print(clifford("1+{1}+{1,2}").odd())  
{1}
```

Definition at line 1070 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.odd\(\)](#).

Referenced by [PyClical.clifford.odd\(\)](#).

6.3.2.40 outer_pow()

```
PyClical.clifford.outer_pow (  
    self,  
    m)
```

Outer product power.

```
>>> x=clifford("2+{1}"); print(x.outer_pow(0))  
1  
>>> x=clifford("2+{1}"); print(x.outer_pow(1))  
2+{1}  
>>> x=clifford("2+{1}"); print(x.outer_pow(2))  
4+4{1}  
>>> print(clifford("1+{1}+{1,2}").outer_pow(3))  
1+3{1}+3{1,2}
```

Definition at line 1004 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.outer_pow\(\)](#).

Referenced by [PyClical.clifford.outer_pow\(\)](#).

6.3.2.41 pow()

```

PyClical.clifford.pow (
    self,
    m)

Power: self to the m.

>>> x=clifford("{1}"); print(x.pow(2))
1
>>> x=clifford("2"); print(x.pow(2))
4
>>> x=clifford("2+{1}"); print(x.pow(0))
1
>>> x=clifford("2+{1}"); print(x.pow(1))
2+{1}
>>> x=clifford("2+{1}"); print(x.pow(2))
5+4{1}
>>> print(clifford("1+{1}+{1,2}").pow(3))
1+3{1}+3{1,2}
>>> i=clifford("{1,2}"); print(exp(pi/2) * i.pow(i))
1

```

Definition at line 980 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.pow\(\)](#).

Referenced by [PyClical.clifford.__pow__\(\)](#), and [PyClical.clifford.pow\(\)](#).

6.3.2.42 pure()

```

PyClical.clifford.pure (
    self)

Pure part.

>>> print(clifford("1+{1}+{1,2}").pure())
{1}+{1,2}
>>> print(clifford("{1,2}").pure())
{1,2}

```

Definition at line 1050 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.pure\(\)](#).

Referenced by [PyClical.clifford.pure\(\)](#).

6.3.2.43 quad()

```

PyClical.clifford.quad (
    self)

Quadratic form == (rev(x)*x)(0).

>>> print(clifford("1+{1}+{1,2}").quad())
3.0
>>> print(clifford("1+{-1}+{1,2}+{1,2,3}").quad())
2.0

```

Definition at line 1153 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.quad\(\)](#).

Referenced by [PyClical.clifford.quad\(\)](#).

6.3.2.44 reframe()

```
PyClical.clifford.reframe (
    self,
    ixt)
```

Put self into a larger frame, containing the union of self.frame() and index set ixt. This can be used to make multiplication faster, by multiplying within a common frame.

```
>>> clifford("2+3{1}").reframe(index_set({1,2,3}))
clifford("2+3{1}")
>>> s=index_set({1,2,3});t=index_set({-3,-2,-1});x=random_clifford(s); x.reframe(t).frame() == (s|t);
True
```

Definition at line 649 of file [PyClical.pyx](#).

6.3.2.45 reverse()

```
PyClical.clifford.reverse (
    self)
```

Reversion, eg. `clifford("{1}")*clifford("{2}") -> clifford("{2}")*clifford("{1}")`.

```
>>> print(clifford("{1}").reverse())
{1}
>>> print((clifford("{2}") * clifford("{1}")).reverse())
{1,2}
>>> print((clifford("{1}") * clifford("{2}")).reverse())
-{1,2}
>>> print(clifford("1+{1}+{1,2}").reverse())
1+{1}-{1,2}
```

Definition at line 1123 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.reverse\(\)](#).

Referenced by [PyClical.clifford.reverse\(\)](#).

6.3.2.46 scalar()

```
PyClical.clifford.scalar (
    self)
```

Scalar part.

```
>>> clifford("1+{1}+{1,2}").scalar()
1.0
>>> clifford("{1,2}").scalar()
0.0
```

Definition at line 1039 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.scalar\(\)](#).

Referenced by [PyClical.clifford.scalar\(\)](#).

6.3.2.47 truncated()

```
PyClical.clifford.truncated (
    self,
    limit)
```

Remove all terms of self with relative size smaller than limit.

```
>>> clifford("1e8+{1}+1e-8{1,2}").truncated(1.0e-6)
clifford("100000000")
>>> clifford("1e4+{1}+1e-4{1,2}").truncated(1.0e-6)
clifford("10000+{1}")
```

Definition at line 1195 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.truncated\(\)](#).

Referenced by [PyClical.clifford.truncated\(\)](#).

6.3.2.48 vector_part()

```
PyClical.clifford.vector_part (
    self,
    frm = None)
```

Vector part of multivector, as a Python list, with respect to frm.

```
>>> print(clifford("1+2{1}+3{2}+4{1,2}").vector_part())
[2.0, 3.0]
>>> print(clifford("1+2{1}+3{2}+4{1,2}").vector_part(index_set((-1,1,2))))
[0.0, 2.0, 3.0]
```

Definition at line 1079 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.clifford.vector_part\(\)](#).

Referenced by [PyClical.clifford.vector_part\(\)](#).

6.3.3 Member Data Documentation

6.3.3.1 instance

```
PyClical.clifford.instance = new Clifford((<clifford>other).unwrap())
```

Definition at line 592 of file [PyClical.pyx](#).

Referenced by [PyClical.clifford.__call__\(\)](#), [PyClical.index_set.__contains__\(\)](#), [PyClical.clifford.__dealloc__\(\)](#), [PyClical.index_set.__dealloc__\(\)](#), [PyClical.clifford.__getitem__\(\)](#), [PyClical.index_set.__getitem__\(\)](#), [PyClical.index_set.__invert__\(\)](#), [PyClical.clifford.__neg__\(\)](#), [PyClical.index_set.__setitem__\(\)](#), [PyClical.clifford.conj\(\)](#), [PyClical.index_set.count\(\)](#), [PyClical.index_set.count_neg\(\)](#), [PyClical.index_set.count_pos\(\)](#), [PyClical.clifford.even\(\)](#), [PyClical.clifford.frame\(\)](#), [PyClical.index_set.hash_fn\(\)](#), [PyClical.clifford.inv\(\)](#), [PyClical.clifford.involute\(\)](#), [PyClical.clifford.isinf\(\)](#), [PyClical.clifford.isnan\(\)](#), [PyClical.index_set.max\(\)](#), [PyClical.clifford.max_abs\(\)](#), [PyClical.index_set.min\(\)](#), [PyClical.clifford.norm\(\)](#), [PyClical.clifford.odd\(\)](#), [PyClical.clifford.outer_pow\(\)](#), [PyClical.clifford.pow\(\)](#), [PyClical.clifford.pure\(\)](#), [PyClical.clifford.quad\(\)](#), [PyClical.clifford.reverse\(\)](#), [PyClical.clifford.scalar\(\)](#), [PyClical.index_set.sign_of_mult\(\)](#), [PyClical.index_set.sign_of_square\(\)](#), [PyClical.clifford.truncated\(\)](#), and [PyClical.clifford.vector_part\(\)](#).

The documentation for this class was generated from the following file:

- [pyclical/PyClical.pyx](#)

6.4 glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T > Class Template Reference

clifford_algebra<> declares the operations of a Clifford algebra

```
#include <clifford_algebra.h>
```

Public Types

- using [scalar_t](#) = Scalar_T
- using [index_set_t](#) = Index_Set_T
- using [multivector_t](#) = Multivector_T
- using [pair_t](#) = std::pair<const [index_set_t](#), Scalar_T>
- using [vector_t](#) = std::vector<Scalar_T>

Public Member Functions

- virtual [~clifford_algebra](#) ()=default
- virtual auto [operator==](#) (const [multivector_t](#) &val) const -> bool=0
Test for equality of multivectors.
- virtual auto [operator==](#) (const Scalar_T &scr) const -> bool=0
Test for equality of multivector and scalar.
- virtual auto [operator+=](#) (const [multivector_t](#) &rhs) -> [multivector_t](#) &=0
Geometric sum.
- virtual auto [operator+=](#) (const Scalar_T &scr) -> [multivector_t](#) &=0
Geometric sum of multivector and scalar.
- virtual auto [operator-=](#) (const [multivector_t](#) &rhs) -> [multivector_t](#) &=0
Geometric difference.
- virtual auto [operator-=](#) (const Scalar_T &scr) -> [multivector_t](#) &=0
Geometric difference of multivector and scalar.
- virtual auto [operator-](#) () const -> const [multivector_t](#)=0
Unary -.
- virtual auto [operator*=](#) (const Scalar_T &scr) -> [multivector_t](#) &=0
Product of multivector and scalar.
- virtual auto [operator*=](#) (const [multivector_t](#) &rhs) -> [multivector_t](#) &=0
Geometric product.
- virtual auto [operator%=-](#) (const [multivector_t](#) &rhs) -> [multivector_t](#) &=0
Contraction.
- virtual auto [operator&=-](#) (const [multivector_t](#) &rhs) -> [multivector_t](#) &=0
Inner product.
- virtual auto [operator^=-](#) (const [multivector_t](#) &rhs) -> [multivector_t](#) &=0
Outer product.
- virtual auto [operator/=](#) (const Scalar_T &scr) -> [multivector_t](#) &=0
Quotient of multivector and scalar.
- virtual auto [operator/=](#) (const [multivector_t](#) &rhs) -> [multivector_t](#) &=0
Geometric quotient.
- virtual auto [operator|=](#) (const [multivector_t](#) &rhs) -> [multivector_t](#) &=0
Transformation via twisted adjoint action.
- virtual auto [inv](#) () const -> const [multivector_t](#)=0

- *Geometric multiplicative inverse.*
- virtual auto `pow` (int m) const -> const `multivector_t=0`
**this to the m*
- virtual auto `outer_pow` (int m) const -> const `multivector_t=0`
Outer product power.
- virtual auto `frame` () const -> const `index_set_t=0`
Subalgebra generated by all generators of terms of given multivector.
- virtual auto `grade` () const -> `index_t=0`
Maximum of the grades of each term.
- virtual auto `operator[]` (const `index_set_t` ist) const -> `Scalar_T=0`
Subscripting: map from index set to scalar coordinate.
- virtual auto `operator()` (`index_t` grade) const -> const `multivector_t=0`
Pure grade-vector part.
- virtual auto `scalar` () const -> `Scalar_T=0`
Scalar part.
- virtual auto `pure` () const -> const `multivector_t=0`
Pure part.
- virtual auto `even` () const -> const `multivector_t=0`
Even part of multivector, sum of even grade terms.
- virtual auto `odd` () const -> const `multivector_t=0`
Odd part of multivector, sum of odd grade terms.
- virtual auto `vector_part` () const -> const `vector_t=0`
Vector part of multivector, as a vector_t with respect to frame()
- virtual auto `vector_part` (const `index_set_t` frm, const bool prechecked) const -> const `vector_t=0`
Vector part of multivector, as a vector_t with respect to frm.
- virtual auto `involute` () const -> const `multivector_t=0`
Main involution, each {i} is replaced by -{i} in each term, eg. {1} -> -{1}.
- virtual auto `reverse` () const -> const `multivector_t=0`
Reversion, eg. {1}{2} -> {2}*{1}.*
- virtual auto `conj` () const -> const `multivector_t=0`
Conjugation, reverse o involute == involute o reverse.
- virtual auto `quad` () const -> `Scalar_T=0`
*Scalar_T quadratic form == (rev(x)*x)(0)*
- virtual auto `norm` () const -> `Scalar_T=0`
Scalar_T norm == sum of norm of coordinates.
- virtual auto `max_abs` () const -> `Scalar_T=0`
Maximum of absolute values of components of multivector: multivector infinity norm.
- virtual auto `truncated` (const `Scalar_T` &limit=`default_truncation`) const -> const `multivector_t=0`
Remove all terms with relative size smaller than limit.
- virtual auto `isinf` () const -> bool=0
Check if a multivector contains any infinite values.
- virtual auto `isnan` () const -> bool=0
Check if a multivector contains any IEEE NaN values.
- virtual void `write` (const std::string &msg="") const =0
Write formatted multivector to output.
- virtual void `write` (std::ofstream &ofile, const std::string &msg="") const =0
Write formatted multivector to file.

Static Public Member Functions

- static auto `classname` () -> const std::string

Static Public Attributes

- static const [index_t v_lo](#) = index_set_t::v_lo
- static const [index_t v_hi](#) = index_set_t::v_hi
- static const Scalar_T [default_truncation](#) = std::numeric_limits<Scalar_T>::epsilon()

Default for truncation.

6.4.1 Detailed Description

```
template<typename Scalar_T, typename Index_Set_T, typename Multivector_T>
class glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >
```

clifford_algebra<> declares the operations of a Clifford algebra

Definition at line 45 of file [clifford_algebra.h](#).

6.4.2 Member Typedef Documentation

6.4.2.1 index_set_t

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
using glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::index_set_t = Index_↵
Set_T
```

Definition at line 49 of file [clifford_algebra.h](#).

6.4.2.2 multivector_t

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
using glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::multivector_t = Multivector_↵
_T
```

Definition at line 52 of file [clifford_algebra.h](#).

6.4.2.3 pair_t

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
using glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::pair_t = std::pair<const
index_set_t, Scalar_T>
```

Definition at line 53 of file [clifford_algebra.h](#).

6.4.2.4 scalar_t

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
using glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::scalar_t = Scalar_T
```

Definition at line 48 of file [clifford_algebra.h](#).

6.4.2.5 vector_t

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
using glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::vector_t = std::vector<Scalar_T>
```

Definition at line 54 of file [clifford_algebra.h](#).

6.4.3 Constructor & Destructor Documentation

6.4.3.1 ~clifford_algebra()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::~~clifford_algebra
() [virtual], [default]
```

6.4.4 Member Function Documentation

6.4.4.1 classname()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::classname () -> const
std::string [static]
```

Definition at line 66 of file [clifford_algebra_imp.h](#).

6.4.4.2 conj()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::conj () const
-> const multivector_t [pure virtual]
```

Conjugation, reverse o involute == involute o reverse.

6.4.4.3 even()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::even () const
-> const multivector_t [pure virtual]
```

Even part of multivector, sum of even grade terms.

6.4.4.4 frame()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::frame () const
-> const index_set_t [pure virtual]
```

Subalgebra generated by all generators of terms of given multivector.

Referenced by [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi\(\)](#), [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >](#) and [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi\(\)](#).

6.4.4.5 grade()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::grade () const
-> index_t [pure virtual]
```

Maximum of the grades of each term.

6.4.4.6 inv()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::inv () const
-> const multivector_t [pure virtual]
```

Geometric multiplicative inverse.

6.4.4.7 involute()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::involute ()
const -> const multivector_t [pure virtual]
```

Main involution, each {i} is replaced by -{i} in each term, eg. {1} -> -{1}.

6.4.4.8 isinf()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isinf () const
-> bool [pure virtual]
```

Check if a multivector contains any infinite values.

6.4.4.9 isnan()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan () const
-> bool [pure virtual]
```

Check if a multivector contains any IEEE NaN values.

Referenced by [glucat::cascade_log\(\)](#), [glucat::exp\(\)](#), [glucat::log\(\)](#), [glucat::matrix_log\(\)](#), [glucat::matrix_sqrt\(\)](#), [glucat::pade_log\(\)](#), and [glucat::sqrt\(\)](#).

6.4.4.10 max_abs()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::max_abs ()
const -> Scalar_T [pure virtual]
```

Maximum of absolute values of components of multivector: multivector infinity norm.

6.4.4.11 norm()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::norm () const
-> Scalar_T [pure virtual]
```

Scalar_T norm == sum of norm of coordinates.

Referenced by [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi\(\)](#).

6.4.4.12 odd()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::odd () const
-> const multivector\_t [pure virtual]
```

Odd part of multivector, sum of odd grade terms.

6.4.4.13 operator%=()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator%= (
    const multivector\_t & rhs) -> multivector\_t & [pure virtual]
```

Contraction.

6.4.4.14 operator&=()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator&= (
    const multivector\_t & rhs) -> multivector\_t & [pure virtual]
```

Inner product.

6.4.4.15 operator()()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator() (
    index\_t grade) const -> const multivector\_t [pure virtual]
```

Pure grade-vector part.

6.4.4.16 operator*=() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator*= (
    const multivector\_t & rhs) -> multivector\_t & [pure virtual]
```

Geometric product.

6.4.4.17 operator*=() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator*= (
    const Scalar_T & scr) -> multivector_t & [pure virtual]
```

Product of multivector and scalar.

6.4.4.18 operator+=() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator+= (
    const multivector_t & rhs) -> multivector_t & [pure virtual]
```

Geometric sum.

6.4.4.19 operator+=() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator+= (
    const Scalar_T & scr) -> multivector_t & [pure virtual]
```

Geometric sum of multivector and scalar.

6.4.4.20 operator-()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator- (
    const -> const multivector_t [pure virtual]
```

Unary -.

6.4.4.21 operator-=() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator-= (
    const multivector_t & rhs) -> multivector_t & [pure virtual]
```

Geometric difference.

6.4.4.22 operator-=() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator-= (
    const Scalar_T & scr) -> multivector_t & [pure virtual]
```

Geometric difference of multivector and scalar.

6.4.4.23 operator/=() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator/= (
    const multivector\_t & rhs) -> multivector\_t & [pure virtual]
```

Geometric quotient.

6.4.4.24 operator/=() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator/= (
    const Scalar_T & scr) -> multivector\_t & [pure virtual]
```

Quotient of multivector and scalar.

6.4.4.25 operator==() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator==(
    const multivector\_t & val) const -> bool [pure virtual]
```

Test for equality of multivectors.

6.4.4.26 operator==() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator==(
    const Scalar_T & scr) const -> bool [pure virtual]
```

Test for equality of multivector and scalar.

6.4.4.27 operator[]()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator[] (
    const index\_set\_t ist) const -> Scalar_T [pure virtual]
```

Subscripting: map from index set to scalar coordinate.

6.4.4.28 operator^=()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator^= (
    const multivector\_t & rhs) -> multivector\_t & [pure virtual]
```

Outer product.

6.4.4.29 operator" |=(

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::operator|= (
    const multivector_t & rhs) -> multivector_t & [pure virtual]
```

Transformation via twisted adjoint action.

6.4.4.30 outer_pow()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::outer_pow (
    int m) const -> const multivector_t [pure virtual]
```

Outer product power.

6.4.4.31 pow()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::pow (
    int m) const -> const multivector_t [pure virtual]
```

*this to the m

6.4.4.32 pure()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::pure () const
-> const multivector_t [pure virtual]
```

Pure part.

6.4.4.33 quad()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::quad () const
-> Scalar_T [pure virtual]
```

Scalar_T quadratic form == (rev(x)*x)(0)

6.4.4.34 reverse()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::reverse ()
const -> const multivector_t [pure virtual]
```

Reversion, eg. {1}*{2} -> {2}*{1}.

6.4.4.35 scalar()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::scalar () const
-> Scalar_T [pure virtual]
```

Scalar part.

Referenced by [glucat::exp\(\)](#), [glucat::matrix_log\(\)](#), and [glucat::matrix_sqrt\(\)](#).

6.4.4.36 truncated()

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::truncated (
    const Scalar_T & limit = default\_truncation) const -> const multivector\_t [pure
virtual]
```

Remove all terms with relative size smaller than limit.

Referenced by [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), and [glucat::operator<<\(\)](#).

6.4.4.37 vector_part() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::vector_part ()
const -> const vector\_t [pure virtual]
```

Vector part of multivector, as a [vector_t](#) with respect to [frame\(\)](#)

6.4.4.38 vector_part() [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual auto glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::vector_part (
    const index\_set\_t frm,
    const bool prechecked) const -> const vector\_t [pure virtual]
```

Vector part of multivector, as a [vector_t](#) with respect to *frm*.

6.4.4.39 write() [1/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual void glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::write (
    const std::string & msg = "") const [pure virtual]
```

Write formatted multivector to output.

6.4.4.40 `write()` [2/2]

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
virtual void glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::write (
    std::ostream & ofile,
    const std::string & msg = "") const [pure virtual]
```

Write formatted multivector to file.

6.4.5 Member Data Documentation

6.4.5.1 `default_truncation`

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
const Scalar_T glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::default_↵
truncation = std::numeric_limits<Scalar_T>::epsilon() [static]
```

Default for truncation.

Definition at line 59 of file [clifford_algebra.h](#).

6.4.5.2 `v_hi`

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
const index\_t glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::v_hi = index↵
_set_t::v_hi [static]
```

Definition at line 51 of file [clifford_algebra.h](#).

6.4.5.3 `v_lo`

```
template<typename Scalar_T , typename Index_Set_T , typename Multivector_T >
const index\_t glucat::clifford\_algebra< Scalar_T, Index_Set_T, Multivector_T >::v_lo = index↵
_set_t::v_lo [static]
```

Definition at line 50 of file [clifford_algebra.h](#).

The documentation for this class was generated from the following files:

- [glucat/clifford_algebra.h](#)
- [glucat/clifford_algebra_imp.h](#)

6.5 `glucat::compare_types< LHS_T, RHS_T >` Class Template Reference

Type comparison.

```
#include <global.h>
```

Public Types

- enum { [are_same](#) = false }

6.5.1 Detailed Description

```
template<typename LHS_T, typename RHS_T>
class glucat::compare_types< LHS_T, RHS_T >
```

Type comparison.

Definition at line 54 of file [global.h](#).

6.5.2 Member Enumeration Documentation

6.5.2.1 anonymous enum

```
template<typename LHS_T , typename RHS_T >
anonymous enum
```

Enumerator

are_same	
--------------------------	--

Definition at line 57 of file [global.h](#).

The documentation for this class was generated from the following file:

- [glucat/global.h](#)

6.6 glucat::compare_types< T, T > Class Template Reference

```
#include <global.h>
```

Public Types

- enum { [are_same](#) = true }

6.6.1 Detailed Description

```
template<typename T>
class glucat::compare_types< T, T >
```

Definition at line 60 of file [global.h](#).

6.6.2 Member Enumeration Documentation

6.6.2.1 anonymous enum

```
template<typename T >
anonymous enum
```

Enumerator

are_same	
----------	--

Definition at line 63 of file [global.h](#).

The documentation for this class was generated from the following file:

- [glucat/global.h](#)

6.7 glucat::control_t Class Reference

Parameters to control tests.

```
#include <control.h>
```

Public Member Functions

- `int call (intfn f) const`
Call a function that returns int.
- `int call (intintfn f, int arg) const`
Call a function of int that returns int.

Static Public Member Functions

- `static const control_t & control (int argc, char **argv)`
- `static bool verbose ()`
Produce more detailed output from tests.

Private Member Functions

- `bool valid () const`
- `bool catch_exceptions () const`
- `control_t (int argc, char **argv)`
Constructor from program arguments.
- `control_t ()=default`
- `~control_t ()=default`
- `control_t (const control_t &)=delete`
- `control_t & operator= (const control_t &)=delete`

Private Attributes

- `bool m_valid`
Test parameters are valid.
- `bool m_catch_exceptions`
Catch exceptions.

Static Private Attributes

- static bool [m_verbose_output](#) = false
Produce more detailed output from tests.

Friends

- class [friend_for_private_destructor](#)

6.7.1 Detailed Description

Parameters to control tests.

Definition at line 39 of file [control.h](#).

6.7.2 Constructor & Destructor Documentation

6.7.2.1 [control_t\(\)](#) [1/3]

```
glucat::control_t::control_t (  
    int argc,  
    char ** argv) [private]
```

Constructor from program arguments.

Test control constructor from program arguments.

Definition at line 88 of file [control.h](#).

References [GLUCAT_PACKAGE_NAME](#), [GLUCAT_VERSION](#), [m_catch_exceptions](#), [m_valid](#), [m_verbose_output](#), and [valid\(\)](#).

6.7.2.2 [control_t\(\)](#) [2/3]

```
glucat::control_t::control_t () [private], [default]
```

6.7.2.3 [~control_t\(\)](#)

```
glucat::control_t::~~control_t () [private], [default]
```

6.7.2.4 [control_t\(\)](#) [3/3]

```
glucat::control_t::control_t (  
    const control\_t & ) [private], [delete]
```

6.7.3 Member Function Documentation

6.7.3.1 call() [1/2]

```
int glucat::control_t::call (  
    intfn f) const [inline]
```

Call a function that returns int.

Definition at line 136 of file [control.h](#).

References [catch_exceptions\(\)](#), [glucat::try_catch\(\)](#), and [valid\(\)](#).

6.7.3.2 call() [2/2]

```
int glucat::control_t::call (  
    intintfn f,  
    int arg) const [inline]
```

Call a function of int that returns int.

Definition at line 150 of file [control.h](#).

References [catch_exceptions\(\)](#), [glucat::try_catch\(\)](#), and [valid\(\)](#).

6.7.3.3 catch_exceptions()

```
bool glucat::control_t::catch_exceptions () const [inline], [private]
```

Definition at line 49 of file [control.h](#).

References [m_catch_exceptions](#).

Referenced by [call\(\)](#), and [call\(\)](#).

6.7.3.4 control()

```
static const control_t & glucat::control_t::control (  
    int argc,  
    char ** argv) [inline], [static]
```

Single instance Ref: Scott Meyers, "Effective C++" Second Edition, Addison-Wesley, 1998.

Definition at line 71 of file [control.h](#).

6.7.3.5 operator=()

```
control_t & glucat::control_t::operator= (  
    const control_t & ) [private], [delete]
```

6.7.3.6 valid()

```
bool glucat::control_t::valid () const [inline], [private]
```

Definition at line 44 of file [control.h](#).

References [m_valid](#).

Referenced by [call\(\)](#), [call\(\)](#), and [control_t\(\)](#).

6.7.3.7 verbose()

```
static bool glucat::control_t::verbose () [inline], [static]
```

Produce more detailed output from tests.

Definition at line 80 of file [control.h](#).

References [m_verbose_output](#).

6.7.4 Friends And Related Symbol Documentation

6.7.4.1 friend_for_private_destructor

```
friend class friend_for_private_destructor [friend]
```

Friend declaration to avoid compiler warning: "... only defines a private destructor and has no friends" Ref: Carlos O'Ryan, ACE <http://doc.ece.uci.edu>

Definition at line 67 of file [control.h](#).

6.7.5 Member Data Documentation

6.7.5.1 m_catch_exceptions

```
bool glucat::control_t::m_catch_exceptions [private]
```

Catch exceptions.

Definition at line 48 of file [control.h](#).

Referenced by [catch_exceptions\(\)](#), and [control_t\(\)](#).

6.7.5.2 m_valid

```
bool glucat::control_t::m_valid [private]
```

Test parameters are valid.

Definition at line 43 of file [control.h](#).

Referenced by [control_t\(\)](#), and [valid\(\)](#).

6.7.5.3 m_verbose_output

```
bool glucat::control_t::m_verbose_output = false [static], [private]
```

Produce more detailed output from tests.

Definition at line 53 of file [control.h](#).

Referenced by [control_t\(\)](#), and [verbose\(\)](#).

The documentation for this class was generated from the following file:

- test/[control.h](#)

6.8 glucat::CTAssertion< bool > Struct Template Reference

Compile time assertion.

6.8.1 Detailed Description

```
template<bool>
struct glucat::CTAssertion< bool >
```

Compile time assertion.

Definition at line 46 of file [global.h](#).

The documentation for this struct was generated from the following file:

- glucat/[global.h](#)

6.9 glucat::CTAssertion< true > Struct Reference

```
#include <global.h>
```

6.9.1 Detailed Description

Definition at line 47 of file [global.h](#).

The documentation for this struct was generated from the following file:

- glucat/[global.h](#)

6.10 `glucat::numeric_traits< Scalar_T >::demoted` Struct Reference

Demoted type for long double.

```
#include <promotion.h>
```

Public Types

- using `type` = float

6.10.1 Detailed Description

```
template<typename Scalar_T>
struct glucat::numeric_traits< Scalar_T >::demoted
```

Demoted type for long double.

Demoted type.

Definition at line 148 of file [scalar.h](#).

6.10.2 Member Typedef Documentation

6.10.2.1 `type`

```
template<typename Scalar_T >
typedef float glucat::numeric_traits< Scalar_T >::demoted::type = float
```

Definition at line 78 of file [promotion.h](#).

The documentation for this struct was generated from the following files:

- [glucat/promotion.h](#)
- [glucat/scalar.h](#)

6.11 `glucat::matrix::eig_genus< Matrix_T >` Struct Template Reference

Structure containing classification of eigenvalues.

```
#include <matrix.h>
```

Public Types

- using `Scalar_T` = typename `Matrix_T::value_type`

Public Attributes

- bool `m_is_singular` = false
Is the matrix singular?
- `eig_case_t` `m_eig_case` = safe_eigs
What kind of eigenvalues does the matrix contain?
- `Scalar_T` `m_safe_arg` = `Scalar_T`(0)
Argument such that $\exp(\pi \cdot m_safe_arg)$ lies between arguments of eigenvalues.

6.11.1 Detailed Description

```
template<typename Matrix_T>
struct glucat::matrix::eig_genus< Matrix_T >
```

Structure containing classification of eigenvalues.

Definition at line 140 of file [matrix.h](#).

6.11.2 Member Typedef Documentation

6.11.2.1 Scalar_T

```
template<typename Matrix_T >
using glucat::matrix::eig_genus< Matrix_T >::Scalar_T = typename Matrix_T::value_type
```

Definition at line 142 of file [matrix.h](#).

6.11.3 Member Data Documentation

6.11.3.1 m_eig_case

```
template<typename Matrix_T >
eig_case_t glucat::matrix::eig_genus< Matrix_T >::m_eig_case = safe_eigs
```

What kind of eigenvalues does the matrix contain?

Definition at line 146 of file [matrix.h](#).

Referenced by [glucat::matrix::classify_eigenvalues\(\)](#).

6.11.3.2 m_is_singular

```
template<typename Matrix_T >
bool glucat::matrix::eig_genus< Matrix_T >::m_is_singular = false
```

Is the matrix singular?

Definition at line 144 of file [matrix.h](#).

Referenced by [glucat::matrix::classify_eigenvalues\(\)](#).

6.11.3.3 m_safe_arg

```
template<typename Matrix_T >  
Scalar_T glucat::matrix::eig_genus< Matrix_T >::m_safe_arg = Scalar_T(0)
```

Argument such that $\exp(\pi \cdot m_safe_arg)$ lies between arguments of eigenvalues.

Definition at line 148 of file [matrix.h](#).

Referenced by [glucat::matrix::classify_eigenvalues\(\)](#).

The documentation for this struct was generated from the following file:

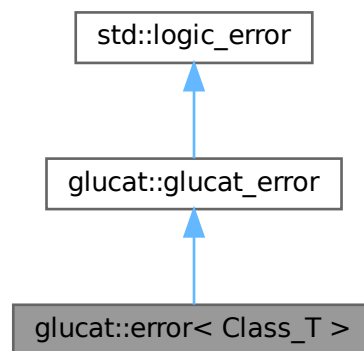
- [glucat/matrix.h](#)

6.12 glucat::error< Class_T > Class Template Reference

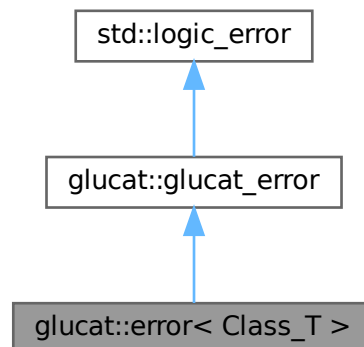
Specific exception class.

```
#include <errors.h>
```

Inheritance diagram for `glucat::error< Class_T >`:



Collaboration diagram for glucat::error< Class_T >:



Public Member Functions

- [error](#) (const std::string &msg)
Specific exception class.
- [error](#) (const std::string &context, const std::string &msg)
- auto [heading](#) () const noexcept -> const std::string override
- auto [classname](#) () const noexcept -> const std::string override
- void [print_error_msg](#) () const override

Public Member Functions inherited from [glucat::glucat_error](#)

- [glucat_error](#) (const std::string &context, const std::string &msg)
- [~glucat_error](#) () noexcept override=default

Additional Inherited Members

Public Attributes inherited from [glucat::glucat_error](#)

- std::string [name](#)

6.12.1 Detailed Description

```
template<class Class_T>
class glucat::error< Class_T >
```

Specific exception class.

Definition at line 56 of file [errors.h](#).

6.12.2 Constructor & Destructor Documentation

6.12.2.1 error() [1/2]

```
template<class Class_T >
glucat::error< Class_T >::error (
    const std::string & msg)
```

Specific exception class.

Definition at line 44 of file [errors_imp.h](#).

6.12.2.2 error() [2/2]

```
template<class Class_T >
glucat::error< Class_T >::error (
    const std::string & context,
    const std::string & msg)
```

Definition at line 50 of file [errors_imp.h](#).

6.12.3 Member Function Documentation

6.12.3.1 classname()

```
template<class Class_T >
auto glucat::error< Class_T >::classname () const -> const std::string [override], [virtual],
[noexcept]
```

Implements [glucat::glucat_error](#).

Definition at line 63 of file [errors_imp.h](#).

6.12.3.2 heading()

```
template<class Class_T >
auto glucat::error< Class_T >::heading () const -> const std::string [override], [virtual],
[noexcept]
```

Implements [glucat::glucat_error](#).

Definition at line 57 of file [errors_imp.h](#).

6.12.3.3 print_error_msg()

```
template<class Class_T >
void glucat::error< Class_T >::print_error_msg () const [override], [virtual]
```

Implements [glucat::glucat_error](#).

Definition at line 69 of file [errors_imp.h](#).

The documentation for this class was generated from the following files:

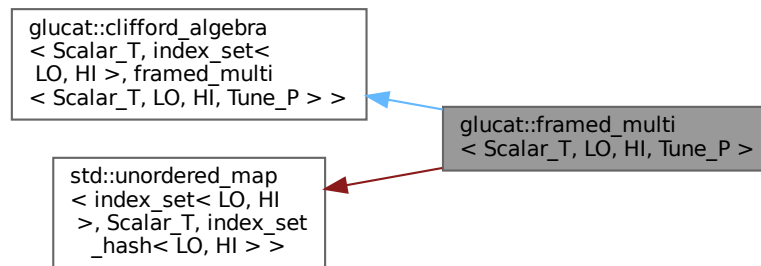
- [glucat/errors.h](#)
- [glucat/errors_imp.h](#)

6.13 glucat::framed_multi< Scalar_T, LO, HI, Tune_P > Class Template Reference

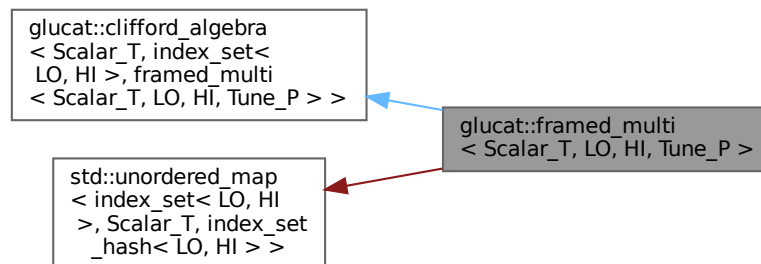
A framed_multi<Scalar_T,LO,HI,Tune_P> is a framed approximation to a multivector.

```
#include <framed_multi.h>
```

Inheritance diagram for glucat::framed_multi< Scalar_T, LO, HI, Tune_P >:



Collaboration diagram for glucat::framed_multi< Scalar_T, LO, HI, Tune_P >:



Classes

- class `hash_size_t`
- class `var_term`

Variable term.

Public Types

- using `multivector_t` = `framed_multi`
- using `framed_multi_t` = `multivector_t`
- using `scalar_t` = `Scalar_T`
- using `tune_p` = `Tune_P`
- using `index_set_t` = `index_set<LO, HI>`
- using `term_t` = `std::pair<const index_set_t, Scalar_T>`
- using `vector_t` = `std::vector<Scalar_T>`
- using `error_t` = `error<multivector_t>`
- using `matrix_multi_t` = `matrix_multi<Scalar_T,LO,HI,Tune_P >`

Public Types inherited from

glucat::clifford_algebra< **Scalar_T**, **index_set**< **LO**, **HI** >, **framed_multi**< **Scalar_T**, **LO**, **HI**, **Tune_P** > >

- using [scalar_t](#)
- using [index_set_t](#)
- using [multivector_t](#)
- using [pair_t](#)
- using [vector_t](#)

Public Member Functions

- [~framed_multi](#) () override=default
Destructor.
- [framed_multi](#) ()
Default constructor.
- template<typename Other_Scalar_T >
[framed_multi](#) (const [framed_multi](#)< Other_Scalar_T, LO, HI, Tune_P > &val)
Construct a multivector from a multivector with a different scalar type.
- template<typename Other_Scalar_T >
[framed_multi](#) (const [framed_multi](#)< Other_Scalar_T, LO, HI, Tune_P > &val, const [index_set_t](#) frm, const bool prechecked=false)
Construct a multivector, within a given frame, from a given multivector.
- [framed_multi](#) (const [framed_multi_t](#) &val, const [index_set_t](#) frm, const bool prechecked=false)
Construct a multivector, within a given frame, from a given multivector.
- [framed_multi](#) (const [index_set_t](#) ist, const [Scalar_T](#) &crd=[Scalar_T](#)(1))
Construct a multivector from an index set and a scalar coordinate.
- [framed_multi](#) (const [index_set_t](#) ist, const [Scalar_T](#) &crd, const [index_set_t](#) frm, const bool prechecked=false)
Construct a multivector, within a given frame, from an index set and a scalar coordinate.
- [framed_multi](#) (const [Scalar_T](#) &scr, const [index_set_t](#) frm=[index_set_t](#)())
Construct a multivector from a scalar (within a frame, if given)
- [framed_multi](#) (const int scr, const [index_set_t](#) frm=[index_set_t](#)())
Construct a multivector from an int (within a frame, if given)
- [framed_multi](#) (const [vector_t](#) &vec, const [index_set_t](#) frm, const bool prechecked=false)
Construct a multivector, within a given frame, from a given vector.
- [framed_multi](#) (const std::string &str)
Construct a multivector from a string: eg: "3+2{1,2}-6.1e-2{2,3}".
- [framed_multi](#) (const std::string &str, const [index_set_t](#) frm, const bool prechecked=false)
Construct a multivector, within a given frame, from a string: eg: "3+2{1,2}-6.1e-2{2,3}".
- [framed_multi](#) (const char *str)
Construct a multivector from a char: eg: "3+2{1,2}-6.1e-2{2,3}".*
- [framed_multi](#) (const char *str, const [index_set_t](#) frm, const bool prechecked=false)
Construct a multivector, within a given frame, from a char: eg: "3+2{1,2}-6.1e-2{2,3}".*
- template<typename Other_Scalar_T >
[framed_multi](#) (const [matrix_multi](#)< Other_Scalar_T, LO, HI, Tune_P > &val)
Construct a multivector from a matrix_multi_t.
- template<typename Other_Scalar_T >
auto [fast_matrix_multi](#) (const [index_set_t](#) frm) const -> const [matrix_multi](#)< Other_Scalar_T, LO, HI, Tune_P >
Use generalized FFT to construct a matrix_multi_t.
- auto [fast_framed_multi](#) () const -> const [framed_multi_t](#)
Use inverse generalized FFT to construct a framed_multi_t.
- [_GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS](#) auto [nbr_terms](#) () const -> unsigned long
Number of terms.
- auto [operator+=](#) (const [term_t](#) &term) -> [multivector_t](#) &
Add a term, if non-zero.

Public Member Functions inherited from**glucat::clifford_algebra< Scalar_T, index_set< LO, HI >, framed_multi< Scalar_T, LO, HI, Tune_P > >**

- virtual `~clifford_algebra()` = default
- virtual auto `operator==` (const `multivector_t` &val) const -> bool=0
Test for equality of multivectors.
- virtual auto `operator==` (const `Scalar_T` &scr) const -> bool=0
Test for equality of multivector and scalar.
- virtual auto `operator+=` (const `multivector_t` &rhs) -> `multivector_t` &=0
Geometric sum.
- virtual auto `operator+=` (const `Scalar_T` &scr) -> `multivector_t` &=0
Geometric sum of multivector and scalar.
- virtual auto `operator-=` (const `multivector_t` &rhs) -> `multivector_t` &=0
Geometric difference.
- virtual auto `operator-=` (const `Scalar_T` &scr) -> `multivector_t` &=0
Geometric difference of multivector and scalar.
- virtual auto `operator-` () const -> const `multivector_t`=0
Unary -.
- virtual auto `operator*=` (const `Scalar_T` &scr) -> `multivector_t` &=0
Product of multivector and scalar.
- virtual auto `operator*=` (const `multivector_t` &rhs) -> `multivector_t` &=0
Geometric product.
- virtual auto `operator%=>` (const `multivector_t` &rhs) -> `multivector_t` &=0
Contraction.
- virtual auto `operator&=>` (const `multivector_t` &rhs) -> `multivector_t` &=0
Inner product.
- virtual auto `operator^=>` (const `multivector_t` &rhs) -> `multivector_t` &=0
Outer product.
- virtual auto `operator/=` (const `Scalar_T` &scr) -> `multivector_t` &=0
Quotient of multivector and scalar.
- virtual auto `operator/=` (const `multivector_t` &rhs) -> `multivector_t` &=0
Geometric quotient.
- virtual auto `operator|=` (const `multivector_t` &rhs) -> `multivector_t` &=0
Transformation via twisted adjoint action.
- virtual auto `inv` () const -> const `multivector_t`=0
Geometric multiplicative inverse.
- virtual auto `pow` (int m) const -> const `multivector_t`=0
**this to the m*
- virtual auto `outer_pow` (int m) const -> const `multivector_t`=0
Outer product power.
- virtual auto `frame` () const -> const `index_set_t`=0
Subalgebra generated by all generators of terms of given multivector.
- virtual auto `grade` () const -> `index_t`=0
Maximum of the grades of each term.
- virtual auto `operator[]` (const `index_set_t` ist) const -> `Scalar_T`=0
Subscripting: map from index set to scalar coordinate.
- virtual auto `operator()` (`index_t` grade) const -> const `multivector_t`=0
Pure grade-vector part.
- virtual auto `scalar` () const -> `Scalar_T`=0
Scalar part.
- virtual auto `pure` () const -> const `multivector_t`=0

- virtual auto `even` () const -> const `multivector_t=0`
Pure part.
- virtual auto `odd` () const -> const `multivector_t=0`
Even part of multivector, sum of even grade terms.
- virtual auto `vector_part` () const -> const `vector_t=0`
Odd part of multivector, sum of odd grade terms.
- virtual auto `vector_part` (const `index_set_t` frm, const bool prechecked) const -> const `vector_t=0`
Vector part of multivector, as a `vector_t` with respect to `frame()`
- virtual auto `involute` () const -> const `multivector_t=0`
Main involution, each $\{i\}$ is replaced by $-\{i\}$ in each term, eg. $\{1\} \rightarrow -\{1\}$.
- virtual auto `reverse` () const -> const `multivector_t=0`
Reversion, eg. $\{1\}\{2\} \rightarrow \{2\}*\{1\}$.*
- virtual auto `conj` () const -> const `multivector_t=0`
Conjugation, reverse o involute == involute o reverse.
- virtual auto `quad` () const -> `Scalar_T=0`
*Scalar_T quadratic form == $(rev(x)*x)(0)$*
- virtual auto `norm` () const -> `Scalar_T=0`
Scalar_T norm == sum of norm of coordinates.
- virtual auto `max_abs` () const -> `Scalar_T=0`
Maximum of absolute values of components of multivector: multivector infinity norm.
- virtual auto `truncated` (const `Scalar_T` &limit=`default_truncation`) const -> const `multivector_t=0`
Remove all terms with relative size smaller than limit.
- virtual auto `isinf` () const -> bool=0
Check if a multivector contains any infinite values.
- virtual auto `isnan` () const -> bool=0
Check if a multivector contains any IEEE NaN values.
- virtual void `write` (const std::string &msg="") const=0
Write formatted multivector to output.
- virtual void `write` (std::ofstream &ofile, const std::string &msg="") const=0
Write formatted multivector to file.

Static Public Member Functions

- static auto `classname` () -> const std::string
Class name used in messages.
- static auto `random` (const `index_set_t` frm, `Scalar_T` fill=`Scalar_T(1)`) -> const `multivector_t`
Random multivector within a frame.

Static Public Member Functions inherited from

`glucat::clifford_algebra< Scalar_T, index_set< LO, HI >, framed_multi< Scalar_T, LO, HI, Tune_P > >`

- static auto `classname` () -> const std::string

Private Types

- using `var_term_t` = class `var_term`
- using `matrix_t` = typename `matrix_multi_t::matrix_t`
- using `sorted_map_t` = `std::map< index_set_t, Scalar_T, std::less<const index_set_t> >`
- using `map_t` = `std::unordered_map<index_set_t, Scalar_T, index_set_hash<LO, HI>>`
- using `framed_pair_t` = `std::pair<const multivector_t, const multivector_t>`
- using `size_type` = typename `map_t::size_type`
- using `iterator` = typename `map_t::iterator`
- using `const_iterator` = typename `map_t::const_iterator`

Private Member Functions

- `framed_multi` (const `hash_size_t` &hash_size)
Private constructor using hash_size.
- auto `fold` (const `index_set_t` frm) const -> `multivector_t`
Subalgebra isomorphism: fold each term within the given frame.
- auto `unfold` (const `index_set_t` frm) const -> `multivector_t`
Subalgebra isomorphism: unfold each term within the given frame.
- auto `centre_pm4_qp4` (`index_t` &p, `index_t` &q) -> `multivector_t` &
Subalgebra isomorphism: $R_{\{p,q\}}$ to $R_{\{p-4,q+4\}}$.
- auto `centre_pp4_qm4` (`index_t` &p, `index_t` &q) -> `multivector_t` &
Subalgebra isomorphism: $R_{\{p,q\}}$ to $R_{\{p+4,q-4\}}$.
- auto `centre_qp1_pm1` (`index_t` &p, `index_t` &q) -> `multivector_t` &
Subalgebra isomorphism: $R_{\{p,q\}}$ to $R_{\{q+1,p-1\}}$.
- auto `divide` (const `index_set_t` ist) const -> const `framed_pair_t`
Divide multivector into part divisible by `index_set` and remainder.
- auto `fast` (const `index_t` level, const bool odd) const -> const `matrix_t`
Generalized FFT from multivector_t to matrix_t.

Friends

- template<typename Other_Scalar_T, const `index_t` Other_LO, const `index_t` Other_HI, typename Other_Tune_P >
class `matrix_multi`
- template<typename Other_Scalar_T, const `index_t` Other_LO, const `index_t` Other_HI, typename Other_Tune_P >
class `framed_multi`
- auto `operator*` (const `multivector_t` &lhs, const `multivector_t` &rhs) -> const `multivector_t`
- auto `operator^` (const `multivector_t` &lhs, const `multivector_t` &rhs) -> const `multivector_t`
- auto `operator&` (const `multivector_t` &lhs, const `multivector_t` &rhs) -> const `multivector_t`
- auto `operator%` (const `multivector_t` &lhs, const `multivector_t` &rhs) -> const `multivector_t`
- auto `star` (const `multivector_t` &lhs, const `multivector_t` &rhs) -> `Scalar_T`
- auto `operator/` (const `multivector_t` &lhs, const `multivector_t` &rhs) -> const `multivector_t`
- auto `operator|` (const `multivector_t` &lhs, const `multivector_t` &rhs) -> const `multivector_t`
- auto `operator>>` (std::istream &s, `multivector_t` &val) -> std::istream &
- auto `operator<<` (std::ostream &os, const `multivector_t` &val) -> std::ostream &
- auto `operator<<` (std::ostream &os, const `term_t` &term) -> std::ostream &
- auto `exp` (const `multivector_t` &val) -> const `multivector_t`

Additional Inherited Members

Static Public Attributes inherited from

[glucat::clifford_algebra](#)< [Scalar_T](#), [index_set](#)< [LO](#), [HI](#) >, [framed_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) > >

- static const [index_t v_lo](#)
- static const [index_t v_hi](#)
- static const [Scalar_T default_truncation](#)

Default for truncation.

6.13.1 Detailed Description

```
template<typename Scalar_T, const index\_t LO, const index\_t HI, typename Tune_P>
class glucat::framed_multi< Scalar\_T, LO, HI, Tune\_P >
```

A [framed_multi](#)<[Scalar_T](#),[LO](#),[HI](#),[Tune_P](#)> is a framed approximation to a multivector.

Definition at line 55 of file [matrix_multi.h](#).

6.13.2 Member Typedef Documentation

6.13.2.1 const_iterator

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
using glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::const_iterator = typename map_t↔
::const_iterator [private]
```

Definition at line 167 of file [framed_multi.h](#).

6.13.2.2 error_t

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
using glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::error_t = error<multivector\_t>
```

Definition at line 138 of file [framed_multi.h](#).

6.13.2.3 framed_multi_t

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
using glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::framed_multi_t = multivector\_t
```

Definition at line 132 of file [framed_multi.h](#).

6.13.2.4 framed_pair_t

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
using glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::framed_pair_t = std::pair<const multivector\_t,
const multivector\_t> [private]
```

Definition at line 164 of file [framed_multi.h](#).

6.13.2.5 index_set_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::index_set_t = index_set<LO, HI>
```

Definition at line 135 of file [framed_multi.h](#).

6.13.2.6 iterator

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::iterator = typename map_t::iterator
[private]
```

Definition at line 166 of file [framed_multi.h](#).

6.13.2.7 map_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::map_t = std::unordered_map<index_set_t,
Scalar_T, index_set_hash<LO, HI>> [private]
```

Definition at line 150 of file [framed_multi.h](#).

6.13.2.8 matrix_multi_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi_t = matrix_multi<Scalar←
_T,LO,HI,Tune_P >
```

Definition at line 139 of file [framed_multi.h](#).

6.13.2.9 matrix_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::matrix_t = typename matrix_multi_t::matrix_t
[private]
```

Definition at line 148 of file [framed_multi.h](#).

6.13.2.10 multivector_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::multivector_t = framed_multi
```

Definition at line 131 of file [framed_multi.h](#).

6.13.2.11 scalar_t

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
using glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::scalar\_t = Scalar_T
```

Definition at line [133](#) of file [framed_multi.h](#).

6.13.2.12 size_type

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
using glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::size_type = typename map_t::size_type
[private]
```

Definition at line [165](#) of file [framed_multi.h](#).

6.13.2.13 sorted_map_t

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
using glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::sorted_map_t = std::map< index\_set\_t,
Scalar_T, std::less<const index\_set\_t> > [private]
```

Definition at line [149](#) of file [framed_multi.h](#).

6.13.2.14 term_t

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
using glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::term_t = std::pair<const index\_set\_t,
Scalar_T>
```

Definition at line [136](#) of file [framed_multi.h](#).

6.13.2.15 tune_p

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
using glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::tune_p = Tune_P
```

Definition at line [134](#) of file [framed_multi.h](#).

6.13.2.16 var_term_t

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
using glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::var_term_t = class var\_term [private]
```

Definition at line [147](#) of file [framed_multi.h](#).

6.13.2.17 vector_t

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
using glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::vector_t = std::vector<Scalar_T>
```

Definition at line [137](#) of file [framed_multi.h](#).

6.13.3 Constructor & Destructor Documentation

6.13.3.1 ~framed_multi()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::~~framed_multi () [override], [default]
```

Destructor.

6.13.3.2 framed_multi() [1/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::~framed_multi ()
```

Default constructor.

Definition at line [59](#) of file [framed_multi_imp.h](#).

6.13.3.3 framed_multi() [2/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::~framed_multi (
    const hash\_size\_t & hash_size) [private]
```

Private constructor using hash_size.

Definition at line [66](#) of file [framed_multi_imp.h](#).

6.13.3.4 framed_multi() [3/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::~framed_multi (
    const framed\_multi< Other_Scalar_T, LO, HI, Tune_P > & val)
```

Construct a multivector from a multivector with a different scalar type.

Definition at line [74](#) of file [framed_multi_imp.h](#).

6.13.3.5 framed_multi() [4/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
    const framed\_multi< Other_Scalar_T, LO, HI, Tune_P > & val,
    const index\_set\_t frm,
    const bool prechecked = false)
```

Construct a multivector, within a given frame, from a given multivector.

Definition at line 85 of file [framed_multi_imp.h](#).

References [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::frame\(\)](#).

6.13.3.6 framed_multi() [5/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
    const framed\_multi\_t & val,
    const index\_set\_t frm,
    const bool prechecked = false)
```

Construct a multivector, within a given frame, from a given multivector.

Definition at line 98 of file [framed_multi_imp.h](#).

References [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::frame\(\)](#).

6.13.3.7 framed_multi() [6/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
    const index\_set\_t ist,
    const Scalar_T & crd = Scalar_T(1))
```

Construct a multivector from an index set and a scalar coordinate.

Definition at line 111 of file [framed_multi_imp.h](#).

6.13.3.8 framed_multi() [7/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
    const index\_set\_t ist,
    const Scalar_T & crd,
    const index\_set\_t frm,
    const bool prechecked = false)
```

Construct a multivector, within a given frame, from an index set and a scalar coordinate.

Definition at line 121 of file [framed_multi_imp.h](#).

6.13.3.9 framed_multi() [8/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
    const Scalar_T & scr,
    const index\_set\_t frm = index\_set\_t() )
```

Construct a multivector from a scalar (within a frame, if given)

Definition at line [134](#) of file [framed_multi_imp.h](#).

6.13.3.10 framed_multi() [9/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
    const int scr,
    const index\_set\_t frm = index\_set\_t() )
```

Construct a multivector from an int (within a frame, if given)

Definition at line [144](#) of file [framed_multi_imp.h](#).

6.13.3.11 framed_multi() [10/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
    const vector\_t & vec,
    const index\_set\_t frm,
    const bool prechecked = false)
```

Construct a multivector, within a given frame, from a given vector.

Definition at line [154](#) of file [framed_multi_imp.h](#).

References [glucat::index_set< LO, HI >::count\(\)](#), [glucat::index_set< LO, HI >::max\(\)](#), and [glucat::index_set< LO, HI >::min\(\)](#).

6.13.3.12 framed_multi() [11/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
    const std::string & str)
```

Construct a multivector from a string: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line [176](#) of file [framed_multi_imp.h](#).

6.13.3.13 framed_multi() [12/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
    const std::string & str,
    const index\_set\_t frm,
    const bool prechecked = false)
```

Construct a multivector, within a given frame, from a string: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 192 of file [framed_multi_imp.h](#).

6.13.3.14 framed_multi() [13/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
    const char * str) [inline]
```

Construct a multivector from a char*: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 209 of file [framed_multi.h](#).

References [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi](#).

6.13.3.15 framed_multi() [14/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
    const char * str,
    const index\_set\_t frm,
    const bool prechecked = false) [inline]
```

Construct a multivector, within a given frame, from a char*: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 212 of file [framed_multi.h](#).

References [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi](#).

6.13.3.16 framed_multi() [15/15]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::framed_multi (
    const matrix\_multi< Other_Scalar_T, LO, HI, Tune_P > & val)
```

Construct a multivector from a [matrix_multi_t](#).

Definition at line 205 of file [framed_multi_imp.h](#).

References [_GLUCAT_HASH_SIZE_T](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_element\(\)](#), [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi\(\)](#), [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::inner\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix](#), [glucat::matrix::nnz\(\)](#), [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::truncated\(\)](#), and [glucat::clifford_algebra< Scalar_T, index_set< LO, HI >, framed_multi< Scalar_T, LO, HI, Tune_P > >::truncated\(\)](#).

6.13.4 Member Function Documentation

6.13.4.1 centre_pm4_qp4()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::centre_pm4_qp4 (
    index_t & p,
    index_t & q) -> multivector_t& [private]
```

Subalgebra isomorphism: $R_{\{p,q\}}$ to $R_{\{p-4,q+4\}}$.

Definition at line 1469 of file [framed_multi_imp.h](#).

Referenced by [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi\(\)](#).

6.13.4.2 centre_pp4_qm4()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::centre_pp4_qm4 (
    index_t & p,
    index_t & q) -> multivector_t& [private]
```

Subalgebra isomorphism: $R_{\{p,q\}}$ to $R_{\{p+4,q-4\}}$.

Definition at line 1511 of file [framed_multi_imp.h](#).

Referenced by [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi\(\)](#).

6.13.4.3 centre_qp1_pm1()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::centre_qp1_pm1 (
    index_t & p,
    index_t & q) -> multivector_t& [private]
```

Subalgebra isomorphism: $R_{\{p,q\}}$ to $R_{\{q+1,p-1\}}$.

Definition at line 1553 of file [framed_multi_imp.h](#).

Referenced by [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi\(\)](#).

6.13.4.4 classname()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::classname () -> const std::string
[static]
```

Class name used in messages.

Definition at line 50 of file [framed_multi_imp.h](#).

6.13.4.5 divide()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::divide (
    const index\_set\_t ist) const -> const framed\_pair\_t [private]
```

Divide multivector into part divisible by [index_set](#) and remainder.

Divide multivector into quotient with terms divisible by index set, and remainder.

Definition at line 1586 of file [framed_multi_imp.h](#).

6.13.4.6 fast()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::fast (
    const index\_t level,
    const bool odd) const -> const matrix\_t [private]
```

Generalized FFT from multivector_t to matrix_t.

Definition at line 1602 of file [framed_multi_imp.h](#).

References [glucat::matrix::kron\(\)](#), [glucat::odd\(\)](#), [glucat::scalar\(\)](#), and [glucat::matrix::unit\(\)](#).

6.13.4.7 fast_framed_multi()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi () const -> const
framed\_multi\_t [inline]
```

Use inverse generalized FFT to construct a framed_multi_t.

Definition at line 1700 of file [framed_multi_imp.h](#).

Referenced by [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi\(\)](#).

6.13.4.8 fast_matrix_multi()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
template<typename Other_Scalar_T >
auto glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::fast_matrix_multi (
    const index\_set\_t frm) const -> const matrix\_multi<Other_Scalar_T,LO,HI,Tune_P >
```

Use generalized FFT to construct a matrix_multi_t.

Definition at line 1668 of file [framed_multi_imp.h](#).

References [glucat::gen::offset_to_super](#), and [glucat::pos_mod\(\)](#).

6.13.4.9 fold()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::fold (
    const index\_set\_t frm) const -> multivector\_t [private]
```

Subalgebra isomorphism: fold each term within the given frame.

Definition at line [1434](#) of file [framed_multi_imp.h](#).

6.13.4.10 nbr_terms()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::nbr_terms () const -> unsigned long
```

Number of terms.

Definition at line [1356](#) of file [framed_multi_imp.h](#).

6.13.4.11 operator+=(())

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::operator+= (
    const term\_t & term) -> multivector\_t& [inline]
```

Add a term, if non-zero.

Insert a term into a multivector, add terms with same index set.

Geometric sum.

Geometric sum of multivector and scalar.

Definition at line [295](#) of file [framed_multi_imp.h](#).

6.13.4.12 random()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::random (
    const index\_set\_t frm,
    Scalar_T fill = Scalar_T(1)) -> const multivector\_t [static]
```

Random multivector within a frame.

Definition at line [1058](#) of file [framed_multi_imp.h](#).

Referenced by [glucat::matrix_multi](#)< Scalar_T, LO, HI, Tune_P >::random().

6.13.4.13 `unfold()`

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::unfold (
    const index\_set\_t frm) const -> multivector\_t [private]
```

Subalgebra isomorphism: unfold each term within the given frame.

Definition at line [1451](#) of file [framed_multi_imp.h](#).

Referenced by [glucat::matrix_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) >::[fast_framed_multi](#)()

6.13.5 Friends And Related Symbol Documentation

6.13.5.1 `exp`

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto exp (
    const multivector\_t & val) -> const multivector\_t [friend]
```

6.13.5.2 `framed_multi`

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
template<typename Other_Scalar_T , const index\_t Other_LO, const index\_t Other_HI, typename
Other_Tune_P >
friend class framed\_multi [friend]
```

Definition at line [143](#) of file [framed_multi.h](#).

Referenced by [glucat::framed_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) >::[framed_multi](#)(), and [glucat::framed_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) >::[fast_framed_multi](#)()

6.13.5.3 `matrix_multi`

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
template<typename Other_Scalar_T , const index\_t Other_LO, const index\_t Other_HI, typename
Other_Tune_P >
friend class matrix\_multi [friend]
```

Definition at line [141](#) of file [framed_multi.h](#).

6.13.5.4 `operator%`

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto operator% (
    const multivector\_t & lhs,
    const multivector\_t & rhs) -> const multivector\_t [friend]
```

6.13.5.5 operator&

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator& (
    const multivector_t & lhs,
    const multivector_t & rhs) -> const multivector_t [friend]
```

6.13.5.6 operator*

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator* (
    const multivector_t & lhs,
    const multivector_t & rhs) -> const multivector_t [friend]
```

6.13.5.7 operator/

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator/ (
    const multivector_t & lhs,
    const multivector_t & rhs) -> const multivector_t [friend]
```

6.13.5.8 operator<< [1/2]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator<< (
    std::ostream & os,
    const multivector_t & val) -> std::ostream & [friend]
```

6.13.5.9 operator<< [2/2]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator<< (
    std::ostream & os,
    const term_t & term) -> std::ostream & [friend]
```

6.13.5.10 operator>>

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator>> (
    std::istream & s,
    multivector_t & val) -> std::istream & [friend]
```

6.13.5.11 operator^

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto operator^ (
    const multivector_t & lhs,
    const multivector_t & rhs) -> const multivector_t [friend]
```

6.13.5.12 operator" |

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto operator| (
    const multivector\_t & lhs,
    const multivector\_t & rhs) -> const multivector\_t [friend]
```

6.13.5.13 star

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto star (
    const multivector\_t & lhs,
    const multivector\_t & rhs) -> Scalar_T [friend]
```

The documentation for this class was generated from the following files:

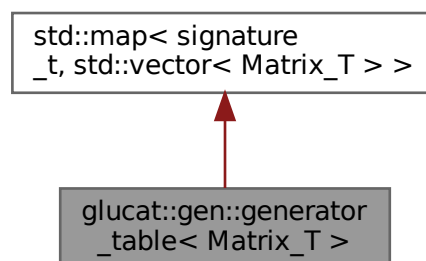
- [glucat/framed_multi.h](#)
- [glucat/matrix_multi.h](#)
- [glucat/framed_multi_imp.h](#)

6.14 [glucat::gen::generator_table< Matrix_T >](#) Class Template Reference

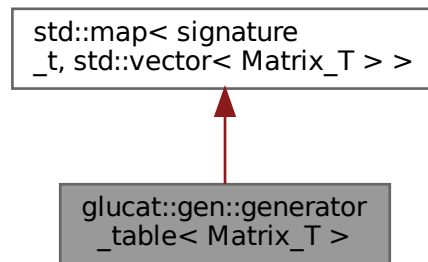
Table of generators for specific signatures.

```
#include <generation.h>
```

Inheritance diagram for [glucat::gen::generator_table< Matrix_T >](#):



Collaboration diagram for glucat::gen::generator_table< Matrix_T >:



Public Member Functions

- auto `operator()` (const `index_t` p, const `index_t` q) -> const `Matrix_T` *
Pointer to generators for a specific signature.
- `generator_table` (const `generator_table` &)=delete
- auto `operator=` (const `generator_table` &) -> `generator_table` &=delete

Static Public Member Functions

- static auto `generator` () -> `generator_table`< `Matrix_T` > &
Single instance of generator table.

Private Member Functions

- auto `gen_vector` (const `index_t` p, const `index_t` q) -> const `std::vector`< `Matrix_T` > &
Construct a vector of generators for a specific signature.
- void `gen_from_pm1_qm1` (const `std::vector`< `Matrix_T` > &old, const `signature_t` sig)
Construct generators for p,q given generators for p-1,q-1.
- void `gen_from_pm4_qp4` (const `std::vector`< `Matrix_T` > &old, const `signature_t` sig)
Construct generators for p,q given generators for p-4,q+4.
- void `gen_from_pp4_qm4` (const `std::vector`< `Matrix_T` > &old, const `signature_t` sig)
Construct generators for p,q given generators for p+4,q-4.
- void `gen_from_qp1_pm1` (const `std::vector`< `Matrix_T` > &old, const `signature_t` sig)
Construct generators for p,q given generators for q+1,p-1.
- `generator_table` ()=default
- `~generator_table` ()=default

Friends

- class `friend_for_private_destructor`

6.14.1 Detailed Description

```
template<class Matrix_T>
class glucat::gen::generator_table< Matrix_T >
```

Table of generators for specific signatures.

Definition at line 52 of file [generation.h](#).

6.14.2 Constructor & Destructor Documentation

6.14.2.1 generator_table() [1/2]

```
template<class Matrix_T >
glucat::gen::generator_table< Matrix_T >::generator_table () [private], [default]
```

6.14.2.2 ~generator_table()

```
template<class Matrix_T >
glucat::gen::generator_table< Matrix_T >::~~generator_table () [private], [default]
```

6.14.2.3 generator_table() [2/2]

```
template<class Matrix_T >
glucat::gen::generator_table< Matrix_T >::generator_table (
    const generator_table< Matrix_T > & ) [delete]
```

6.14.3 Member Function Documentation

6.14.3.1 gen_from_pm1_qm1()

```
template<class Matrix_T >
void glucat::gen::generator_table< Matrix_T >::gen_from_pm1_qm1 (
    const std::vector< Matrix_T > & old,
    const signature_t sig) [private]
```

Construct generators for p,q given generators for p-1,q-1.

Definition at line 127 of file [generation_imp.h](#).

References [glucat::matrix::mono_kron\(\)](#), and [glucat::matrix::unit\(\)](#).

6.14.3.2 `gen_from_pm4_qp4()`

```
template<class Matrix_T >
void glucat::gen::generator_table< Matrix_T >::gen_from_pm4_qp4 (
    const std::vector< Matrix_T > & old,
    const signature_t sig) [private]
```

Construct generators for p,q given generators for p-4,q+4.

Definition at line 165 of file `generation_imp.h`.

References `glucat::matrix::mono_prod()`.

6.14.3.3 `gen_from_pp4_qm4()`

```
template<class Matrix_T >
void glucat::gen::generator_table< Matrix_T >::gen_from_pp4_qm4 (
    const std::vector< Matrix_T > & old,
    const signature_t sig) [private]
```

Construct generators for p,q given generators for p+4,q-4.

Definition at line 198 of file `generation_imp.h`.

References `glucat::matrix::mono_prod()`.

6.14.3.4 `gen_from_qp1_pm1()`

```
template<class Matrix_T >
void glucat::gen::generator_table< Matrix_T >::gen_from_qp1_pm1 (
    const std::vector< Matrix_T > & old,
    const signature_t sig) [private]
```

Construct generators for p,q given generators for q+1,p-1.

Definition at line 231 of file `generation_imp.h`.

References `glucat::matrix::mono_prod()`.

6.14.3.5 `gen_vector()`

```
template<class Matrix_T >
auto glucat::gen::generator_table< Matrix_T >::gen_vector (
    const index_t p,
    const index_t q) -> const std::vector<Matrix_T>& [private]
```

Construct a vector of generators for a specific signature.

Definition at line 79 of file `generation_imp.h`.

References `glucat::pos_mod()`, and `glucat::matrix::unit()`.

6.14.3.6 generator()

```
template<class Matrix_T >
auto glucat::gen::generator_table< Matrix_T >::generator () -> generator_table<Matrix_T>&
[static]
```

Single instance of generator table.

Definition at line 49 of file [generation_imp.h](#).

Referenced by [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_element\(\)](#).

6.14.3.7 operator()()

```
template<class Matrix_T >
auto glucat::gen::generator_table< Matrix_T >::operator() (
    const index_t p,
    const index_t q) -> const Matrix_T* [inline]
```

Pointer to generators for a specific signature.

Definition at line 58 of file [generation_imp.h](#).

References [glucat::gen::offset_to_super](#), and [glucat::pos_mod\(\)](#).

6.14.3.8 operator=()

```
template<class Matrix_T >
auto glucat::gen::generator_table< Matrix_T >::operator= (
    const generator_table< Matrix_T > & ) -> generator_table &=delete [delete]
```

6.14.4 Friends And Related Symbol Documentation

6.14.4.1 friend_for_private_destructor

```
template<class Matrix_T >
friend class friend_for_private_destructor [friend]
```

Friend declaration to avoid compiler warning: "... only defines a private destructor and has no friends" Ref: Carlos O'Ryan, ACE <http://doc.ece.uci.edu>

Definition at line 75 of file [generation.h](#).

The documentation for this class was generated from the following files:

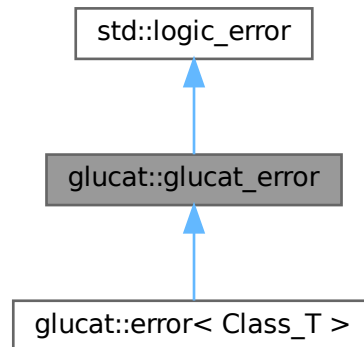
- [glucat/generation.h](#)
- [glucat/generation_imp.h](#)

6.15 glucat::glucat_error Class Reference

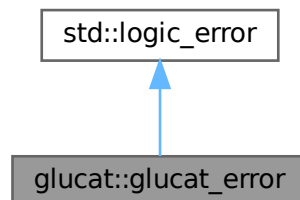
Abstract exception class.

```
#include <errors.h>
```

Inheritance diagram for glucat::glucat_error:



Collaboration diagram for glucat::glucat_error:



Public Member Functions

- `glucat_error` (const std::string &context, const std::string &msg)
- `~glucat_error` () noexcept override=default
- virtual auto `heading` () const noexcept -> const std::string=0
- virtual auto `classname` () const noexcept -> const std::string=0
- virtual void `print_error_msg` () const =0

Public Attributes

- std::string `name`

6.15.1 Detailed Description

Abstract exception class.

Definition at line 41 of file [errors.h](#).

6.15.2 Constructor & Destructor Documentation

6.15.2.1 `glucat_error()`

```
glucat::glucat_error::glucat_error (
    const std::string & context,
    const std::string & msg) [inline]
```

Definition at line 44 of file [errors.h](#).

6.15.2.2 `~glucat_error()`

```
glucat::glucat_error::~glucat_error () [override], [default], [noexcept]
```

6.15.3 Member Function Documentation

6.15.3.1 `classname()`

```
virtual auto glucat::glucat_error::classname () const -> const std::string [pure virtual],
[noexcept]
```

Implemented in [glucat::error< Class_T >](#).

6.15.3.2 `heading()`

```
virtual auto glucat::glucat_error::heading () const -> const std::string [pure virtual],
[noexcept]
```

Implemented in [glucat::error< Class_T >](#).

6.15.3.3 `print_error_msg()`

```
virtual void glucat::glucat_error::print_error_msg () const [pure virtual]
```

Implemented in [glucat::error< Class_T >](#).

6.15.4 Member Data Documentation

6.15.4.1 name

`std::string glucat::glucat_error::name`

Definition at line 51 of file [errors.h](#).

The documentation for this class was generated from the following file:

- [glucat/errors.h](#)

6.16 glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t Class Reference

Public Member Functions

- [hash_size_t](#) (size_t hash_size)
- auto [operator\(\)](#) () const -> size_t

Private Attributes

- size_t [n](#)

6.16.1 Detailed Description

```
template<typename Scalar_T, const index\_t LO, const index\_t HI, typename Tune_P>
class glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t
```

Definition at line 152 of file [framed_multi.h](#).

6.16.2 Constructor & Destructor Documentation

6.16.2.1 hash_size_t()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t::hash_size_t (
    size_t hash_size) [inline]
```

Definition at line 155 of file [framed_multi.h](#).

6.16.3 Member Function Documentation

6.16.3.1 operator()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t::operator() () const ->
size_t    [inline]
```

Definition at line 158 of file [framed_multi.h](#).

References [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t::n](#).

6.16.4 Member Data Documentation

6.16.4.1 n

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
size_t glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t::n [private]
```

Definition at line 161 of file [framed_multi.h](#).

Referenced by [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t::operator\(\)](#).

The documentation for this class was generated from the following file:

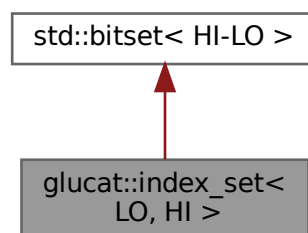
- [glucat/framed_multi.h](#)

6.17 [glucat::index_set](#)< LO, HI > Class Template Reference

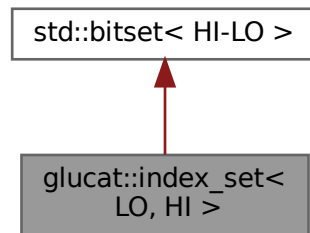
Index set class based on `std::bitset<>` in Gnu standard C++ library.

```
#include <index_set.h>
```

Inheritance diagram for [glucat::index_set](#)< LO, HI >:



Collaboration diagram for glucat::index_set< LO, HI >:



Classes

- class [reference](#)
Index set member reference.

Public Types

- using [index_set_t](#) = [index_set](#)
- using [index_pair_t](#) = `std::pair<index_t, index_t>`

Public Member Functions

- [index_set](#) ()=default
Default constructor creates an empty set.
- [index_set](#) (const [bitset_t](#) bst)
Constructor from [bitset_t](#).
- [index_set](#) (const [index_t](#) idx)
Constructor from [index](#).
- [index_set](#) (const [set_value_t](#) folded_val, const [index_set_t](#) frm, const bool prechecked=false)
Constructor from [set value](#) of an [index set](#) folded within the given frame.
- [index_set](#) (const [index_pair_t](#) &range, const bool prechecked=false)
Constructor from range of indices from [range.first](#) to [range.second](#).
- [index_set](#) (const std::string &str)
Constructor from [string](#).
- auto [operator==](#) (const [index_set_t](#) rhs) const -> bool
Equality.
- auto [operator!=](#) (const [index_set_t](#) rhs) const -> bool
Inequality.
- auto [operator~](#) () const -> [index_set_t](#)
Set complement: [not](#).
- auto [operator^](#) = (const [index_set_t](#) rhs) -> [index_set_t](#) &
Symmetric set difference: [exclusive or](#).
- auto [operator&=](#) (const [index_set_t](#) rhs) -> [index_set_t](#) &

- Set intersection: and.*

 - auto `operator|=` (const `index_set_t` rhs) -> `index_set_t` &
- Set union: or.*

 - auto `operator[]` (const `index_t` idx) const -> bool
- Subscripting: Test idx for membership: test value of bit idx.*

 - auto `test` (const `index_t` idx) const -> bool
- Test idx for membership: test value of bit idx.*

 - auto `set` () -> `index_set_t` &
- Include all indices except 0: set all bits except 0.*

 - auto `set` (const `index_t` idx) -> `index_set_t` &
- Include idx: Set bit at idx if idx != 0.*

 - auto `set` (const `index_t` idx, const int val) -> `index_set_t` &
- Set membership of idx to val if idx != 0: Set bit at idx to val if idx != 0.*

 - auto `reset` () -> `index_set_t` &
- Make set empty: Set all bits to 0.*

 - auto `reset` (const `index_t` idx) -> `index_set_t` &
- Exclude idx: Set bit at idx to 0.*

 - auto `flip` () -> `index_set_t` &
- Set complement, except 0: flip all bits, except 0.*

 - auto `flip` (const `index_t` idx) -> `index_set_t` &
- Complement membership of idx if idx != 0: flip bit at idx if idx != 0.*

 - auto `count` () const -> `index_t`
- Cardinality: Number of indices included in set.*

 - auto `count_neg` () const -> `index_t`
- Number of negative indices included in set.*

 - auto `count_pos` () const -> `index_t`
- Number of positive indices included in set.*

 - auto `min` () const -> `index_t`
- Minimum member.*

 - auto `max` () const -> `index_t`
- Maximum member.*

 - auto `operator<` (const `index_set_t` rhs) const -> bool
- Less than operator used for comparisons, map, etc.*

 - auto `is_contiguous` () const -> bool
- Determine if the index set is contiguous, ie. has no gaps.*

 - auto `fold` () const -> const `index_set_t`
- Fold this index set within itself as a frame.*

 - auto `fold` (const `index_set_t` frm, const bool prechecked=false) const -> const `index_set_t`
- Fold this index set within the given frame.*

 - auto `unfold` (const `index_set_t` frm, const bool prechecked=false) const -> const `index_set_t`
- Unfold this index set within the given frame.*

 - auto `value_of_fold` (const `index_set_t` frm) const -> `set_value_t`
- The set value of the fold of this index set within the given frame.*

 - auto `sign_of_mult` (const `index_set_t` ist) const -> int
- Sign of geometric product of two Clifford basis elements.*

 - auto `sign_of_square` () const -> int
- Sign of geometric square of a Clifford basis element.*

 - auto `hash_fn` () const -> `size_t`
- Hash function.*

 - auto `operator[]` (`index_t` idx) -> `reference`
- Subscripting: Element access.*

Static Public Member Functions

- static auto [classname](#) () -> const std::string

Static Public Attributes

- static const [index_t v_lo](#) = LO
- static const [index_t v_hi](#) = HI

Private Types

- using [bitset_t](#) = std::bitset<HI - LO>
- using [error_t](#) = [error](#)<[index_set](#)>

Private Member Functions

- [BOOST_STATIC_ASSERT](#) ((LO<=0) &&(0<=HI) &&(LO< HI) &&(-LO< _GLUCAT_BITS_PER_ULONG) &&(HI< _GLUCAT_BITS_PER_ULONG) &&(HI-LO<=_GLUCAT_BITS_PER_ULONG))
- auto [lex_less_than](#) (const [index_set_t](#) rhs) const -> bool
*Lexicographic ordering of two sets: *this < rhs.*

Friends

- class [reference](#)
- auto [operator^](#) (const [index_set_t](#) &lhs, const [index_set_t](#) &rhs) -> const [index_set_t](#)
- auto [operator&](#) (const [index_set_t](#) &lhs, const [index_set_t](#) &rhs) -> const [index_set_t](#)
- auto [operator|](#) (const [index_set_t](#) &lhs, const [index_set_t](#) &rhs) -> const [index_set_t](#)
- auto [compare](#) (const [index_set_t](#) &lhs, const [index_set_t](#) &rhs) -> int

6.17.1 Detailed Description

```
template<const index\_t LO, const index\_t HI>
class glucat::index_set< LO, HI >
```

Index set class based on std::bitset<> in Gnu standard C++ library.

Definition at line 73 of file [index_set.h](#).

6.17.2 Member Typedef Documentation

6.17.2.1 [bitset_t](#)

```
template<const index\_t LO, const index\_t HI>
using glucat::index_set< LO, HI >::bitset_t = std::bitset<HI - LO> [private]
```

Definition at line 81 of file [index_set.h](#).

6.17.2.2 error_t

```
template<const index_t LO, const index_t HI>
using glucat::index_set< LO, HI >::error_t = error<index_set> [private]
```

Definition at line 82 of file [index_set.h](#).

6.17.2.3 index_pair_t

```
template<const index_t LO, const index_t HI>
using glucat::index_set< LO, HI >::index_pair_t = std::pair<index_t, index_t>
```

Definition at line 85 of file [index_set.h](#).

6.17.2.4 index_set_t

```
template<const index_t LO, const index_t HI>
using glucat::index_set< LO, HI >::index_set_t = index_set
```

Definition at line 84 of file [index_set.h](#).

6.17.3 Constructor & Destructor Documentation

6.17.3.1 index_set() [1/6]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::index_set () [default]
```

Default constructor creates an empty set.

6.17.3.2 index_set() [2/6]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::index_set (
    const bitset_t bst)
```

Constructor from bitset_t.

Definition at line 61 of file [index_set_imp.h](#).

6.17.3.3 index_set() [3/6]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::index_set (
    const index_t idx)
```

Constructor from index.

Constructor from index value.

Definition at line 55 of file [index_set_imp.h](#).

6.17.3.4 index_set() [4/6]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::index_set (
    const set_value_t folded_val,
    const index_set_t frm,
    const bool prechecked = false)
```

Constructor from set value of an index set folded within the given frame.

Definition at line 68 of file [index_set_imp.h](#).

References [glucat::index_set< LO, HI >::count\(\)](#), [glucat::index_set< LO, HI >::fold\(\)](#), [glucat::index_set< LO, HI >::min\(\)](#), and [glucat::index_set< LO, HI >::unfold\(\)](#).

6.17.3.5 index_set() [5/6]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::index_set (
    const index_pair_t & range,
    const bool prechecked = false)
```

Constructor from range of indices from range.first to range.second.

Definition at line 82 of file [index_set_imp.h](#).

6.17.3.6 index_set() [6/6]

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::index_set (
    const std::string & str)
```

Constructor from string.

Definition at line 102 of file [index_set_imp.h](#).

6.17.4 Member Function Documentation**6.17.4.1 BOOST_STATIC_ASSERT()**

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::BOOST_STATIC_ASSERT (
    (LO<=0) && (0<=HI) && (LO< HI) && (-LO< _GLUCAT_BITS_PER_ULONGLONG) && (HI< _GLUCAT_←
    BITS_PER_ULONGLONG) && (HI-LO<=_GLUCAT_BITS_PER_ULONGLONG) ) [private]
```

6.17.4.2 classname()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::classname () -> const std::string [inline], [static]
```

Definition at line 49 of file [index_set_imp.h](#).

6.17.4.3 count()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::count () const -> index_t [inline]
```

Cardinality: Number of indices included in set.

Definition at line 344 of file [index_set_imp.h](#).

Referenced by [glucat::index_set< LO, HI >::count_neg\(\)](#), [glucat::index_set< LO, HI >::count_pos\(\)](#), [glucat::framed_multi< Scalar_T, index_set_t, LO, HI >::framed_multi\(\)](#), [glucat::index_set< LO, HI >::index_set\(\)](#), and [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#).

6.17.4.4 count_neg()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::count_neg () const -> index_t [inline]
```

Number of negative indices included in set.

Definition at line 364 of file [index_set_imp.h](#).

References [glucat::index_set< LO, HI >::count\(\)](#).

6.17.4.5 count_pos()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::count_pos () const -> index_t [inline]
```

Number of positive indices included in set.

Definition at line 376 of file [index_set_imp.h](#).

References [glucat::index_set< LO, HI >::count\(\)](#).

6.17.4.6 flip() [1/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::flip () -> index_set_t& [inline]
```

Set complement, except 0: flip all bits, except 0.

Definition at line 319 of file [index_set_imp.h](#).

6.17.4.7 flip() [2/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::flip (
    const index_t idx) -> index_set_t& [inline]
```

Complement membership of idx if idx != 0: flip bit at idx if idx != 0.

Definition at line 330 of file [index_set_imp.h](#).

6.17.4.8 fold() [1/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::fold () const -> const index_set_t [inline]
```

Fold this index set within itself as a frame.

Definition at line 747 of file [index_set_imp.h](#).

Referenced by [glucat::index_set< LO, HI >::index_set\(\)](#).

6.17.4.9 fold() [2/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::fold (
    const index_set_t frm,
    const bool prechecked = false) const -> const index_set_t
```

Fold this index set within the given frame.

Definition at line 755 of file [index_set_imp.h](#).

6.17.4.10 hash_fn()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::hash_fn () const -> size_t [inline]
```

Hash function.

Definition at line 950 of file [index_set_imp.h](#).

6.17.4.11 is_contiguous()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::is_contiguous () const -> bool [inline]
```

Determine if the index set is contiguous, ie. has no gaps.

Determine if the index set is contiguous, ie. has no gaps when 0 is included.

Definition at line 732 of file [index_set_imp.h](#).

6.17.4.12 lex_less_than()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::lex_less_than (
    const index_set_t rhs) const -> bool [inline], [private]
```

Lexicographic ordering of two sets: *this < rhs.

Definition at line 588 of file [index_set_imp.h](#).

6.17.4.13 max()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::max () const -> index_t
```

Maximum member.

Maximum member, or 0 if none.

Definition at line 550 of file [index_set_imp.h](#).

Referenced by [PyClical.index_set::__iter__\(\)](#), [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi\(\)](#), and [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#).

6.17.4.14 min()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::min () const -> index_t
```

Minimum member.

Minimum member, or 0 if none.

Definition at line 461 of file [index_set_imp.h](#).

Referenced by [PyClical.index_set::__iter__\(\)](#), [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi\(\)](#), [glucat::index_set< LO, HI >::index_set\(\)](#), and [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#).

6.17.4.15 operator"!="()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator!= (
    const index_set_t rhs) const -> bool [inline]
```

Inequality.

Definition at line 130 of file [index_set_imp.h](#).

6.17.4.16 operator&=()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator&= (
    const index_set_t rhs) -> index_set_t& [inline]
```

Set intersection: and.

Definition at line 174 of file [index_set_imp.h](#).

6.17.4.17 operator<()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator< (
    const index_set_t rhs) const -> bool [inline]
```

Less than operator used for comparisons, map, etc.

Definition at line 596 of file [index_set_imp.h](#).

6.17.4.18 operator==(())

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator==( (
    const index_set_t rhs) const -> bool [inline]
```

Equality.

Definition at line 119 of file [index_set_imp.h](#).

6.17.4.19 operator[]() [1/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator[] (
    const index_t idx) const -> bool [inline]
```

Subscripting: Test idx for membership: test value of bit idx.

Definition at line 232 of file [index_set_imp.h](#).

6.17.4.20 operator[]() [2/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator[] (
    index_t idx) -> reference [inline]
```

Subscripting: Element access.

Definition at line 224 of file [index_set_imp.h](#).

6.17.4.21 operator^=()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator^= (
    const index_set_t rhs) -> index_set_t& [inline]
```

Symmetric set difference: exclusive or.

Definition at line 149 of file [index_set_imp.h](#).

6.17.4.22 operator" |=()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator|= (
    const index_set_t rhs) -> index_set_t& [inline]
```

Set union: or.

Definition at line 199 of file [index_set_imp.h](#).

6.17.4.23 operator~()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::operator~ () const -> index_set_t [inline]
```

Set complement: not.

Definition at line 141 of file [index_set_imp.h](#).

6.17.4.24 reset() [1/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reset () -> index_set_t& [inline]
```

Make set empty: Set all bits to 0.

Definition at line 294 of file [index_set_imp.h](#).

6.17.4.25 reset() [2/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reset (
    const index_t idx) -> index_set_t& [inline]
```

Exclude idx: Set bit at idx to 0.

Definition at line 305 of file [index_set_imp.h](#).

6.17.4.26 set() [1/3]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::set () -> index_set_t& [inline]
```

Include all indices except 0: set all bits except 0.

Definition at line 255 of file [index_set_imp.h](#).

6.17.4.27 set() [2/3]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::set (
    const index_t idx) -> index_set_t& [inline]
```

Include idx: Set bit at idx if idx != 0.

Definition at line 266 of file [index_set_imp.h](#).

6.17.4.28 set() [3/3]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::set (
    const index_t idx,
    const int val) -> index_set_t& [inline]
```

Set membership of idx to val if idx != 0: Set bit at idx to val if idx != 0.

Definition at line 280 of file [index_set_imp.h](#).

6.17.4.29 sign_of_mult()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::sign_of_mult (
    const index_set_t ist) const -> int
```

Sign of geometric product of two Clifford basis elements.

Definition at line 880 of file [index_set_imp.h](#).

References [glucat::inverse_gray\(\)](#), and [glucat::inverse_reversed_gray\(\)](#).

6.17.4.30 sign_of_square()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::sign_of_square () const -> int [inline]
```

Sign of geometric square of a Clifford basis element.

Definition at line 930 of file [index_set_imp.h](#).

6.17.4.31 test()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::test (
    const index_t idx) const -> bool [inline]
```

Test idx for membership: test value of bit idx.

Definition at line 240 of file [index_set_imp.h](#).

6.17.4.32 `unfold()`

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::unfold (
    const index_set_t frm,
    const bool prechecked = false) const -> const index_set_t
```

Unfold this index set within the given frame.

Definition at line 794 of file `index_set_imp.h`.

Referenced by `glucat::index_set< LO, HI >::index_set()`.

6.17.4.33 `value_of_fold()`

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::value_of_fold (
    const index_set_t frm) const -> set_value_t [inline]
```

The set value of the fold of this index set within the given frame.

Definition at line 829 of file `index_set_imp.h`.

6.17.5 Friends And Related Symbol Documentation

6.17.5.1 `compare`

```
template<const index_t LO, const index_t HI>
auto compare (
    const index_set_t & lhs,
    const index_set_t & rhs) -> int [friend]
```

6.17.5.2 `operator&`

```
template<const index_t LO, const index_t HI>
auto operator& (
    const index_set_t & lhs,
    const index_set_t & rhs) -> const index_set_t [friend]
```

6.17.5.3 `operator^`

```
template<const index_t LO, const index_t HI>
auto operator^ (
    const index_set_t & lhs,
    const index_set_t & rhs) -> const index_set_t [friend]
```

6.17.5.4 operator" |

```
template<const index_t LO, const index_t HI>
auto operator| (
    const index_set_t & lhs,
    const index_set_t & rhs) -> const index_set_t [friend]
```

6.17.5.5 reference

```
template<const index_t LO, const index_t HI>
friend class reference [friend]
```

Definition at line 174 of file [index_set.h](#).

6.17.6 Member Data Documentation

6.17.6.1 v_hi

```
template<const index_t LO, const index_t HI>
const index_t glucat::index_set< LO, HI >::v_hi = HI [static]
```

Definition at line 88 of file [index_set.h](#).

6.17.6.2 v_lo

```
template<const index_t LO, const index_t HI>
const index_t glucat::index_set< LO, HI >::v_lo = LO [static]
```

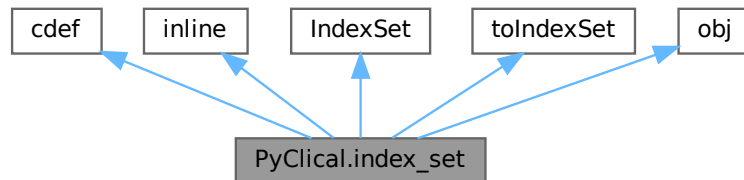
Definition at line 87 of file [index_set.h](#).

The documentation for this class was generated from the following files:

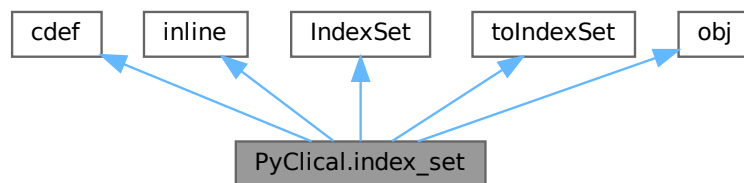
- [glucat/index_set.h](#)
- [glucat/index_set_imp.h](#)

6.18 PyClical.index_set Class Reference

Inheritance diagram for PyClical.index_set:



Collaboration diagram for PyClical.index_set:



Public Member Functions

- [__cinit__](#) (self, other=0)
- [__dealloc__](#) (self)
- [__richcmp__](#) (lhs, rhs, int, op)
- [__setitem__](#) (self, idx, val)
- [__getitem__](#) (self, idx)
- [__contains__](#) (self, idx)
- [__iter__](#) (self)
- [__invert__](#) (self)
- [__xor__](#) (lhs, rhs)
- [__ixor__](#) (self, rhs)
- [__and__](#) (lhs, rhs)
- [__iand__](#) (self, rhs)
- [__or__](#) (lhs, rhs)
- [__ior__](#) (self, rhs)
- [count](#) (self)
- [count_neg](#) (self)
- [count_pos](#) (self)
- [min](#) (self)
- [max](#) (self)

- [hash_fn](#) (self)
- [sign_of_mult](#) (self, *rhs*)
- [sign_of_square](#) (self)
- [__repr__](#) (self)
- [__str__](#) (self)

Public Attributes

- [instance](#) = new [IndexSet](#)((<[index_set](#)>other).unwrap())
- bool [instance](#) = True

6.18.1 Detailed Description

Return the C++ `IndexSet` instance wrapped by `index_set(obj)`.

Python class `index_set` wraps C++ class `IndexSet`.

Definition at line 46 of file [PyClical.pyx](#).

6.18.2 Member Function Documentation

6.18.2.1 `__and__()`

```
PyClical.index_set.__and__ (
    lhs,
    rhs)
```

Set intersection: `and`.

```
>>> print(index_set({1}) & index_set({2}))
{}
>>> print(index_set({1,2}) & index_set({2}))
{2}
```

Definition at line 271 of file [PyClical.pyx](#).

6.18.2.2 `__cinit__()`

```
PyClical.index_set.__cinit__ (
    self,
    other = 0)
```

Construct an object of type `index_set`.

```
>>> print(index_set(1))
{1}
>>> print(index_set({1,2}))
{1,2}
>>> print(index_set(index_set({1,2})))
{1,2}
>>> print(index_set({1,2}))
{1,2}
>>> print(index_set({1,2,1}))
{1,2}
>>> print(index_set("{1,2,1}"))
{1,2}
>>> print(index_set(""))
{}
```

Definition at line 74 of file [PyClical.pyx](#).

6.18.2.3 `__contains__()`

```
PyClical.index_set.__contains__ (
    self,
    idx)
```

Check that an `index_set` object contains the index `idx`: `idx in self`.

```
>>> 1 in index_set({1})
True
>>> 2 in index_set({1})
False
>>> -1 in index_set({2})
False
>>> 1 in index_set({2})
False
>>> 2 in index_set({2})
True
>>> 33 in index_set({2})
False
```

Definition at line 210 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

6.18.2.4 `__dealloc__()`

```
PyClical.index_set.__dealloc__ (
    self)
```

Clean up by deallocating the instance of C++ class `IndexSet`.

Definition at line 116 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

6.18.2.5 `__getitem__()`

```
PyClical.index_set.__getitem__ (
    self,
    idx)
```

Get the value of an `index_set` object at an index.

```
>>> index_set({1})[1]
True
>>> index_set({1})[2]
False
>>> index_set({2})[-1]
False
>>> index_set({2})[1]
False
>>> index_set({2})[2]
True
>>> index_set({2})[33]
False
```

Definition at line 191 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

6.18.2.6 `__iand__()`

```
PyClical.index_set.__iand__ (
    self,
    rhs)
```

Set intersection: and.

```
>>> x = index_set({1}); x &= index_set({2}); print(x)
{}
>>> x = index_set({1,2}); x &= index_set({2}); print(x)
{2}
```

Definition at line 282 of file [PyClical.pyx](#).

6.18.2.7 `__invert__()`

```
PyClical.index_set.__invert__ (
    self)
```

Set complement: not.

```
>>> print(~index_set({-16,-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8,9,10,11,12,13,14,
{-32,-31,-30,-29,-28,-27,-26,-25,-24,-23,-22,-21,-20,-19,-18,-17,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
```

Definition at line 240 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

6.18.2.8 `__ior__()`

```
PyClical.index_set.__ior__ (
    self,
    rhs)
```

Set union: or.

```
>>> x = index_set({1}); x |= index_set({2}); print(x)
{1,2}
>>> x = index_set({1,2}); x |= index_set({2}); print(x)
{1,2}
```

Definition at line 304 of file [PyClical.pyx](#).

6.18.2.9 `__iter__()`

```
PyClical.index_set.__iter__ (
    self)
```

Iterate over the indices of an index_set.

```
>>> for i in index_set({-3,4,7}):print(i, end=",")
-3,4,7,
```

Definition at line 229 of file [PyClical.pyx](#).

References [glucat::index_set< LO, HI >.max\(\)](#), [glucat::index_set< DEFAULT_LO, DEFAULT_HI >.max\(\)](#), [PyClical.index_set.max\(\)](#), [glucat::index_set< LO, HI >.min\(\)](#), [glucat::index_set< DEFAULT_LO, DEFAULT_HI >.min\(\)](#), and [PyClical.index_set.min\(\)](#).

6.18.2.10 `__ixor__()`

```
PyClical.index_set.__ixor__ (
    self,
    rhs)
```

Symmetric set difference: exclusive or.

```
>>> x = index_set({1}); x ^= index_set({2}); print(x)
{1,2}
>>> x = index_set({1,2}); x ^= index_set({2}); print(x)
{1}
```

Definition at line 260 of file [PyClical.pyx](#).

6.18.2.11 `__or__()`

```
PyClical.index_set.__or__ (
    lhs,
    rhs)
```

Set union: or.

```
>>> print(index_set({1}) | index_set({2}))
{1,2}
>>> print(index_set({1,2}) | index_set({2}))
{1,2}
```

Definition at line 293 of file [PyClical.pyx](#).

6.18.2.12 `__repr__()`

```
PyClical.index_set.__repr__ (
    self)
```

The “official” string representation of self.

```
>>> index_set({1,2}).__repr__()
'index_set({1,2})'
>>> repr(index_set({1,2}))
'index_set({1,2})'
```

Definition at line 384 of file [PyClical.pyx](#).

References [index_set_to_repr\(\)](#).

6.18.2.13 __richcmp__()

```
PyClical.index_set.__richcmp__ (
    lhs,
    rhs,
    int,
    op)
```

Compare two objects of class index_set.

```
>>> index_set(1) == index_set({1})
True
>>> index_set({1}) != index_set({1})
False
>>> index_set({1}) != index_set({2})
True
>>> index_set({1}) == index_set({2})
False
>>> index_set({1}) < index_set({2})
True
>>> index_set({1}) <= index_set({2})
True
>>> index_set({1}) > index_set({2})
False
>>> index_set({1}) >= index_set({2})
False
```

Definition at line 122 of file [PyClical.pyx](#).

6.18.2.14 __setitem__()

```
PyClical.index_set.__setitem__ (
    self,
    idx,
    val)
```

Set the value of an index_set object at index idx to value val.

```
>>> s=index_set({1}); s[2] = True; print(s)
{1,2}
>>> s=index_set({1,2}); s[1] = False; print(s)
{2}
```

Definition at line 179 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

6.18.2.15 __str__()

```
PyClical.index_set.__str__ (
    self)
```

The “informal” string representation of self.

```
>>> index_set({1,2}).__str__()
'{1,2}'
>>> str(index_set({1,2}))
'{1,2}'
```

Definition at line 395 of file [PyClical.pyx](#).

References [index_set_to_str\(\)](#).

6.18.2.16 `__xor__()`

```
PyClical.index_set.__xor__ (
    lhs,
    rhs)
```

Symmetric set difference: exclusive or.

```
>>> print(index_set({1}) ^ index_set({2}))
{1,2}
>>> print(index_set({1,2}) ^ index_set({2}))
{1}
```

Definition at line 249 of file [PyClical.pyx](#).

6.18.2.17 `count()`

```
PyClical.index_set.count (
    self)
```

Cardinality: Number of indices included in set.

```
>>> index_set({-1,1,2}).count()
3
```

Definition at line 315 of file [PyClical.pyx](#).

References [PyClical.index_set.count\(\)](#), [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

Referenced by [PyClical.index_set.count\(\)](#).

6.18.2.18 `count_neg()`

```
PyClical.index_set.count_neg (
    self)
```

Number of negative indices included in set.

```
>>> index_set({-1,1,2}).count_neg()
1
```

Definition at line 324 of file [PyClical.pyx](#).

References [PyClical.index_set.count_neg\(\)](#), [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

Referenced by [PyClical.index_set.count_neg\(\)](#).

6.18.2.19 count_pos()

```
PyClical.index_set.count_pos (  
    self)
```

Number of positive indices included in set.

```
>>> index_set({-1,1,2}).count_pos()  
2
```

Definition at line 333 of file [PyClical.pyx](#).

References [PyClical.index_set.count_pos\(\)](#), [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

Referenced by [PyClical.index_set.count_pos\(\)](#).

6.18.2.20 hash_fn()

```
PyClical.index_set.hash_fn (  
    self)
```

Hash function.

Definition at line 360 of file [PyClical.pyx](#).

References [PyClical.index_set.hash_fn\(\)](#), [PyClical.clifford.instance](#), and [PyClical.index_set.instance](#).

Referenced by [PyClical.index_set.hash_fn\(\)](#).

6.18.2.21 max()

```
PyClical.index_set.max (  
    self)
```

Maximum member.

```
>>> index_set({-1,1,2}).max()  
2
```

Definition at line 351 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.index_set.max\(\)](#).

Referenced by [PyClical.index_set.__iter__\(\)](#), and [PyClical.index_set.max\(\)](#).

6.18.2.22 min()

```
PyClical.index_set.min (  
    self)
```

Minimum member.

```
>>> index_set ({-1,1,2}).min()  
-1
```

Definition at line 342 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.index_set.min\(\)](#).

Referenced by [PyClical.index_set.__iter__\(\)](#), and [PyClical.index_set.min\(\)](#).

6.18.2.23 sign_of_mult()

```
PyClical.index_set.sign_of_mult (  
    self,  
    rhs)
```

Sign of geometric product of two Clifford basis elements.

```
>>> s = index_set ({1,2}); t=index_set ({-1}); s.sign_of_mult(t)  
1
```

Definition at line 366 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.index_set.sign_of_mult\(\)](#).

Referenced by [PyClical.index_set.sign_of_mult\(\)](#).

6.18.2.24 sign_of_square()

```
PyClical.index_set.sign_of_square (  
    self)
```

Sign of geometric square of a Clifford basis element.

```
>>> s = index_set ({1,2}); s.sign_of_square()  
-1
```

Definition at line 375 of file [PyClical.pyx](#).

References [PyClical.clifford.instance](#), [PyClical.index_set.instance](#), and [PyClical.index_set.sign_of_square\(\)](#).

Referenced by [PyClical.index_set.sign_of_square\(\)](#).

6.18.3 Member Data Documentation

6.18.3.1 instance [1/2]

```
PyClical.index_set.instance = new IndexSet((<index_set>other).unwrap())
```

Definition at line 95 of file [PyClical.pyx](#).

Referenced by [PyClical.clifford.__call__\(\)](#), [PyClical.index_set.__contains__\(\)](#), [PyClical.clifford.__dealloc__\(\)](#), [PyClical.index_set.__dealloc__\(\)](#), [PyClical.clifford.__getitem__\(\)](#), [PyClical.index_set.__getitem__\(\)](#), [PyClical.index_set.__invert__\(\)](#), [PyClical.clifford.__neg__\(\)](#), [PyClical.index_set.__setitem__\(\)](#), [PyClical.clifford.conj\(\)](#), [PyClical.index_set.count\(\)](#), [PyClical.index_set.count_neg\(\)](#), [PyClical.index_set.count_pos\(\)](#), [PyClical.clifford.even\(\)](#), [PyClical.clifford.frame\(\)](#), [PyClical.index_set.hash_fn\(\)](#), [PyClical.clifford.inv\(\)](#), [PyClical.clifford.involute\(\)](#), [PyClical.clifford.isinf\(\)](#), [PyClical.clifford.isnan\(\)](#), [PyClical.index_set.max\(\)](#), [PyClical.clifford.max_abs\(\)](#), [PyClical.index_set.min\(\)](#), [PyClical.clifford.norm\(\)](#), [PyClical.clifford.odd\(\)](#), [PyClical.clifford.outer_pow\(\)](#), [PyClical.clifford.pow\(\)](#), [PyClical.clifford.pure\(\)](#), [PyClical.clifford.quad\(\)](#), [PyClical.clifford.reverse\(\)](#), [PyClical.clifford.scalar\(\)](#), [PyClical.index_set.sign_of_mult\(\)](#), [PyClical.index_set.sign_of_square\(\)](#), [PyClical.clifford.truncated\(\)](#), and [PyClical.clifford.vector_part\(\)](#).

6.18.3.2 instance [2/2]

```
bool PyClical.index_set.instance = True
```

Definition at line 100 of file [PyClical.pyx](#).

Referenced by [PyClical.clifford.__call__\(\)](#), [PyClical.index_set.__contains__\(\)](#), [PyClical.clifford.__dealloc__\(\)](#), [PyClical.index_set.__dealloc__\(\)](#), [PyClical.clifford.__getitem__\(\)](#), [PyClical.index_set.__getitem__\(\)](#), [PyClical.index_set.__invert__\(\)](#), [PyClical.clifford.__neg__\(\)](#), [PyClical.index_set.__setitem__\(\)](#), [PyClical.clifford.conj\(\)](#), [PyClical.index_set.count\(\)](#), [PyClical.index_set.count_neg\(\)](#), [PyClical.index_set.count_pos\(\)](#), [PyClical.clifford.even\(\)](#), [PyClical.clifford.frame\(\)](#), [PyClical.index_set.hash_fn\(\)](#), [PyClical.clifford.inv\(\)](#), [PyClical.clifford.involute\(\)](#), [PyClical.clifford.isinf\(\)](#), [PyClical.clifford.isnan\(\)](#), [PyClical.index_set.max\(\)](#), [PyClical.clifford.max_abs\(\)](#), [PyClical.index_set.min\(\)](#), [PyClical.clifford.norm\(\)](#), [PyClical.clifford.odd\(\)](#), [PyClical.clifford.outer_pow\(\)](#), [PyClical.clifford.pow\(\)](#), [PyClical.clifford.pure\(\)](#), [PyClical.clifford.quad\(\)](#), [PyClical.clifford.reverse\(\)](#), [PyClical.clifford.scalar\(\)](#), [PyClical.index_set.sign_of_mult\(\)](#), [PyClical.index_set.sign_of_square\(\)](#), [PyClical.clifford.truncated\(\)](#), and [PyClical.clifford.vector_part\(\)](#).

The documentation for this class was generated from the following file:

- [pyclical/PyClical.pyx](#)

6.19 glucat::index_set_hash< LO, HI > Class Template Reference

```
#include <framed_multi.h>
```

Public Types

- using [index_set_t](#) = [index_set](#)<LO, HI>

Public Member Functions

- auto [operator\(\)](#) ([index_set_t](#) val) const -> [size_t](#)

6.19.1 Detailed Description

```
template<const index\_t LO, const index\_t HI>
class glucat::index_set_hash< LO, HI >
```

Definition at line 117 of file [framed_multi.h](#).

6.19.2 Member Typedef Documentation

6.19.2.1 [index_set_t](#)

```
template<const index\_t LO, const index\_t HI>
using glucat::index\_set\_hash< LO, HI >::index_set_t = index\_set<LO, HI>
```

Definition at line 120 of file [framed_multi.h](#).

6.19.3 Member Function Documentation

6.19.3.1 [operator\(\)](#)

```
template<const index\_t LO, const index\_t HI>
auto glucat::index\_set\_hash< LO, HI >::operator() (
    index\_set\_t val) const -> size\_t    [inline]
```

Definition at line 121 of file [framed_multi.h](#).

The documentation for this class was generated from the following file:

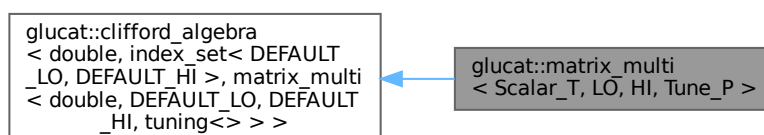
- [glucat/framed_multi.h](#)

6.20 [glucat::matrix_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) > Class Template Reference

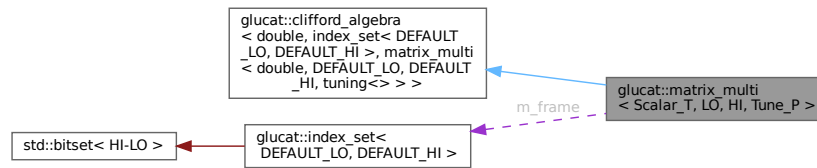
A [matrix_multi](#)<[Scalar_T](#),[LO](#),[HI](#),[Tune_P](#)> is a matrix approximation to a multivector.

```
#include <matrix\_multi.h>
```

Inheritance diagram for [glucat::matrix_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) >:



Collaboration diagram for glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >:



Public Types

- using `multivector_t` = `matrix_multi`
- using `matrix_multi_t` = `multivector_t`
- using `scalar_t` = `Scalar_T`
- using `tune_p` = `Tune_P`
- using `index_set_t` = `index_set<LO, HI>`
- using `term_t` = `std::pair<const index_set_t, Scalar_T>`
- using `vector_t` = `std::vector<Scalar_T>`
- using `error_t` = `error<multivector_t>`
- using `framed_multi_t` = `framed_multi<Scalar_T,LO,HI,Tune_P>`

Public Types inherited from

`glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, matrix_multi< double, DE`

- using `scalar_t`
- using `index_set_t`
- using `multivector_t`
- using `pair_t`
- using `vector_t`

Public Member Functions

- `~matrix_multi` () override=default
Destructor.
- `matrix_multi` ()
Default constructor.
- `template<typename Other_Scalar_T > matrix_multi` (const `matrix_multi`< Other_Scalar_T, LO, HI, Tune_P > &val)
Construct a multivector from a multivector with a different scalar type.
- `template<typename Other_Scalar_T > matrix_multi` (const `matrix_multi`< Other_Scalar_T, LO, HI, Tune_P > &val, const `index_set_t` frm, const bool prechecked=false)
Construct a multivector, within a given frame, from a given multivector.
- `matrix_multi` (const `multivector_t` &val, const `index_set_t` frm, const bool prechecked=false)
Construct a multivector, within a given frame, from a given multivector.
- `matrix_multi` (const `index_set_t` ist, const `Scalar_T` &crd=Scalar_T(1))
Construct a multivector from an index set and a scalar coordinate.
- `matrix_multi` (const `index_set_t` ist, const `Scalar_T` &crd, const `index_set_t` frm, const bool prechecked=false)

- Construct a multivector, within a given frame, from an index set and a scalar coordinate.*
- `matrix_multi` (const Scalar_T &scr, const `index_set_t` frm=`index_set_t`())
Construct a multivector from a scalar (within a frame, if given)
- `matrix_multi` (const int scr, const `index_set_t` frm=`index_set_t`())
Construct a multivector from an int (within a frame, if given)
- `matrix_multi` (const `vector_t` &vec, const `index_set_t` frm, const bool prechecked=false)
Construct a multivector, within a given frame, from a given vector.
- `matrix_multi` (const std::string &str)
Construct a multivector from a string: eg: "3+2{1,2}-6.1e-2{2,3}".
- `matrix_multi` (const std::string &str, const `index_set_t` frm, const bool prechecked=false)
Construct a multivector, within a given frame, from a string: eg: "3+2{1,2}-6.1e-2{2,3}".
- `matrix_multi` (const char *str)
Construct a multivector from a char: eg: "3+2{1,2}-6.1e-2{2,3}".*
- `matrix_multi` (const char *str, const `index_set_t` frm, const bool prechecked=false)
Construct a multivector, within a given frame, from a char: eg: "3+2{1,2}-6.1e-2{2,3}".*
- template<typename Other_Scalar_T >
`matrix_multi` (const `framed_multi`< Other_Scalar_T, LO, HI, Tune_P > &val)
Construct a multivector from a framed_multi_t.
- template<typename Other_Scalar_T >
`matrix_multi` (const `framed_multi`< Other_Scalar_T, LO, HI, Tune_P > &val, const `index_set_t` frm, const bool prechecked=false)
Construct a multivector, within a given frame, from a framed_multi_t.
- auto `fast_matrix_multi` (const `index_set_t` frm) const -> const `matrix_multi_t`
Use generalized FFT to construct a matrix_multi_t.
- template<typename Other_Scalar_T >
auto `fast_framed_multi` () const -> const `framed_multi`< Other_Scalar_T, LO, HI, Tune_P >
Use inverse generalized FFT to construct a framed_multi_t.
- `_GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS` auto `operator=` (const `multivector_t` &rhs) -> `multivector_t` &
Assignment operator.
- auto `operator+=` (const `term_t` &rhs) -> `multivector_t` &
Add a term, if non-zero.

Public Member Functions inherited from

`glucat::clifford_algebra`< double, `index_set`< DEFAULT_LO, DEFAULT_HI >, `matrix_multi`< double, DE

- virtual `~clifford_algebra` ()=default
- virtual auto `operator==` (const `multivector_t` &val) const -> bool=0
Test for equality of multivectors.
- virtual auto `operator==` (const double &scr) const -> bool=0
Test for equality of multivector and scalar.
- virtual auto `operator+=` (const `multivector_t` &rhs) -> `multivector_t` &=0
Geometric sum.
- virtual auto `operator+=` (const double &scr) -> `multivector_t` &=0
Geometric sum of multivector and scalar.
- virtual auto `operator-=` (const `multivector_t` &rhs) -> `multivector_t` &=0
Geometric difference.
- virtual auto `operator-=` (const double &scr) -> `multivector_t` &=0
Geometric difference of multivector and scalar.
- virtual auto `operator-` () const -> const `multivector_t`=0
Unary -.

- virtual auto `operator*=(const double &scr) -> multivector_t &=0`
Product of multivector and scalar.
- virtual auto `operator*=(const multivector_t &rhs) -> multivector_t &=0`
Geometric product.
- virtual auto `operator%=(const multivector_t &rhs) -> multivector_t &=0`
Contraction.
- virtual auto `operator&=(const multivector_t &rhs) -> multivector_t &=0`
Inner product.
- virtual auto `operator^=(const multivector_t &rhs) -> multivector_t &=0`
Outer product.
- virtual auto `operator/=(const double &scr) -> multivector_t &=0`
Quotient of multivector and scalar.
- virtual auto `operator/=(const multivector_t &rhs) -> multivector_t &=0`
Geometric quotient.
- virtual auto `operator|=(const multivector_t &rhs) -> multivector_t &=0`
Transformation via twisted adjoint action.
- virtual auto `inv () const -> const multivector_t=0`
Geometric multiplicative inverse.
- virtual auto `pow (int m) const -> const multivector_t=0`
**this to the m*
- virtual auto `outer_pow (int m) const -> const multivector_t=0`
Outer product power.
- virtual auto `frame () const -> const index_set_t=0`
Subalgebra generated by all generators of terms of given multivector.
- virtual auto `grade () const -> index_t=0`
Maximum of the grades of each term.
- virtual auto `operator[] (const index_set_t ist) const -> double=0`
Subscripting: map from index set to scalar coordinate.
- virtual auto `operator() (index_t grade) const -> const multivector_t=0`
Pure grade-vector part.
- virtual auto `scalar () const -> double=0`
Scalar part.
- virtual auto `pure () const -> const multivector_t=0`
Pure part.
- virtual auto `even () const -> const multivector_t=0`
Even part of multivector, sum of even grade terms.
- virtual auto `odd () const -> const multivector_t=0`
Odd part of multivector, sum of odd grade terms.
- virtual auto `vector_part () const -> const vector_t=0`
Vector part of multivector, as a vector_t with respect to frame()
- virtual auto `vector_part (const index_set_t frm, const bool prechecked) const -> const vector_t=0`
Vector part of multivector, as a vector_t with respect to frm.
- virtual auto `involute () const -> const multivector_t=0`
Main involution, each {i} is replaced by -{i} in each term, eg. {1} -> -{1}.
- virtual auto `reverse () const -> const multivector_t=0`
Reversion, eg. {1}{2} -> {2}*{1}.*
- virtual auto `conj () const -> const multivector_t=0`
Conjugation, reverse o involute == involute o reverse.
- virtual auto `quad () const -> double=0`
*Scalar_T quadratic form == (rev(x)*x)(0)*
- virtual auto `norm () const -> double=0`

- *Scalar_T norm == sum of norm of coordinates.*
- virtual auto `max_abs` () const -> double=0
Maximum of absolute values of components of multivector: multivector infinity norm.
- virtual auto `truncated` (const double &limit=`default_truncation`) const -> const `multivector_t`=0
Remove all terms with relative size smaller than limit.
- virtual auto `isinf` () const -> bool=0
Check if a multivector contains any infinite values.
- virtual auto `isnan` () const -> bool=0
Check if a multivector contains any IEEE NaN values.
- virtual void `write` (const std::string &msg="") const=0
Write formatted multivector to output.
- virtual void `write` (std::ofstream &ofile, const std::string &msg="") const=0
Write formatted multivector to file.

Static Public Member Functions

- static auto `classname` () -> const std::string
Class name used in messages.
- static auto `random` (const `index_set_t` frm, Scalar_T fill=Scalar_T(1)) -> const `matrix_multi_t`
Random multivector within a frame.

Static Public Member Functions inherited from

`glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, matrix_multi< double, DE`

- static auto `classname` () -> const std::string

Private Types

- using `orientation_t` = ublas::row_major
- using `basis_matrix_t` = ublas::compressed_matrix<int, `orientation_t`>
- using `matrix_t` = ublas::matrix<Scalar_T, `orientation_t`>
- using `matrix_index_t` = typename matrix_t::size_type

Private Member Functions

- template<typename Matrix_T >
`matrix_multi` (const Matrix_T &mtx, const `index_set_t` frm)
Construct a multivector within a given frame from a given matrix.
- `matrix_multi` (const `matrix_t` &mtx, const `index_set_t` frm)
Construct a multivector within a given frame from a given matrix.
- auto `basis_element` (const `index_set`< LO, HI > &ist) const -> const `basis_matrix_t`
Create a basis element matrix within the current frame.

Private Attributes

- `index_set_t m_frame`
Index set representing the frame for the subalgebra which contains the multivector.
- `matrix_t m_matrix`
Matrix value representing the multivector within the folded frame.

Friends

- template<typename Other_Scalar_T , const [index_t](#) Other_LO, const [index_t](#) Other_HI, typename Other_Tune_P >
class [framed_multi](#)
- template<typename Other_Scalar_T , const [index_t](#) Other_LO, const [index_t](#) Other_HI, typename Other_Tune_P >
class [matrix_multi](#)
- auto [operator*](#) (const [matrix_multi_t](#) &lhs, const [matrix_multi_t](#) &rhs) -> const [matrix_multi_t](#)
- auto [operator^](#) (const [matrix_multi_t](#) &lhs, const [matrix_multi_t](#) &rhs) -> const [matrix_multi_t](#)
- auto [operator&](#) (const [matrix_multi_t](#) &lhs, const [matrix_multi_t](#) &rhs) -> const [matrix_multi_t](#)
- auto [operator%](#) (const [matrix_multi_t](#) &lhs, const [matrix_multi_t](#) &rhs) -> const [matrix_multi_t](#)
- auto [star](#) (const [matrix_multi_t](#) &lhs, const [matrix_multi_t](#) &rhs) -> [Scalar_T](#)
- auto [operator/](#) (const [matrix_multi_t](#) &lhs, const [matrix_multi_t](#) &rhs) -> const [matrix_multi_t](#)
- auto [operator|](#) (const [matrix_multi_t](#) &lhs, const [matrix_multi_t](#) &rhs) -> const [matrix_multi_t](#)
- auto [operator>>](#) (std::istream &s, [multivector_t](#) &val) -> std::istream &
- auto [operator<<](#) (std::ostream &os, const [multivector_t](#) &val) -> std::ostream &
- template<typename Other_Scalar_T , const [index_t](#) Other_LO, const [index_t](#) Other_HI, typename Other_Tune_P >
auto [reframe](#) (const [matrix_multi](#)< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &lhs, const [matrix_multi](#)< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &rhs, [matrix_multi](#)< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &lhs_reframed, [matrix_multi](#)< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &rhs_reframed) -> const [index_set](#)< Other_LO, Other_HI >
- template<typename Other_Scalar_T , const [index_t](#) Other_LO, const [index_t](#) Other_HI, typename Other_Tune_P >
auto [matrix_sqrt](#) (const [matrix_multi](#)< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &val, const [matrix_multi](#)< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &i, const [index_t](#) level) -> const [matrix_multi](#)< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P >
- template<typename Other_Scalar_T , const [index_t](#) Other_LO, const [index_t](#) Other_HI, typename Other_Tune_P >
auto [matrix_log](#) (const [matrix_multi](#)< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &val, const [matrix_multi](#)< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > &i, const [index_t](#) level) -> const [matrix_multi](#)< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P >

Additional Inherited Members

Static Public Attributes inherited from

[glucat::clifford_algebra](#)< [double](#), [index_set](#)< [DEFAULT_LO](#), [DEFAULT_HI](#) >, [matrix_multi](#)< [double](#), [DE](#)

- static const [index_t](#) [v_lo](#)
- static const [index_t](#) [v_hi](#)
- static const double [default_truncation](#)

Default for truncation.

6.20.1 Detailed Description

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT\_LO, const index\_t HI = DEFAULT\_HI,
typename Tune_P = tuning<>>
class glucat::matrix_multi< Scalar\_T, LO, HI, Tune\_P >
```

A [matrix_multi](#)<[Scalar_T](#),[LO](#),[HI](#),[Tune_P](#)> is a matrix approximation to a multivector.

Definition at line [137](#) of file [matrix_multi.h](#).

6.20.2 Member Typedef Documentation

6.20.2.1 basis_matrix_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::basis_matrix_t = ublas::compressed_matrix<int, orientation_t> [private]
```

Definition at line 157 of file [matrix_multi.h](#).

6.20.2.2 error_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::error_t = error<multivector_t>
```

Definition at line 148 of file [matrix_multi.h](#).

6.20.2.3 framed_multi_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::framed_multi_t = framed_multi<Scalar_T, LO, HI, Tune_P>
```

Definition at line 149 of file [matrix_multi.h](#).

6.20.2.4 index_set_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::index_set_t = index_set<LO, HI>
```

Definition at line 145 of file [matrix_multi.h](#).

6.20.2.5 matrix_index_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_index_t = typename matrix_t::size_type [private]
```

Definition at line 159 of file [matrix_multi.h](#).

6.20.2.6 matrix_multi_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi_t = multivector_t
```

Definition at line 142 of file [matrix_multi.h](#).

6.20.2.7 matrix_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_t = ublas::matrix<Scalar_T, orientation_t> [private]
```

Definition at line 158 of file [matrix_multi.h](#).

6.20.2.8 multivector_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::multivector_t = matrix_multi
```

Definition at line 141 of file [matrix_multi.h](#).

6.20.2.9 orientation_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::orientation_t = ublas::row_major [private]
```

Definition at line 156 of file [matrix_multi.h](#).

6.20.2.10 scalar_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::scalar_t = Scalar_T
```

Definition at line 143 of file [matrix_multi.h](#).

6.20.2.11 term_t

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::term_t = std::pair<const index_set_t, Scalar_T>
```

Definition at line 146 of file [matrix_multi.h](#).

6.20.2.12 tune_p

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::tune_p = Tune_P
```

Definition at line 144 of file [matrix_multi.h](#).

6.20.2.13 vector_t

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
using glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::vector_t = std::vector<Scalar_T>
```

Definition at line 147 of file [matrix_multi.h](#).

6.20.3 Constructor & Destructor Documentation

6.20.3.1 ~matrix_multi()

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::~~matrix_multi () [override], [default]
```

Destructor.

6.20.3.2 matrix_multi() [1/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi ()
```

Default constructor.

Definition at line 106 of file [matrix_multi_imp.h](#).

References [glucat::matrix_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) >::m_matrix.

6.20.3.3 matrix_multi() [2/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const matrix\_multi< Other_Scalar_T, LO, HI, Tune_P > & val)
```

Construct a multivector from a multivector with a different scalar type.

Definition at line 115 of file [matrix_multi_imp.h](#).

References [glucat::matrix_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) >::m_matrix.

6.20.3.4 matrix_multi() [3/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const matrix\_multi< Other_Scalar_T, LO, HI, Tune_P > & val,
    const index\_set\_t frm,
    const bool prechecked = false)
```

Construct a multivector, within a given frame, from a given multivector.

Definition at line 134 of file [matrix_multi_imp.h](#).

References [glucat::folded_dim\(\)](#), [glucat::matrix_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) >::m_frame, [glucat::matrix_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) >::m_matrix, and [glucat::numeric_traits](#)< [Scalar_T](#) >::to_scalar_t().

6.20.3.5 matrix_multi() [4/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const multivector\_t & val,
    const index\_set\_t frm,
    const bool prechecked = false)
```

Construct a multivector, within a given frame, from a given multivector.

Definition at line 159 of file [matrix_multi_imp.h](#).

References [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_frame](#), and [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m](#)

6.20.3.6 matrix_multi() [5/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const index\_set\_t ist,
    const Scalar_T & crd = Scalar_T(1))
```

Construct a multivector from an index set and a scalar coordinate.

Definition at line 171 of file [matrix_multi_imp.h](#).

References [glucat::folded_dim\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_frame](#), and [glucat::matrix_multi< Scalar_T, L](#)

6.20.3.7 matrix_multi() [6/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const index\_set\_t ist,
    const Scalar_T & crd,
    const index\_set\_t frm,
    const bool prechecked = false)
```

Construct a multivector, within a given frame, from an index set and a scalar coordinate.

Definition at line 183 of file [matrix_multi_imp.h](#).

References [glucat::folded_dim\(\)](#), and [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix](#).

6.20.3.8 matrix_multi() [7/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const Scalar_T & scr,
    const index\_set\_t frm = index\_set\_t() )
```

Construct a multivector from a scalar (within a frame, if given)

Definition at line 197 of file [matrix_multi_imp.h](#).

References [glucat::folded_dim\(\)](#), and [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix](#).

6.20.3.9 matrix_multi() [8/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const int scr,
    const index\_set\_t frm = index\_set\_t() )
```

Construct a multivector from an int (within a frame, if given)

Definition at line 209 of file [matrix_multi_imp.h](#).

6.20.3.10 matrix_multi() [9/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const vector\_t & vec,
    const index\_set\_t frm,
    const bool prechecked = false)
```

Construct a multivector, within a given frame, from a given vector.

Definition at line 215 of file [matrix_multi_imp.h](#).

References [glucat::index_set< LO, HI >::count\(\)](#), [glucat::folded_dim\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix](#), [glucat::index_set< LO, HI >::max\(\)](#), and [glucat::index_set< LO, HI >::min\(\)](#).

6.20.3.11 matrix_multi() [10/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const std::string & str)
```

Construct a multivector from a string: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 240 of file [matrix_multi_imp.h](#).

6.20.3.12 matrix_multi() [11/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const std::string & str,
    const index\_set\_t frm,
    const bool prechecked = false)
```

Construct a multivector, within a given frame, from a string: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 246 of file [matrix_multi_imp.h](#).

6.20.3.13 matrix_multi() [12/17]

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const char * str) [inline]
```

Construct a multivector from a char*: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 196 of file [matrix_multi.h](#).

References [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi](#).

6.20.3.14 matrix_multi() [13/17]

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const char * str,
    const index_set_t frm,
    const bool prechecked = false) [inline]
```

Construct a multivector, within a given frame, from a char*: eg: "3+2{1,2}-6.1e-2{2,3}".

Definition at line 199 of file [matrix_multi.h](#).

References [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi](#).

6.20.3.15 matrix_multi() [14/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const framed_multi< Other_Scalar_T, LO, HI, Tune_P > & val)
```

Construct a multivector from a framed_multi_t.

Definition at line 253 of file [matrix_multi_imp.h](#).

References [glucat::folded_dim\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_frame](#), and [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::truncated\(\)](#).

6.20.3.16 matrix_multi() [15/17]

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T >
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const framed_multi< Other_Scalar_T, LO, HI, Tune_P > & val,
    const index_set_t frm,
    const bool prechecked = false)
```

Construct a multivector, within a given frame, from a framed_multi_t.

Definition at line 277 of file [matrix_multi_imp.h](#).

References [glucat::folded_dim\(\)](#), and [glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::truncated\(\)](#).

6.20.3.17 `matrix_multi()` [16/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
template<typename Matrix_T >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const Matrix_T & mtx,
    const index\_set\_t frm) [private]
```

Construct a multivector within a given frame from a given matrix.

Definition at line 303 of file [matrix_multi_imp.h](#).

References [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix](#).

6.20.3.18 `matrix_multi()` [17/17]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi (
    const matrix\_t & mtx,
    const index\_set\_t frm) [private]
```

Construct a multivector within a given frame from a given matrix.

Definition at line 322 of file [matrix_multi_imp.h](#).

6.20.4 Member Function Documentation

6.20.4.1 `basis_element()`

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::basis_element (
    const index\_set< LO, HI > & ist) const -> const basis\_matrix\_t [private]
```

Create a basis element matrix within the current frame.

Definition at line 1186 of file [matrix_multi_imp.h](#).

References [glucat::gen::generator_table< Matrix_T >::generator\(\)](#), [glucat::matrix::mono_prod\(\)](#), [glucat::offset_level\(\)](#), and [glucat::matrix::unit\(\)](#).

Referenced by [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi\(\)](#).

6.20.4.2 `classname()`

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::classname () -> const std::string
[static]
```

Class name used in messages.

Definition at line 78 of file [matrix_multi_imp.h](#).

6.20.4.3 fast_framed_multi()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
template<typename Other_Scalar_T >
auto glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_framed_multi () const -> const
framed_multi<Other_Scalar_T,LO,HI,Tune_P>
```

Use inverse generalized FFT to construct a framed_multi_t.

Definition at line 1109 of file [matrix_multi_imp.h](#).

References [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::centre_pm4_qp4\(\)](#), [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::centre_qp1_pm1\(\)](#), [glucat::fast\(\)](#), [glucat::gen::offset_to_super](#), [glucat::pos_mod\(\)](#), and [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::unfold\(\)](#).

6.20.4.4 fast_matrix_multi()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::fast_matrix_multi (
    const index_set_t frm) const -> const matrix_multi_t [inline]
```

Use generalized FFT to construct a matrix_multi_t.

Definition at line 1096 of file [matrix_multi_imp.h](#).

6.20.4.5 operator+=()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::operator+= (
    const term_t & rhs) -> multivector_t& [inline]
```

Add a term, if non-zero.

Geometric sum.

Geometric sum of multivector and scalar.

Definition at line 416 of file [matrix_multi_imp.h](#).

6.20.4.6 operator=()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
auto glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::operator= (
    const multivector_t & rhs) -> multivector_t&
```

Assignment operator.

Definition at line 330 of file [matrix_multi_imp.h](#).

6.20.4.7 random()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::matrix\_multi< Scalar_T, LO, HI, Tune_P >::random (
    const index\_set\_t frm,
    Scalar_T fill = Scalar_T(1)) -> const matrix\_multi\_t [static]
```

Random multivector within a frame.

Definition at line 926 of file [matrix_multi_imp.h](#).

References [glucat::framed_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) >::random().

6.20.5 Friends And Related Symbol Documentation

6.20.5.1 framed_multi

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
template<typename Other_Scalar_T , const index\_t Other_LO, const index\_t Other_HI, typename Other_Tune_P >
friend class framed\_multi [friend]
```

Definition at line 151 of file [matrix_multi.h](#).

6.20.5.2 matrix_log

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
template<typename Other_Scalar_T , const index\_t Other_LO, const index\_t Other_HI, typename Other_Tune_P >
auto matrix\_log (
    const matrix\_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & val,
    const matrix\_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & i,
    const index\_t level) -> const matrix\_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > [friend]
```

6.20.5.3 matrix_multi

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
template<typename Other_Scalar_T , const index\_t Other_LO, const index\_t Other_HI, typename Other_Tune_P >
friend class matrix\_multi [friend]
```

Definition at line 153 of file [matrix_multi.h](#).

Referenced by [glucat::matrix_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) >::matrix_multi(), and [glucat::matrix_multi](#)< [Scalar_T](#), [LO](#), [HI](#), [Tune_P](#) >::matrix_multi().

6.20.5.4 matrix_sqrt

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
template<typename Other_Scalar_T , const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
auto matrix_sqrt (
    const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & val,
    const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & i,
    const index_t level) -> const matrix_multi< Other_Scalar_T, Other_LO, Other_HI,
Other_Tune_P > [friend]
```

6.20.5.5 operator%

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
auto operator% (
    const matrix_multi_t & lhs,
    const matrix_multi_t & rhs) -> const matrix_multi_t [friend]
```

6.20.5.6 operator&

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
auto operator& (
    const matrix_multi_t & lhs,
    const matrix_multi_t & rhs) -> const matrix_multi_t [friend]
```

6.20.5.7 operator*

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
auto operator* (
    const matrix_multi_t & lhs,
    const matrix_multi_t & rhs) -> const matrix_multi_t [friend]
```

6.20.5.8 operator/

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
auto operator/ (
    const matrix_multi_t & lhs,
    const matrix_multi_t & rhs) -> const matrix_multi_t [friend]
```

6.20.5.9 operator<<

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
auto operator<< (
    std::ostream & os,
    const multivector_t & val) -> std::ostream & [friend]
```

6.20.5.10 operator>>

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
auto operator>> (
    std::istream & s,
    multivector_t & val) -> std::istream & [friend]
```

6.20.5.11 operator^

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
auto operator^ (
    const matrix_multi_t & lhs,
    const matrix_multi_t & rhs) -> const matrix_multi_t [friend]
```

6.20.5.12 operator"|

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
auto operator| (
    const matrix_multi_t & lhs,
    const matrix_multi_t & rhs) -> const matrix_multi_t [friend]
```

6.20.5.13 reframe

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
template<typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename Other_Tune_P >
auto reframe (
    const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & lhs,
    const matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & rhs,
    matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & lhs_reframed,
    matrix_multi< Other_Scalar_T, Other_LO, Other_HI, Other_Tune_P > & rhs_reframed)
-> const index_set< Other_LO, Other_HI > [friend]
```

6.20.5.14 star

```
template<typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
auto star (
    const matrix_multi_t & lhs,
    const matrix_multi_t & rhs) -> Scalar_T [friend]
```

6.20.6 Member Data Documentation

6.20.6.1 `m_frame`

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
index\_set\_t glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_frame [private]
```

Index set representing the frame for the subalgebra which contains the multivector.

Definition at line 278 of file [matrix_multi.h](#).

Referenced by [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), and [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#).

6.20.6.2 `m_matrix`

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
matrix\_t glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::m_matrix [private]
```

Matrix value representing the multivector within the folded frame.

Definition at line 280 of file [matrix_multi.h](#).

Referenced by [glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::framed_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), and [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#).

The documentation for this class was generated from the following files:

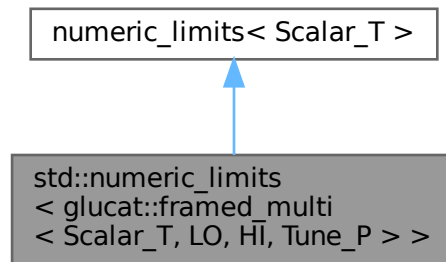
- [glucat/framed_multi.h](#)
- [glucat/matrix_multi.h](#)
- [glucat/matrix_multi_imp.h](#)

6.21 `std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > >` Struct Template Reference

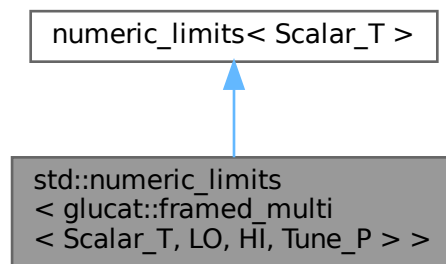
Numeric limits for `framed_multi` inherit limits for the corresponding scalar type.

```
#include <framed_multi.h>
```

Inheritance diagram for `std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > >`:



Collaboration diagram for `std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > >`:



6.21.1 Detailed Description

```
template<typename Scalar_T, const glucat::index_t LO, const glucat::index_t HI, typename Tune_P>
struct std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > >
```

Numeric limits for `framed_multi` inherit limits for the corresponding scalar type.

Definition at line 345 of file [framed_multi.h](#).

The documentation for this struct was generated from the following file:

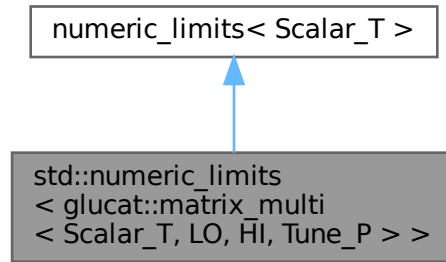
- [glucat/framed_multi.h](#)

6.22 std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > > Struct Template Reference

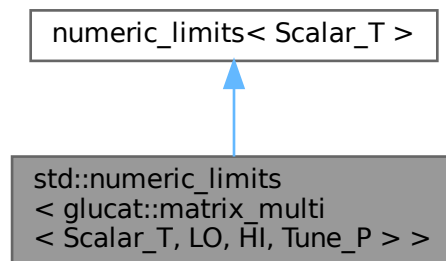
Numeric limits for matrix_multi inherit limits for the corresponding scalar type.

```
#include <matrix_multi.h>
```

Inheritance diagram for std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > >:



Collaboration diagram for std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > >:



6.22.1 Detailed Description

```
template<typename Scalar_T, const glucat::index_t LO, const glucat::index_t HI, typename Tune_P>
struct std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > >
```

Numeric limits for matrix_multi inherit limits for the corresponding scalar type.

Definition at line 296 of file [matrix_multi.h](#).

The documentation for this struct was generated from the following file:

- [glucat/matrix_multi.h](#)

6.23 `glucat::numeric_traits< Scalar_T >` Class Template Reference

Extra traits which extend numeric limits.

```
#include <scalar.h>
```

Classes

- struct [demoted](#)
Demoted type for long double.
- struct [promoted](#)
Extra traits which extend numeric limits.

Public Member Functions

- auto [pi](#) () -> long double
Pi for long double.
- auto [ln_2](#) () -> long double
log(2) for long double
- auto [to_scalar_t](#) (const Other_Scalar_T &val) -> float
Extra traits which extend numeric limits.
- auto [to_scalar_t](#) (const Other_Scalar_T &val) -> double
Cast to double.
- auto [to_scalar_t](#) (const dd_real &val) -> long double
Cast to long double.
- auto [to_scalar_t](#) (const qd_real &val) -> long double
Cast to long double.
- auto [to_scalar_t](#) (const long double &val) -> dd_real
Cast to dd_real.
- auto [to_scalar_t](#) (const qd_real &val) -> dd_real
Cast to dd_real.
- auto [to_scalar_t](#) (const long double &val) -> qd_real
Cast to qd_real.
- auto [to_scalar_t](#) (const dd_real &val) -> qd_real
Cast to qd_real.

Static Public Member Functions

- static auto [isInf](#) (const Scalar_T &val) -> bool
Smart isinf.
- static auto [isNaN](#) (const Scalar_T &val) -> bool
Smart isnan.
- static auto [isNaN_or_isInf](#) (const Scalar_T &val) -> bool
Smart isnan or isinf.
- static auto [NaN](#) () -> Scalar_T
Smart NaN.
- static auto [to_int](#) (const Scalar_T &val) -> int
Cast to int.
- static auto [to_double](#) (const Scalar_T &val) -> double

Cast to double.

- `template<typename Other_Scalar_T >`
`static auto to_scalar_t (const Other_Scalar_T &val) -> Scalar_T`

Cast to Scalar_T.

- `static auto fmod (const Scalar_T &lhs, const Scalar_T &rhs) -> Scalar_T`

Modulo function for scalar.

- `static auto conj (const Scalar_T &val) -> Scalar_T`

Complex conjugate of scalar.

- `static auto real (const Scalar_T &val) -> Scalar_T`

Real part of scalar.

- `static auto imag (const Scalar_T &val) -> Scalar_T`

Imaginary part of scalar.

- `static auto abs (const Scalar_T &val) -> Scalar_T`

Absolute value of scalar.

- `static auto pi () -> Scalar_T`

Pi.

- `static auto ln_2 () -> Scalar_T`

log(2)

- `static auto pow (const Scalar_T &val, int n) -> Scalar_T`

Integer power.

- `static auto sqrt (const Scalar_T &val) -> Scalar_T`

Square root of scalar.

- `static auto exp (const Scalar_T &val) -> Scalar_T`

Exponential.

- `static auto log (const Scalar_T &val) -> Scalar_T`

Logarithm of scalar.

- `static auto log2 (const Scalar_T &val) -> Scalar_T`

Log base 2.

- `static auto cos (const Scalar_T &val) -> Scalar_T`

Cosine of scalar.

- `static auto acos (const Scalar_T &val) -> Scalar_T`

Inverse cosine of scalar.

- `static auto cosh (const Scalar_T &val) -> Scalar_T`

Hyperbolic cosine of scalar.

- `static auto sin (const Scalar_T &val) -> Scalar_T`

Sine of scalar.

- `static auto asin (const Scalar_T &val) -> Scalar_T`

Inverse sine of scalar.

- `static auto sinh (const Scalar_T &val) -> Scalar_T`

Hyperbolic sine of scalar.

- `static auto tan (const Scalar_T &val) -> Scalar_T`

Tangent of scalar.

- `static auto atan (const Scalar_T &val) -> Scalar_T`

Inverse tangent of scalar.

- `static auto tanh (const Scalar_T &val) -> Scalar_T`

Hyperbolic tangent of scalar.

Static Private Member Functions

- static auto `isInf` (const Scalar_T &val, `bool_to_type`< false >) -> bool
Smart isinf specialised for Scalar_T without infinity.
- static auto `isInf` (const Scalar_T &val, `bool_to_type`< true >) -> bool
Smart isinf specialised for Scalar_T with infinity.
- static auto `isNaN` (const Scalar_T &val, `bool_to_type`< false >) -> bool
Smart isnan specialised for Scalar_T without quiet NaN.
- static auto `isNaN` (const Scalar_T &val, `bool_to_type`< true >) -> bool
Smart isnan specialised for Scalar_T with quiet NaN.

6.23.1 Detailed Description

```
template<typename Scalar_T>
class glucat::numeric_traits< Scalar_T >
```

Extra traits which extend numeric limits.

Definition at line 47 of file [scalar.h](#).

6.23.2 Member Function Documentation

6.23.2.1 `abs()`

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::abs (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Absolute value of scalar.

Definition at line 182 of file [scalar.h](#).

6.23.2.2 `acos()`

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::acos (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Inverse cosine of scalar.

Definition at line 245 of file [scalar.h](#).

6.23.2.3 `asin()`

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::asin (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Inverse sine of scalar.

Definition at line 266 of file [scalar.h](#).

6.23.2.4 atan()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::atan (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Inverse tangent of scalar.

Definition at line 287 of file [scalar.h](#).

6.23.2.5 conj()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::conj (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Complex conjugate of scalar.

Definition at line 161 of file [scalar.h](#).

6.23.2.6 cos()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::cos (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Cosine of scalar.

Definition at line 238 of file [scalar.h](#).

6.23.2.7 cosh()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::cosh (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Hyperbolic cosine of scalar.

Definition at line 252 of file [scalar.h](#).

6.23.2.8 exp()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::exp (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Exponential.

Definition at line 217 of file [scalar.h](#).

6.23.2.9 fmod()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::fmod (
    const Scalar_T & lhs,
    const Scalar_T & rhs) -> Scalar_T    [inline], [static]
```

Modulo function for scalar.

Definition at line 154 of file [scalar.h](#).

6.23.2.10 imag()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::imag (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Imaginary part of scalar.

Definition at line 175 of file [scalar.h](#).

6.23.2.11 isInf() [1/3]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isInf (
    const Scalar_T & val) -> bool    [inline], [static]
```

Smart isinf.

Definition at line 83 of file [scalar.h](#).

References [glucat::numeric_traits< Scalar_T >::isInf\(\)](#).

6.23.2.12 isInf() [2/3]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isInf (
    const Scalar_T & val,
    bool_to_type< false > ) -> bool    [inline], [static], [private]
```

Smart isinf specialised for Scalar_T without infinity.

Definition at line 54 of file [scalar.h](#).

Referenced by [glucat::numeric_traits< Scalar_T >::isInf\(\)](#), and [glucat::numeric_traits< Scalar_T >::isNaN_or_isInf\(\)](#).

6.23.2.13 `isInf()` [3/3]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isInf (
    const Scalar_T & val,
    bool_to_type< true > ) -> bool    [inline], [static], [private]
```

Smart `isinf` specialised for `Scalar_T` with infinity.

Definition at line 61 of file `scalar.h`.

References `_GLUCAT_ISINF`.

6.23.2.14 `isNaN()` [1/3]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isNaN (
    const Scalar_T & val) -> bool    [inline], [static]
```

Smart `isnan`.

Definition at line 93 of file `scalar.h`.

References `glucat::numeric_traits< Scalar_T >::isNaN()`.

6.23.2.15 `isNaN()` [2/3]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isNaN (
    const Scalar_T & val,
    bool_to_type< false > ) -> bool    [inline], [static], [private]
```

Smart `isnan` specialised for `Scalar_T` without quiet NaN.

Definition at line 68 of file `scalar.h`.

Referenced by `glucat::numeric_traits< Scalar_T >::isNaN()`, and `glucat::numeric_traits< Scalar_T >::isNaN_or_isInf()`.

6.23.2.16 `isNaN()` [3/3]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::isNaN (
    const Scalar_T & val,
    bool_to_type< true > ) -> bool    [inline], [static], [private]
```

Smart `isnan` specialised for `Scalar_T` with quiet NaN.

Definition at line 75 of file `scalar.h`.

References `_GLUCAT_ISNAN`.

6.23.2.17 isNaN_or_isInf()

```
template<typename Scalar_T >
static auto glucat::numeric\_traits< Scalar_T >::isNaN_or_isInf (
    const Scalar_T & val) -> bool    [inline], [static]
```

Smart isnan or isinf.

Definition at line 103 of file [scalar.h](#).

References [glucat::numeric_traits< Scalar_T >::isInf\(\)](#), and [glucat::numeric_traits< Scalar_T >::isNaN\(\)](#).

6.23.2.18 ln_2() [1/2]

```
auto glucat::numeric\_traits< longdouble >::ln_2 () -> long double    [inline]
```

log(2) for long double

Definition at line 59 of file [long_double.h](#).

References [glucat::l_ln2](#).

6.23.2.19 ln_2() [2/2]

```
template<typename Scalar_T >
static auto glucat::numeric\_traits< Scalar_T >::ln_2 () -> Scalar_T    [inline], [static]
```

log(2)

Definition at line 196 of file [scalar.h](#).

Referenced by [glucat::numeric_traits< Scalar_T >::log2\(\)](#).

6.23.2.20 log()

```
template<typename Scalar_T >
static auto glucat::numeric\_traits< Scalar_T >::log (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Logarithm of scalar.

Definition at line 224 of file [scalar.h](#).

Referenced by [glucat::numeric_traits< Scalar_T >::log2\(\)](#).

6.23.2.21 log2()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::log2 (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Log base 2.

Definition at line 231 of file [scalar.h](#).

References [glucat::numeric_traits< Scalar_T >::ln_2\(\)](#), and [glucat::numeric_traits< Scalar_T >::log\(\)](#).

Referenced by [glucat::log2\(\)](#).

6.23.2.22 NaN()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::NaN () -> Scalar_T    [inline], [static]
```

Smart NaN.

Definition at line 115 of file [scalar.h](#).

Referenced by [glucat::cr_sqrt\(\)](#), [glucat::db_sqrt\(\)](#), [glucat::matrix::norm_frob2\(\)](#), [glucat::operator*\(\)](#), and [glucat::matrix::trace\(\)](#).

6.23.2.23 pi() [1/2]

```
auto glucat::numeric_traits< longdouble >::pi () -> long double    [inline]
```

Pi for long double.

Definition at line 51 of file [long_double.h](#).

References [glucat::l_pi](#).

6.23.2.24 pi() [2/2]

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::pi () -> Scalar_T    [inline], [static]
```

Pi.

Definition at line 189 of file [scalar.h](#).

Referenced by [glucat::matrix::classify_eigenvalues\(\)](#).

6.23.2.25 pow()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::pow (
    const Scalar_T & val,
    int n) -> Scalar_T    [inline], [static]
```

Integer power.

Definition at line 203 of file [scalar.h](#).

Referenced by [glucat::error_squared_tol\(\)](#).

6.23.2.26 real()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::real (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Real part of scalar.

Definition at line 168 of file [scalar.h](#).

6.23.2.27 sin()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::sin (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Sine of scalar.

Definition at line 259 of file [scalar.h](#).

6.23.2.28 sinh()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::sinh (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Hyperbolic sine of scalar.

Definition at line 273 of file [scalar.h](#).

6.23.2.29 sqrt()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::sqrt (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Square root of scalar.

Definition at line 210 of file [scalar.h](#).

Referenced by [glucat::abs\(\)](#).

6.23.2.30 tan()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::tan (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Tangent of scalar.

Definition at line 280 of file [scalar.h](#).

6.23.2.31 tanh()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::tanh (
    const Scalar_T & val) -> Scalar_T    [inline], [static]
```

Hyperbolic tangent of scalar.

Definition at line 294 of file [scalar.h](#).

6.23.2.32 to_double()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::to_double (
    const Scalar_T & val) -> double    [inline], [static]
```

Cast to double.

Definition at line 133 of file [scalar.h](#).

Referenced by [glucat::operator<<\(\)](#), [glucat::numeric_traits< Scalar_T >::to_scalar_t\(\)](#), and [glucat::numeric_traits< Scalar_T >::to_](#)

6.23.2.33 to_int()

```
template<typename Scalar_T >
static auto glucat::numeric_traits< Scalar_T >::to_int (
    const Scalar_T & val) -> int    [inline], [static]
```

Cast to int.

Definition at line 126 of file [scalar.h](#).

6.23.2.34 to_scalar_t() [1/9]

```
auto glucat::numeric_traits< longdouble >::to_scalar_t (
    const dd_real & val) -> long double    [inline]
```

Cast to long double.

Definition at line 71 of file [scalar_imp.h](#).

6.23.2.35 to_scalar_t() [2/9]

```
auto glucat::numeric_traits< qd_real >::to_scalar_t (
    const dd_real & val) -> qd_real    [inline]
```

Cast to qd_real.

Definition at line 116 of file [scalar_imp.h](#).

6.23.2.36 to_scalar_t() [3/9]

```
auto glucat::numeric_traits< dd_real >::to_scalar_t (
    const long double & val) -> dd_real    [inline]
```

Cast to dd_real.

Definition at line 89 of file [scalar_imp.h](#).

6.23.2.37 to_scalar_t() [4/9]

```
auto glucat::numeric_traits< qd_real >::to_scalar_t (
    const long double & val) -> qd_real    [inline]
```

Cast to qd_real.

Definition at line 107 of file [scalar_imp.h](#).

6.23.2.38 to_scalar_t() [5/9]

```
auto glucat::numeric_traits< double >::to_scalar_t (
    const Other_Scalar_T & val) -> double    [inline]
```

Cast to double.

Definition at line 61 of file [scalar_imp.h](#).

References [glucat::numeric_traits< Scalar_T >::to_double\(\)](#).

6.23.2.39 to_scalar_t() [6/9]

```
auto glucat::numeric_traits< float >::to_scalar_t (
    const Other_Scalar_T & val) -> float    [inline]
```

Extra traits which extend numeric limits.

Cast to float

Definition at line 52 of file [scalar_imp.h](#).

References [glucat::numeric_traits< Scalar_T >::to_double\(\)](#).

6.23.2.40 `to_scalar_t()` [7/9]

```
template<typename Scalar_T >
template<typename Other_Scalar_T >
static auto glucat::numeric\_traits< Scalar_T >::to_scalar_t (
    const Other_Scalar_T & val) -> Scalar_T    [inline], [static]
```

Cast to `Scalar_T`.

Definition at line [141](#) of file [scalar.h](#).

Referenced by [glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >::matrix_multi\(\)](#), [glucat::matrix::nork_range\(\)](#), [glucat::to_demote\(\)](#), and [glucat::to_promote\(\)](#).

6.23.2.41 `to_scalar_t()` [8/9]

```
auto glucat::numeric\_traits< dd_real >::to_scalar_t (
    const qd_real & val) -> dd_real    [inline]
```

Cast to `dd_real`.

Definition at line [98](#) of file [scalar_imp.h](#).

6.23.2.42 `to_scalar_t()` [9/9]

```
auto glucat::numeric\_traits< longdouble >::to_scalar_t (
    const qd_real & val) -> long double    [inline]
```

Cast to long double.

Definition at line [80](#) of file [scalar_imp.h](#).

The documentation for this class was generated from the following file:

- [glucat/scalar.h](#)

6.24 `pade::pade_log_denom< Scalar_T >` Struct Template Reference

Coefficients of denominator polynomials of Pade approximations produced by `Pade1(log(1+x),x,n,n)`

```
#include <matrix_multi_imp.h>
```

Public Types

- using [array](#) = `std::array<Scalar_T, 14>`

Static Public Attributes

- static const [array](#) [denom](#)

6.24.1 Detailed Description

```
template<typename Scalar_T>
struct pade::pade_log_denom< Scalar_T >
```

Coefficients of denominator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)

Definition at line 1731 of file [matrix_multi_imp.h](#).

6.24.2 Member Typedef Documentation

6.24.2.1 array

```
template<typename Scalar_T >
using pade::pade_log_denom< Scalar_T >::array = std::array<Scalar_T, 14>
```

Definition at line 1733 of file [matrix_multi_imp.h](#).

6.24.3 Member Data Documentation

6.24.3.1 denom

```
template<typename Scalar_T >
const pade_log_denom< longdouble >::array pade::pade_log_denom< Scalar_T >::denom [static]
```

Initial value:

```
=
{
    1.0,                13.0/2.0,        468.0/25.0,        1573.0/50.0,
    1573.0/46.0,        11583.0/460.0,    10296.0/805.0,    2574.0/575.0,
    11583.0/10925.0,    143.0/874.0,        572.0/37145.0,    117.0/148580.0,
    13.0/742900.0,      1.0/10400600.0
}
```

Definition at line 1734 of file [matrix_multi_imp.h](#).

The documentation for this struct was generated from the following file:

- [glucat/matrix_multi_imp.h](#)

6.25 pade::pade_log_denom< dd_real > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using [array](#) = std::array<dd_real, 22>

Static Public Attributes

- static const [array](#) `denom`

6.25.1 Detailed Description

Definition at line 1820 of file [matrix_multi_imp.h](#).

6.25.2 Member Typedef Documentation

6.25.2.1 `array`

```
using pade::pade\_log\_denom< dd\_real >::array = std::array<dd_real, 22>
```

Definition at line 1822 of file [matrix_multi_imp.h](#).

6.25.3 Member Data Documentation

6.25.3.1 `denom`

```
const pade\_log\_denom< dd\_real >::array pade::pade\_log\_denom< dd\_real >::denom [static]
```

Initial value:

```
=
{
    dd_real("1"),
    dd_real("2100")/dd_real("41"),
    dd_real("341145")/dd_real("1066"),
    dd_real("11069856")/dd_real("19721"),
    dd_real("6918660")/dd_real("19721"),
    dd_real("1410864")/dd_real("16687"),
    dd_real("734825")/dd_real("94054"),
    dd_real("348840")/dd_real("1363783"),
    dd_real("6783")/dd_real("2727566"),
    dd_real("266")/dd_real("53187537"),
    dd_real("7")/dd_real("8155422340"),
    dd_real("21")/dd_real("2"),
    dd_real("12635")/dd_real("82"),
    dd_real("1037799")/dd_real("2132"),
    dd_real("9883800")/dd_real("19721"),
    dd_real("293930")/dd_real("1517"),
    dd_real("88179")/dd_real("3034"),
    dd_real("305235")/dd_real("188108"),
    dd_real("40698")/dd_real("1363783"),
    dd_real("9975")/dd_real("70916716"),
    dd_real("7")/dd_real("70916716"),
    dd_real("1")/dd_real("538257874440")
}
```

Definition at line 1823 of file [matrix_multi_imp.h](#).

The documentation for this struct was generated from the following file:

- [glucat/matrix_multi_imp.h](#)

6.26 `pade::pade_log_denom< float >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using [array](#) = std::array<float, 10>

Static Public Attributes

- static const [array](#) [denom](#)

6.26.1 Detailed Description

Definition at line 1758 of file [matrix_multi_imp.h](#).

6.26.2 Member Typedef Documentation

6.26.2.1 array

```
using pade::pade\_log\_denom< float >::array = std::array<float, 10>
```

Definition at line 1760 of file [matrix_multi_imp.h](#).

6.26.3 Member Data Documentation

6.26.3.1 denom

```
const pade\_log\_denom< float >::array pade::pade\_log\_denom< float >::denom [static]
```

Initial value:

```
=
{
    1.0,          9.0/2.0,      144.0/17.0,   147.0/17.0,
    441.0/85.0,   63.0/34.0,    84.0/221.0,   9.0/221.0,
    9.0/4862.0,   1.0/48620.0
}
```

Definition at line 1761 of file [matrix_multi_imp.h](#).

The documentation for this struct was generated from the following file:

- [glucat/matrix_multi_imp.h](#)

6.27 [pade::pade_log_denom](#)< long double > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using [array](#) = std::array<long double, 18>

Static Public Attributes

- static const [array](#) [denom](#)

6.27.1 Detailed Description

Definition at line 1785 of file [matrix_multi_imp.h](#).

6.27.2 Member Typedef Documentation

6.27.2.1 array

```
using pade::pade\_log\_denom< long double >::array = std::array<long double, 18>
```

Definition at line 1787 of file [matrix_multi_imp.h](#).

6.27.3 Member Data Documentation

6.27.3.1 denom

```
const array pade::pade\_log\_denom< long double >::denom [static]
```

Definition at line 1788 of file [matrix_multi_imp.h](#).

The documentation for this struct was generated from the following file:

- [glucat/matrix_multi_imp.h](#)

6.28 pade::pade_log_denom< qd_real > Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using [array](#) = std::array<qd_real, 34>

Static Public Attributes

- static const [array](#) [denom](#)

6.28.1 Detailed Description

Definition at line 1867 of file [matrix_multi_imp.h](#).

6.28.2 Member Typedef Documentation

6.28.2.1 array

using `pade::pade_log_denom< qd_real >::array` = `std::array<qd_real, 34>`

Definition at line 1869 of file `matrix_multi_imp.h`.

6.28.3 Member Data Documentation

6.28.3.1 denom

const `pade_log_denom< qd_real >::array pade::pade_log_denom< qd_real >::denom` [static]

Initial value:

```
=
{
    qd_real("1"),
    qd_real("33")/qd_real("2"),
    qd_real("8448")/qd_real("65"),
    qd_real("42284")/qd_real("65"),
    qd_real("211420")/qd_real("91"),
    qd_real("573562")/qd_real("91"),
    qd_real("32119472")/qd_real("2379"),
    qd_real("92917044")/qd_real("3965"),
    qd_real("603960786")/qd_real("17995"),
    qd_real("144626625")/qd_real("3599"),
    qd_real("2776831200")/qd_real("68381"),
    qd_real("16692542100")/qd_real("478667"),
    qd_real("12241197540")/qd_real("478667"),
    qd_real("1098569010")/qd_real("68381"),
    qd_real("31387686000")/qd_real("3624193"),
    qd_real("9939433900")/qd_real("2479711"),
    qd_real("67091178825")/qd_real("42155087"),
    qd_real("2683647153")/qd_real("4959422"),
    qd_real("19083713088")/qd_real("121505839"),
    qd_real("4708152900")/qd_real("121505839"),
    qd_real("941630580")/qd_real("116546417"),
    qd_real("88704330")/qd_real("62755763"),
    qd_real("12902448")/qd_real("62755763"),
    qd_real("1542684")/qd_real("62755763"),
    qd_real("6427850")/qd_real("2698497809"),
    qd_real("3471039")/qd_real("18889484663"),
    qd_real("8544096")/qd_real("774468871183"),
    qd_real("39556")/qd_real("79027435835"),
    qd_real("118668")/qd_real("7191496660985"),
    qd_real("10230")/qd_real("27327687311743"),
    qd_real("5456")/qd_real("1011124430534491"),
    qd_real("44")/qd_real("1011124430534491"),
    qd_real("11")/qd_real("70778710137414370"),
    qd_real("1")/qd_real("7219428434016265740")
}
```

Definition at line 1870 of file `matrix_multi_imp.h`.

The documentation for this struct was generated from the following file:

- `glucat/matrix_multi_imp.h`

6.29 pade::pade_log_numer< Scalar_T > Struct Template Reference

Coefficients of numerator polynomials of Pade approximations produced by `Pade1(log(1+x),x,n,n)`

```
#include <matrix_multi_imp.h>
```

Public Types

- using [array](#) = std::array<Scalar_T, 14>

Static Public Attributes

- static const [array](#) [numer](#)

6.29.1 Detailed Description

```
template<typename Scalar_T>
struct pade::pade_log_numer< Scalar_T >
```

Coefficients of numerator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)

Definition at line 1714 of file [matrix_multi_imp.h](#).

6.29.2 Member Typedef Documentation

6.29.2.1 array

```
template<typename Scalar_T >
using pade::pade\_log\_numer< Scalar\_T >::array = std::array<Scalar_T, 14>
```

Definition at line 1716 of file [matrix_multi_imp.h](#).

6.29.3 Member Data Documentation

6.29.3.1 numer

```
template<typename Scalar_T >
const pade\_log\_numer< longdouble >::array pade::pade\_log\_numer< Scalar\_T >::numer [static]
```

Initial value:

```
=
{
    0.0,          1.0,          6.0,          4741.0/300.0,
    1441.0/60.0,   107091.0/4600.0,   8638.0/575.0,   263111.0/40250.0,
    153081.0/80500.0,   395243.0/1101240.0,   28549.0/688275.0,   605453.0/228813200.0,
    785633.0/10296594000.0,   1145993.0/1873980108000.0
}
```

Definition at line 1717 of file [matrix_multi_imp.h](#).

The documentation for this struct was generated from the following file:

- [glucat/matrix_multi_imp.h](#)

6.30 `pade::pade_log_numer< dd_real >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using `array` = `std::array<dd_real, 22>`

Static Public Attributes

- static const `array` `numer`

6.30.1 Detailed Description

Definition at line 1800 of file `matrix_multi_imp.h`.

6.30.2 Member Typedef Documentation

6.30.2.1 `array`

```
using pade::pade_log_numer< dd_real >::array = std::array<dd_real, 22>
```

Definition at line 1802 of file `matrix_multi_imp.h`.

6.30.3 Member Data Documentation

6.30.3.1 `numer`

```
const pade_log_numer< dd_real >::array pade::pade_log_numer< dd_real >::numer [static]
```

Initial value:

```
=
{
    dd_real("0"),
    dd_real("10"),
    dd_real("21603")/dd_real("164"),
    dd_real("978724")/dd_real("2665"),
    dd_real("12874933")/dd_real("39442"),
    dd_real("2406734")/dd_real("22755"),
    dd_real("30653165")/dd_real("2402928"),
    dd_real("25346331")/dd_real("47074027"),
    dd_real("105689791")/dd_real("15601677520"),
    dd_real("969715")/dd_real("53502994116"),
    dd_real("118999")/dd_real("26204577562592"),
    dd_real("1"),
    dd_real("22781")/dd_real("492"),
    dd_real("5492649")/dd_real("21320"),
    dd_real("4191605")/dd_real("10619"),
    dd_real("11473457")/dd_real("54612"),
    dd_real("166770367")/dd_real("4004880"),
    dd_real("647746389")/dd_real("215195552"),
    dd_real("278270613")/dd_real("3900419380"),
    dd_real("606046475")/dd_real("1379188292768"),
    dd_real("11098301")/dd_real("26204577562592"),
    dd_real("18858053")/dd_real("1392249205900512960")
}
```

Definition at line 1803 of file `matrix_multi_imp.h`.

The documentation for this struct was generated from the following file:

- `glucat/matrix_multi_imp.h`

6.31 `pade::pade_log_numer< float >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using `array` = `std::array<float, 10>`

Static Public Attributes

- static const `array numer`

6.31.1 Detailed Description

Definition at line 1746 of file `matrix_multi_imp.h`.

6.31.2 Member Typedef Documentation

6.31.2.1 `array`

```
using pade::pade_log_numer< float >::array = std::array<float, 10>
```

Definition at line 1748 of file `matrix_multi_imp.h`.

6.31.3 Member Data Documentation

6.31.3.1 `numer`

```
const pade_log_numer< float >::array pade::pade_log_numer< float >::numer [static]
```

Initial value:

```
=
{
    0.0,          1.0,          4.0,          1337.0/204.0,
    385.0/68.0,   1879.0/680.0,   193.0/255.0,   197.0/1820.0,
    419.0/61880.0, 7129.0/61261200.0
}
```

Definition at line 1749 of file `matrix_multi_imp.h`.

The documentation for this struct was generated from the following file:

- `glucat/matrix_multi_imp.h`

6.32 `pade::pade_log_numer< long double >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using `array` = `std::array<long double, 18>`

Static Public Attributes

- static const `array numer`

6.32.1 Detailed Description

Definition at line 1771 of file `matrix_multi_imp.h`.

6.32.2 Member Typedef Documentation

6.32.2.1 `array`

```
using pade::pade_log_number< long double >::array = std::array<long double, 18>
```

Definition at line 1773 of file `matrix_multi_imp.h`.

6.32.3 Member Data Documentation

6.32.3.1 `numer`

```
const array pade::pade_log_number< long double >::numer [static]
```

Definition at line 1774 of file `matrix_multi_imp.h`.

The documentation for this struct was generated from the following file:

- `glucat/matrix_multi_imp.h`

6.33 `pade::pade_log_number< qd_real >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using `array` = `std::array<qd_real, 34>`

Static Public Attributes

- static const `array numer`

6.33.1 Detailed Description

Definition at line 1841 of file [matrix_multi_imp.h](#).

6.33.2 Member Typedef Documentation

6.33.2.1 array

```
using pade::pade_log_number< qd_real >::array = std::array<qd_real, 34>
```

Definition at line 1843 of file [matrix_multi_imp.h](#).

6.33.3 Member Data Documentation

6.33.3.1 numer

```
const pade_log_number< qd_real >::array pade::pade_log_number< qd_real >::numer [static]
```

Initial value:

```
=
{
    qd_real("0"),
    qd_real("16"),
    qd_real("95201")/qd_real("780"),
    qd_real("30721")/qd_real("52"),
    qd_real("7416257")/qd_real("3640"),
    qd_real("1039099")/qd_real("195"),
    qd_real("6097772319")/qd_real("555100"),
    qd_real("1564058073")/qd_real("85400"),
    qd_real("30404640205")/qd_real("1209264"),
    qd_real("725351278")/qd_real("25193"),
    qd_real("4092322670789")/qd_real("147429436"),
    qd_real("4559713849589")/qd_real("201040140"),
    qd_real("5049361751189")/qd_real("320023080"),
    qd_real("74979677195")/qd_real("8000577"),
    qd_real("16569850691873")/qd_real("3481514244"),
    qd_real("1065906022369")/qd_real("515779888"),
    qd_real("335956770855841")/qd_real("438412904800"),
    qd_real("1462444287585964")/qd_real("6041877844275"),
    qd_real("397242326339851")/qd_real("6122436215532"),
    qd_real("64211291334131")/qd_real("4373168725380"),
    qd_real("142322343550859")/qd_real("51080680851480"),
    qd_real("154355972958659")/qd_real("351179680853925"),
    qd_real("167483568676259")/qd_real("2937139148960100"),
    qd_real("4230788929433")/qd_real("704913395750424"),
    qd_real("197968763176019")/qd_real("392923948371995600"),
    qd_real("10537522306718")/qd_real("319250708052246425"),
    qd_real("236648286272519")/qd_real("144249197475035425500"),
    qd_real("260715545088119")/qd_real("4375558990076074573500"),
    qd_real("289596255666839")/qd_real("192874640282553367199880"),
    qd_real("8802625510547")/qd_real("361639950529787563499775"),
    qd_real("373831661521439")/qd_real("1659204093030665341336967700"),
    qd_real("446033437968239")/qd_real("464577146048586295574350956000"),
    qd_real("53676090078349")/qd_real("47386868896955802148583797512000")
}
```

Definition at line 1844 of file [matrix_multi_imp.h](#).

The documentation for this struct was generated from the following file:

- [glucat/matrix_multi_imp.h](#)

6.34 pade::pade_sqrt_denom< Scalar_T > Struct Template Reference

Coefficients of denominator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)

```
#include <matrix_multi_imp.h>
```

Public Types

- using [array](#) = std::array<Scalar_T, 14>

Static Public Attributes

- static const [array](#) [denom](#)

6.34.1 Detailed Description

```
template<typename Scalar_T>
struct pade::pade_sqrt_denom< Scalar_T >
```

Coefficients of denominator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)

Definition at line [1401](#) of file [matrix_multi_imp.h](#).

6.34.2 Member Typedef Documentation

6.34.2.1 array

```
template<typename Scalar_T >
using pade::pade\_sqrt\_denom< Scalar_T >::array = std::array<Scalar_T, 14>
```

Definition at line [1403](#) of file [matrix_multi_imp.h](#).

6.34.3 Member Data Documentation

6.34.3.1 denom

```
template<typename Scalar_T >
const pade\_sqrt\_denom< longdouble >::array pade::pade\_sqrt\_denom< Scalar_T >::denom [static]
```

Initial value:

```
=
{
    1.0,          25.0/4.0,          69.0/4.0,          1771.0/64.0,
    7315.0/256.0,  20349.0/1024.0,   4845.0/512.0,   12597.0/4096.0,
    21879.0/32768.0, 12155.0/131072.0, 1001.0/131072.0, 1365.0/4194304.0,
    91.0/16777216.0, 1.0/67108864.0
}
```

Definition at line [1404](#) of file [matrix_multi_imp.h](#).

The documentation for this struct was generated from the following file:

- [glucat/matrix_multi_imp.h](#)

6.35 `pade::pade_sqrt_denom< dd_real >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using `array` = `std::array<dd_real, 22>`

Static Public Attributes

- static const `array` `denom`

6.35.1 Detailed Description

Definition at line 1491 of file `matrix_multi_imp.h`.

6.35.2 Member Typedef Documentation

6.35.2.1 `array`

```
using pade::pade_sqrt_denom< dd_real >::array = std::array<dd_real, 22>
```

Definition at line 1493 of file `matrix_multi_imp.h`.

6.35.3 Member Data Documentation

6.35.3.1 `denom`

```
const pade_sqrt_denom< dd_real >::array pade::pade_sqrt_denom< dd_real >::denom [static]
```

Initial value:

```
=
{
    dd_real("1"),
    dd_real("195")/dd_real("4"),
    dd_real("73815")/dd_real("256"),
    dd_real("121737")/dd_real("256"),
    dd_real("4539051")/dd_real("16384"),
    dd_real("4032015")/dd_real("65536"),
    dd_real("86493225")/dd_real("16777216"),
    dd_real("5014575")/dd_real("33554432"),
    dd_real("5311735")/dd_real("4294967296"),
    dd_real("33649")/dd_real("17179869184"),
    dd_real("231")/dd_real("1099511627776"),
    dd_real("41")/dd_real("4"),
    dd_real("9139")/dd_real("64"),
    dd_real("435897")/dd_real("1024"),
    dd_real("840565")/dd_real("2048"),
    dd_real("9641775")/dd_real("65536"),
    dd_real("84672315")/dd_real("4194304"),
    dd_real("67863915")/dd_real("67108864"),
    dd_real("4345965")/dd_real("268435456"),
    dd_real("1081575")/dd_real("17179869184"),
    dd_real("8855")/dd_real("274877906944"),
    dd_real("1")/dd_real("4398046511104")
}
```

Definition at line 1494 of file `matrix_multi_imp.h`.

The documentation for this struct was generated from the following file:

- `glucat/matrix_multi_imp.h`

6.36 `pade::pade_sqrt_denom< float >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using `array` = `std::array<float, 10>`

Static Public Attributes

- static const `array` `denom`

6.36.1 Detailed Description

Definition at line 1428 of file `matrix_multi_imp.h`.

6.36.2 Member Typedef Documentation

6.36.2.1 `array`

```
using pade::pade_sqrt_denom< float >::array = std::array<float, 10>
```

Definition at line 1430 of file `matrix_multi_imp.h`.

6.36.3 Member Data Documentation

6.36.3.1 `denom`

```
const pade_sqrt_denom< float >::array pade::pade_sqrt_denom< float >::denom [static]
```

Initial value:

```
=
{
    1.0,          17.0/4.0,      15.0/2.0,      455.0/64.0,
    1001.0/256.0,  1287.0/1024.0,  231.0/1024.0,  165.0/8192.0,
    45.0/65536,   1.0/262144.0
}
```

Definition at line 1431 of file `matrix_multi_imp.h`.

The documentation for this struct was generated from the following file:

- `glucat/matrix_multi_imp.h`

6.37 `pade::pade_sqrt_denom< long double >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using `array` = `std::array<long double, 18>`

Static Public Attributes

- static const `array` `denom`

6.37.1 Detailed Description

Definition at line 1455 of file `matrix_multi_imp.h`.

6.37.2 Member Typedef Documentation

6.37.2.1 `array`

```
using pade::pade_sqrt_denom< long double >::array = std::array<long double, 18>
```

Definition at line 1457 of file `matrix_multi_imp.h`.

6.37.3 Member Data Documentation

6.37.3.1 `denom`

```
const array pade::pade_sqrt_denom< long double >::denom [static]
```

Definition at line 1458 of file `matrix_multi_imp.h`.

The documentation for this struct was generated from the following file:

- `glucat/matrix_multi_imp.h`

6.38 `pade::pade_sqrt_denom< qd_real >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using `array` = `std::array<qd_real, 34>`

Static Public Attributes

- static const `array` `denom`

6.38.1 Detailed Description

Definition at line 1538 of file [matrix_multi_imp.h](#).

6.38.2 Member Typedef Documentation

6.38.2.1 array

```
using pade::pade_sqrt_denom< qd_real >::array = std::array<qd_real, 34>
```

Definition at line 1540 of file [matrix_multi_imp.h](#).

6.38.3 Member Data Documentation

6.38.3.1 denom

```
const pade_sqrt_denom< qd_real >::array pade::pade_sqrt_denom< qd_real >::denom [static]
```

Initial value:

```
=
{
    qd_real("1"),
    qd_real("126"),
    qd_real("557845")/qd_real("256"),
    qd_real("12515965")/qd_real("1024"),
    qd_real("1916797311")/qd_real("65536"),
    qd_real("4450881435")/qd_real("131072"),
    qd_real("171503444385")/qd_real("8388608"),
    qd_real("221120793075")/qd_real("33554432"),
    qd_real("4923689695575")/qd_real("4294967296"),
    qd_real("456864812569")/qd_real("4294967296"),
    qd_real("3486599885395")/qd_real("137438953472"),
    qd_real("2804116503573")/qd_real("549755813888"),
    qd_real("1886827875075")/qd_real("2199023255552"),
    qd_real("263012370465")/qd_real("2199023255552"),
    qd_real("240141729555")/qd_real("17592186044416"),
    qd_real("176848560525")/qd_real("140737488355328"),
    qd_real("51538723353")/qd_real("562949953421312"),
    qd_real("1450433115")/qd_real("281474976710656"),
    qd_real("977699359")/qd_real("4503599627370496"),
    qd_real("118183439")/qd_real("18014398509481984"),
    qd_real("9652005")/qd_real("72057594037927936"),
    qd_real("121737")/qd_real("72057594037927936"),
    qd_real("6545")/qd_real("576460752303423488"),
    qd_real("561")/qd_real("18446744073709551616"),
    qd_real("1")/qd_real("73786976294838206464")
}
```

Definition at line 1541 of file [matrix_multi_imp.h](#).

The documentation for this struct was generated from the following file:

- [glucat/matrix_multi_imp.h](#)

6.39 pade::pade_sqrt_numer< Scalar_T > Struct Template Reference

Coefficients of numerator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)

```
#include <matrix_multi_imp.h>
```

Public Types

- using [array](#) = std::array<Scalar_T, 14>

Static Public Attributes

- static const [array](#) [number](#)

6.39.1 Detailed Description

```
template<typename Scalar_T>
struct pade::pade_sqrt_number< Scalar_T >
```

Coefficients of numerator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)

Definition at line [1384](#) of file [matrix_multi_imp.h](#).

6.39.2 Member Typedef Documentation

6.39.2.1 array

```
template<typename Scalar_T >
using pade::pade\_sqrt\_number< Scalar_T >::array = std::array<Scalar_T, 14>
```

Definition at line [1386](#) of file [matrix_multi_imp.h](#).

6.39.3 Member Data Documentation

6.39.3.1 number

```
template<typename Scalar_T >
const pade\_sqrt\_number< longdouble >::array pade::pade\_sqrt\_number< Scalar_T >::number [static]
```

Initial value:

```
=
{
    1.0,                27.0/4.0,        81.0/4.0,        2277.0/64.0,
    10395.0/256.0,      32319.0/1024.0,   8721.0/512.0,   26163.0/4096.0,
    53703.0/32768.0,    36465.0/131072.0, 3861.0/131072.0, 7371.0/4194304.0,
    819.0/16777216.0,   27.0/67108864.0
}
```

Definition at line [1387](#) of file [matrix_multi_imp.h](#).

The documentation for this struct was generated from the following file:

- [glucat/matrix_multi_imp.h](#)

6.40 `pade::pade_sqrt_numer< dd_real >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using `array` = `std::array<dd_real, 22>`

Static Public Attributes

- static const `array numer`

6.40.1 Detailed Description

Definition at line 1471 of file `matrix_multi_imp.h`.

6.40.2 Member Typedef Documentation

6.40.2.1 `array`

```
using pade::pade_sqrt_numer< dd_real >::array = std::array<dd_real, 22>
```

Definition at line 1473 of file `matrix_multi_imp.h`.

6.40.3 Member Data Documentation

6.40.3.1 `numer`

```
const pade_sqrt_numer< dd_real >::array pade::pade_sqrt_numer< dd_real >::numer [static]
```

Initial value:

```
=
{
    dd_real("1"),
    dd_real("215")/dd_real("4"),
    dd_real("90687")/dd_real("256"),
    dd_real("168861")/dd_real("256"),
    dd_real("7228859")/dd_real("16384"),
    dd_real("7538115")/dd_real("65536"),
    dd_real("195747825")/dd_real("16777216"),
    dd_real("14375115")/dd_real("33554432"),
    dd_real("20764055")/dd_real("4294967296"),
    dd_real("206701")/dd_real("17179869184"),
    dd_real("3311")/dd_real("1099511627776"),
    dd_real("43")/dd_real("4"),
    dd_real("10621")/dd_real("64"),
    dd_real("567987")/dd_real("1024"),
    dd_real("1246355")/dd_real("2048"),
    dd_real("16583853")/dd_real("65536"),
    dd_real("173376645")/dd_real("4194304"),
    dd_real("171655785")/dd_real("67108864"),
    dd_real("14375115")/dd_real("268435456"),
    dd_real("5167525")/dd_real("17179869184"),
    dd_real("76153")/dd_real("274877906944"),
    dd_real("43")/dd_real("4398046511104")
}
```

Definition at line 1474 of file `matrix_multi_imp.h`.

The documentation for this struct was generated from the following file:

- `glucat/matrix_multi_imp.h`

6.41 `pade::pade_sqrt_numer< float >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using `array` = `std::array<float, 10>`

Static Public Attributes

- static const `array numer`

6.41.1 Detailed Description

Definition at line 1416 of file `matrix_multi_imp.h`.

6.41.2 Member Typedef Documentation

6.41.2.1 `array`

```
using pade::pade_sqrt_numer< float >::array = std::array<float, 10>
```

Definition at line 1418 of file `matrix_multi_imp.h`.

6.41.3 Member Data Documentation

6.41.3.1 `numer`

```
const pade_sqrt_numer< float >::array pade::pade_sqrt_numer< float >::numer [static]
```

Initial value:

```
=
{
    1.0,          19.0/4.0,      19.0/2.0,      665.0/64.0,
    1729.0/256.0,  2717.0/1024.0,  627.0/1024.0,  627.0/8192.0,
    285.0/65536.0, 19.0/262144.0
}
```

Definition at line 1419 of file `matrix_multi_imp.h`.

The documentation for this struct was generated from the following file:

- `glucat/matrix_multi_imp.h`

6.42 `pade::pade_sqrt_numer< long double >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using `array` = `std::array<long double, 18>`

Static Public Attributes

- static const `array numer`

6.42.1 Detailed Description

Definition at line 1441 of file `matrix_multi_imp.h`.

6.42.2 Member Typedef Documentation

6.42.2.1 `array`

```
using pade::pade_sqrt_numer< long double >::array = std::array<long double, 18>
```

Definition at line 1443 of file `matrix_multi_imp.h`.

6.42.3 Member Data Documentation

6.42.3.1 `numer`

```
const array pade::pade_sqrt_numer< long double >::numer [static]
```

Definition at line 1444 of file `matrix_multi_imp.h`.

The documentation for this struct was generated from the following file:

- `glucat/matrix_multi_imp.h`

6.43 `pade::pade_sqrt_numer< qd_real >` Struct Reference

```
#include <matrix_multi_imp.h>
```

Public Types

- using `array` = `std::array<qd_real, 34>`

Static Public Attributes

- static const `array numer`

6.43.1 Detailed Description

Definition at line 1512 of file [matrix_multi_imp.h](#).

6.43.2 Member Typedef Documentation

6.43.2.1 array

```
using pade::pade_sqrt_number< qd_real >::array = std::array<qd_real, 34>
```

Definition at line 1514 of file [matrix_multi_imp.h](#).

6.43.3 Member Data Documentation

6.43.3.1 numer

```
const pade_sqrt_number< qd_real >::array pade::pade_sqrt_number< qd_real >::numer [static]
```

Initial value:

```
=
{
    qd_real("1"),
    qd_real("134"),
    qd_real("633485")/qd_real("256"),
    qd_real("15246721")/qd_real("1024"),
    qd_real("2518145487")/qd_real("65536"),
    qd_real("6344873535")/qd_real("131072"),
    qd_real("267226297065")/qd_real("8388608"),
    qd_real("379874182975")/qd_real("33554432"),
    qd_real("9425348845815")/qd_real("4294967296"),
    qd_real("987417498133")/qd_real("4294967296"),
    qd_real("8055248011085")/qd_real("137438953472"),
    qd_real("6958363175533")/qd_real("549755813888"),
    qd_real("5056698705201")/qd_real("2199023255552"),
    qd_real("766166470485")/qd_real("2199023255552"),
    qd_real("766166470485")/qd_real("17592186044416"),
    qd_real("623623871325")/qd_real("140737488355328"),
    qd_real("203123203803")/qd_real("562949953421312"),
    qd_real("6478601247")/qd_real("281474976710656"),
    qd_real("5038912081")/qd_real("4503599627370496"),
    qd_real("719844583")/qd_real("18014398509481984"),
    qd_real("71853815")/qd_real("72057594037927936"),
    qd_real("1165197")/qd_real("72057594037927936"),
    qd_real("87703")/qd_real("576460752303423488"),
    qd_real("12529")/qd_real("18446744073709551616"),
    qd_real("67")/qd_real("73786976294838206464")
}
```

Definition at line 1515 of file [matrix_multi_imp.h](#).

The documentation for this struct was generated from the following file:

- [glucat/matrix_multi_imp.h](#)

6.44 glucat::numeric_traits< Scalar_T >::promoted Struct Reference

Extra traits which extend numeric limits.

```
#include <promotion.h>
```

Public Types

- using [type](#) = double
- using [type](#) = long double

6.44.1 Detailed Description

```
template<typename Scalar_T>
struct glucat::numeric_traits< Scalar_T >::promoted
```

Extra traits which extend numeric limits.

Promoted type.

Promoted type for long double.

Promoted type for double

Definition at line [145](#) of file [scalar.h](#).

6.44.2 Member Typedef Documentation

6.44.2.1 [type](#) [1/2]

```
template<typename Scalar_T >
typedef double glucat::numeric\_traits< Scalar_T >::promoted::type = double
```

Definition at line [72](#) of file [promotion.h](#).

6.44.2.2 [type](#) [2/2]

```
template<typename Scalar_T >
using glucat::numeric\_traits< Scalar_T >::promoted::type = long double
```

Definition at line [86](#) of file [promotion.h](#).

The documentation for this struct was generated from the following files:

- [glucat/promotion.h](#)
- [glucat/scalar.h](#)

6.45 [glucat::random_generator](#)< [Scalar_T](#) > Class Template Reference

Random number generator with single instance per [Scalar_T](#).

```
#include <random.h>
```

Public Member Functions

- [random_generator](#) (const [random_generator](#) &)=delete
- auto [operator=](#) (const [random_generator](#) &) -> [random_generator](#) &=delete
- auto [uniform](#) () -> Scalar_T
- auto [normal](#) () -> Scalar_T

Static Public Member Functions

- static auto [generator](#) () -> [random_generator](#) &
Single instance of Random number generator.

Private Member Functions

- [random_generator](#) ()
- [~random_generator](#) ()=default

Private Attributes

- std::mt19937 [uint_gen](#)
- std::uniform_real_distribution< double > [uniform_dist](#)
- std::normal_distribution< double > [normal_dist](#)

Static Private Attributes

- static const unsigned long [seed](#) = 19590921UL

Friends

- class [friend_for_private_destructor](#)

6.45.1 Detailed Description

```
template<typename Scalar_T>
class glucat::random_generator< Scalar_T >
```

Random number generator with single instance per Scalar_T.

Definition at line 42 of file [random.h](#).

6.45.2 Constructor & Destructor Documentation

6.45.2.1 random_generator() [1/2]

```
template<typename Scalar_T >
glucat::random_generator< Scalar_T >::random_generator (
    const random\_generator< Scalar_T > & ) [delete]
```

6.45.2.2 `random_generator()` [2/2]

```
template<typename Scalar_T >
glucat::random_generator< Scalar_T >::random_generator () [inline], [private]
```

Definition at line 61 of file [random.h](#).

References [glucat::random_generator< Scalar_T >::seed](#).

6.45.2.3 `~random_generator()`

```
template<typename Scalar_T >
glucat::random_generator< Scalar_T >::~~random_generator () [private], [default]
```

6.45.3 Member Function Documentation

6.45.3.1 `generator()`

```
template<typename Scalar_T >
static auto glucat::random_generator< Scalar_T >::generator () -> random_generator& [inline],
[static]
```

Single instance of Random number generator.

Definition at line 51 of file [random.h](#).

6.45.3.2 `normal()`

```
template<typename Scalar_T >
auto glucat::random_generator< Scalar_T >::normal () -> Scalar_T [inline]
```

Definition at line 70 of file [random.h](#).

References [glucat::random_generator< Scalar_T >::normal_dist](#).

6.45.3.3 `operator=()`

```
template<typename Scalar_T >
auto glucat::random_generator< Scalar_T >::operator= (
    const random_generator< Scalar_T > & ) -> random_generator &=delete [delete]
```

6.45.3.4 `uniform()`

```
template<typename Scalar_T >
auto glucat::random_generator< Scalar_T >::uniform () -> Scalar_T [inline]
```

Definition at line 68 of file [random.h](#).

References [glucat::random_generator< Scalar_T >::uniform_dist](#).

6.45.4 Friends And Related Symbol Documentation

6.45.4.1 friend_for_private_destructor

```
template<typename Scalar_T >
friend class friend_for_private_destructor [friend]
```

Friend declaration to avoid compiler warning: "... only defines a private destructor and has no friends" Ref: Carlos O'Ryan, ACE <http://doc.ece.uci.edu>

Definition at line 48 of file [random.h](#).

6.45.5 Member Data Documentation

6.45.5.1 normal_dist

```
template<typename Scalar_T >
std::normal_distribution<double> glucat::random_generator< Scalar_T >::normal_dist [private]
```

Definition at line 59 of file [random.h](#).

Referenced by [glucat::random_generator< Scalar_T >::normal\(\)](#).

6.45.5.2 seed

```
template<typename Scalar_T >
const unsigned long glucat::random_generator< Scalar_T >::seed = 19590921UL [static], [private]
```

Definition at line 55 of file [random.h](#).

Referenced by [glucat::random_generator< Scalar_T >::random_generator\(\)](#).

6.45.5.3 uint_gen

```
template<typename Scalar_T >
std::mt19937 glucat::random_generator< Scalar_T >::uint_gen [private]
```

Definition at line 57 of file [random.h](#).

6.45.5.4 uniform_dist

```
template<typename Scalar_T >
std::uniform_real_distribution<double> glucat::random_generator< Scalar_T >::uniform_dist
[private]
```

Definition at line 58 of file [random.h](#).

Referenced by [glucat::random_generator< Scalar_T >::uniform\(\)](#).

The documentation for this class was generated from the following file:

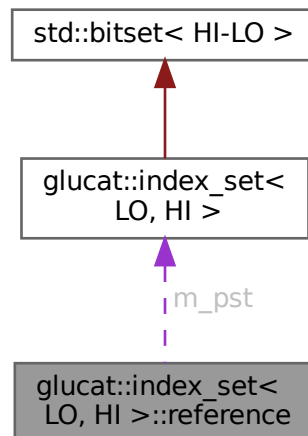
- [glucat/random.h](#)

6.46 glucat::index_set< LO, HI >::reference Class Reference

Index set member reference.

```
#include <index_set.h>
```

Collaboration diagram for glucat::index_set< LO, HI >::reference:



Public Member Functions

- `reference ()=delete`
Default constructor is deleted.
- `reference (index_set_t &ist, index_t idx)`
index_set reference
- `~reference ()=default`
- `auto operator== (const reference &c_j) const -> bool`
for b[i] == c[j];
- `auto operator= (const bool x) -> reference &`
for b[i] = x;
- `auto operator= (const reference &c_j) -> reference &`
for b[i] = c[j];
- `auto operator~ () const -> bool`
Flips a bit.
- `operator bool () const`
for x = b[i];
- `auto flip () -> reference &`
for b[i].flip();

Private Attributes

- `index_set_t * m_pst`
- `index_t m_idx`

Friends

- class [index_set](#)

6.46.1 Detailed Description

template<const [index_t](#) LO, const [index_t](#) HI>
class glucat::index_set< LO, HI >::reference

Index set member reference.

Definition at line 177 of file [index_set.h](#).

6.46.2 Constructor & Destructor Documentation

6.46.2.1 reference() [1/2]

```
template<const index\_t LO, const index\_t HI>
glucat::index_set< LO, HI >::reference::reference () [delete]
```

Default constructor is deleted.

6.46.2.2 reference() [2/2]

```
template<const index\_t LO, const index\_t HI>
glucat::index_set< LO, HI >::reference::reference (
    index\_set\_t & ist,
    index\_t idx) [inline]
```

[index_set](#) reference

Definition at line 985 of file [index_set_imp.h](#).

6.46.2.3 ~reference()

```
template<const index\_t LO, const index\_t HI>
glucat::index_set< LO, HI >::reference::~~reference () [default]
```

6.46.3 Member Function Documentation

6.46.3.1 flip()

```
template<const index\_t LO, const index\_t HI>
auto glucat::index_set< LO, HI >::reference::flip () -> reference& [inline]
```

for b[i].[flip\(\)](#);

Definition at line 1049 of file [index_set_imp.h](#).

6.46.3.2 operator bool()

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::reference::operator bool () const [inline]
```

for x = b[i];

Definition at line 1041 of file [index_set_imp.h](#).

6.46.3.3 operator=() [1/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reference::operator= (
    const bool x) -> reference& [inline]
```

for b[i] = x;

Definition at line 1003 of file [index_set_imp.h](#).

6.46.3.4 operator=() [2/2]

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reference::operator= (
    const reference & c_j) -> reference& [inline]
```

for b[i] = c[j];

Definition at line 1017 of file [index_set_imp.h](#).

6.46.3.5 operator==()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reference::operator== (
    const reference & c_j) const -> bool [inline]
```

for b[i] == c[j];

Definition at line 995 of file [index_set_imp.h](#).

6.46.3.6 operator~()

```
template<const index_t LO, const index_t HI>
auto glucat::index_set< LO, HI >::reference::operator~ () const -> bool [inline]
```

Flips a bit.

flips the bit

Definition at line 1034 of file [index_set_imp.h](#).

6.46.4 Friends And Related Symbol Documentation

6.46.4.1 `index_set`

```
template<const index\_t LO, const index\_t HI>
friend class index\_set [friend]
```

Definition at line 178 of file [index_set.h](#).

6.46.5 Member Data Documentation

6.46.5.1 `m_idx`

```
template<const index\_t LO, const index\_t HI>
index\_t glucat::index\_set< LO, HI >::reference::m_idx [private]
```

Definition at line 200 of file [index_set.h](#).

6.46.5.2 `m_pst`

```
template<const index\_t LO, const index\_t HI>
index\_set\_t\* glucat::index\_set< LO, HI >::reference::m_pst [private]
```

Definition at line 199 of file [index_set.h](#).

The documentation for this class was generated from the following files:

- [glucat/index_set.h](#)
- [glucat/index_set_imp.h](#)

6.47 `glucat::sorted_range< Map_T, Sorted_Map_T >` Class Template Reference

Sorted range for use with output.

```
#include <framed_multi_imp.h>
```

Public Types

- using [map_t](#) = `Map_T`
- using [sorted_map_t](#) = `Sorted_Map_T`
- using [sorted_iterator](#) = `typename Sorted_Map_T::const_iterator`

Public Member Functions

- [sorted_range](#) (`Sorted_Map_T &sorted_val, const Map_T &val`)

Public Attributes

- [sorted_iterator sorted_begin](#)
- [sorted_iterator sorted_end](#)

6.47.1 Detailed Description

```
template<typename Map_T, typename Sorted_Map_T>
class glucat::sorted_range< Map_T, Sorted_Map_T >
```

Sorted range for use with output.

Definition at line 1112 of file [framed_multi_imp.h](#).

6.47.2 Member Typedef Documentation

6.47.2.1 map_t

```
template<typename Map_T , typename Sorted_Map_T >
using glucat::sorted_range< Map_T, Sorted_Map_T >::map_t = Map_T
```

Definition at line 1115 of file [framed_multi_imp.h](#).

6.47.2.2 sorted_iterator

```
template<typename Map_T , typename Sorted_Map_T >
using glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_iterator = typename Sorted_Map_T<
::const_iterator
```

Definition at line 1117 of file [framed_multi_imp.h](#).

6.47.2.3 sorted_map_t

```
template<typename Map_T , typename Sorted_Map_T >
using glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_map_t = Sorted_Map_T
```

Definition at line 1116 of file [framed_multi_imp.h](#).

6.47.3 Constructor & Destructor Documentation

6.47.3.1 sorted_range()

```
template<typename Map_T , typename Sorted_Map_T >
glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_range (
    Sorted_Map_T & sorted_val,
    const Map_T & val) [inline]
```

Definition at line 1119 of file [framed_multi_imp.h](#).

References [glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_begin](#), and [glucat::sorted_range< Map_T, Sorted_Map_T >::s](#)

6.47.4 Member Data Documentation

6.47.4.1 `sorted_begin`

```
template<typename Map_T , typename Sorted_Map_T >
sorted_iterator glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_begin
```

Definition at line 1126 of file `framed_multi_imp.h`.

Referenced by `glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_range()`.

6.47.4.2 `sorted_end`

```
template<typename Map_T , typename Sorted_Map_T >
sorted_iterator glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_end
```

Definition at line 1127 of file `framed_multi_imp.h`.

Referenced by `glucat::sorted_range< Map_T, Sorted_Map_T >::sorted_range()`.

The documentation for this class was generated from the following file:

- `glucat/framed_multi_imp.h`

6.48 `glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >` Class Template Reference

```
#include <framed_multi_imp.h>
```

Public Types

- using `map_t` = `Sorted_Map_T`
- using `sorted_map_t` = `Sorted_Map_T`
- using `sorted_iterator` = `typename Sorted_Map_T::const_iterator`

Public Member Functions

- `sorted_range` (`Sorted_Map_T &sorted_val, const Sorted_Map_T &val`)

Public Attributes

- `sorted_iterator sorted_begin`
- `sorted_iterator sorted_end`

6.48.1 Detailed Description

```
template<typename Sorted_Map_T>
class glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >
```

Definition at line 1131 of file [framed_multi_imp.h](#).

6.48.2 Member Typedef Documentation

6.48.2.1 map_t

```
template<typename Sorted_Map_T >
using glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >::map_t = Sorted_Map_T
```

Definition at line 1134 of file [framed_multi_imp.h](#).

6.48.2.2 sorted_iterator

```
template<typename Sorted_Map_T >
using glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >::sorted_iterator = typename Sorted_Map_T::const_iterator
```

Definition at line 1136 of file [framed_multi_imp.h](#).

6.48.2.3 sorted_map_t

```
template<typename Sorted_Map_T >
using glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >::sorted_map_t = Sorted_Map_T
```

Definition at line 1135 of file [framed_multi_imp.h](#).

6.48.3 Constructor & Destructor Documentation

6.48.3.1 sorted_range()

```
template<typename Sorted_Map_T >
glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >::sorted_range (
    Sorted_Map_T & sorted_val,
    const Sorted_Map_T & val) [inline]
```

Definition at line 1138 of file [framed_multi_imp.h](#).

6.48.4 Member Data Documentation

6.48.4.1 sorted_begin

```
template<typename Sorted_Map_T >
sorted_iterator glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >::sorted_begin
```

Definition at line 1142 of file [framed_multi_imp.h](#).

6.48.4.2 sorted_end

```
template<typename Sorted_Map_T >  
sorted_iterator glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >::sorted_end
```

Definition at line 1143 of file [framed_multi_imp.h](#).

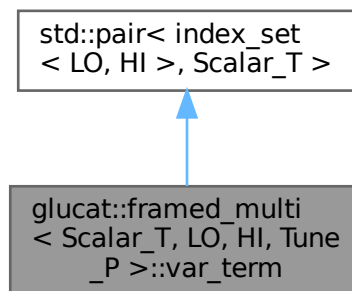
The documentation for this class was generated from the following file:

- [glucat/framed_multi_imp.h](#)

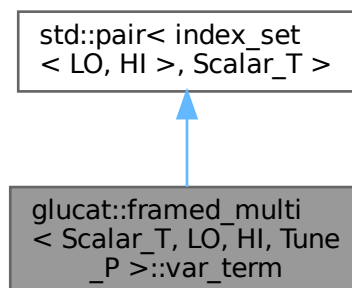
6.49 glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term Class Reference

Variable term.

Inheritance diagram for glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term:



Collaboration diagram for glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term:



Public Types

- using `var_pair_t` = `std::pair<index_set<LO, HI>, Scalar_T>`

Public Member Functions

- `~var_term` ()=default
Destructor.
- `var_term` ()
Default constructor.
- `var_term` (const `index_set_t` ist, const `Scalar_T` &crd=`Scalar_T`(1))
Construct a variable term from an index set and a scalar coordinate.
- auto `operator*=` (const `term_t` &rhs) -> `var_term_t` &
Product of variable term and term.

Static Public Member Functions

- static auto `classname` () -> const `std::string`
Class name used in messages.

6.49.1 Detailed Description

```
template<typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P>
class glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term
```

Variable term.

Definition at line 279 of file `framed_multi.h`.

6.49.2 Member Typedef Documentation

6.49.2.1 var_pair_t

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
using glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term::var_pair_t = std::pair<index_set<LO,
HI>, Scalar_T>
```

Definition at line 283 of file `framed_multi.h`.

6.49.3 Constructor & Destructor Documentation

6.49.3.1 ~var_term()

```
template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term::~~var_term () [default]
```

Destructor.

6.49.3.2 var_term() [1/2]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::var_term::var_term () [inline]
```

Default constructor.

Definition at line 291 of file [framed_multi.h](#).

6.49.3.3 var_term() [2/2]

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::var_term::var_term (
    const index\_set\_t ist,
    const Scalar_T & crd = Scalar_T(1)) [inline]
```

Construct a variable term from an index set and a scalar coordinate.

Definition at line 295 of file [framed_multi.h](#).

6.49.4 Member Function Documentation

6.49.4.1 classname()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
static auto glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::var_term::classname () -> const
std::string [inline], [static]
```

Class name used in messages.

Definition at line 286 of file [framed_multi.h](#).

6.49.4.2 operator*=()

```
template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >
auto glucat::framed\_multi< Scalar_T, LO, HI, Tune_P >::var_term::operator*= (
    const term\_t & rhs) -> var\_term\_t& [inline]
```

Product of variable term and term.

Definition at line 299 of file [framed_multi.h](#).

The documentation for this class was generated from the following file:

- [glucat/framed_multi.h](#)

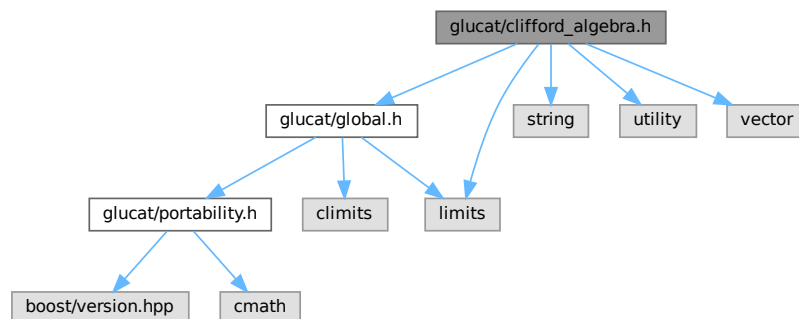
Chapter 7

File Documentation

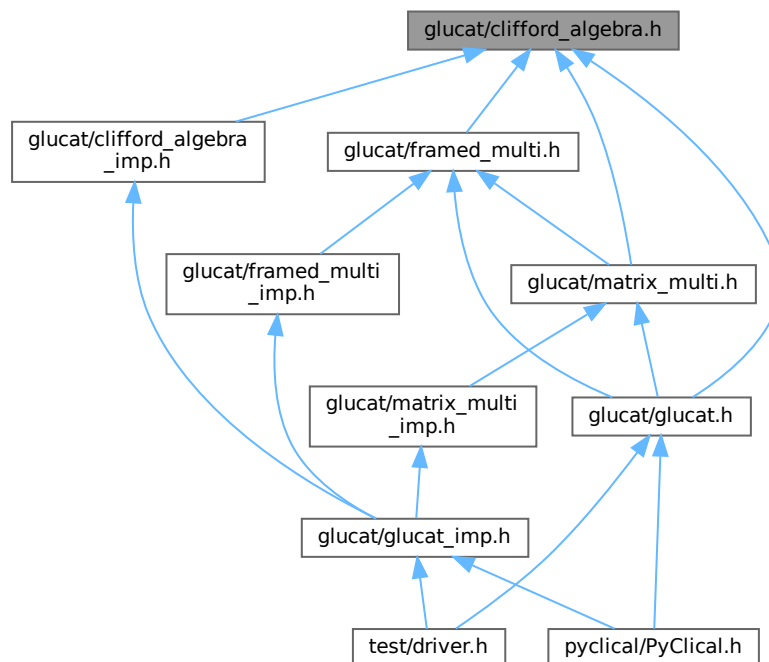
7.1 glucat/clifford_algebra.h File Reference

```
#include "glucat/global.h"  
#include <limits>  
#include <string>  
#include <utility>  
#include <vector>
```

Include dependency graph for clifford_algebra.h:



This graph shows which files directly or indirectly include this file:



Classes

- class `glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >`
clifford_algebra<> declares the operations of a Clifford algebra

Namespaces

- namespace `glucat`

Macros

- `#define _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS`

Functions

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::operator!= (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> bool`
Test for inequality of multivectors.
- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::operator!= (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> bool`

Test for inequality of multivector and scalar.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::operator!= (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> bool`

Test for inequality of scalar and multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::error_squared_tol (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

Quadratic norm error tolerance relative to a specific multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::error_squared (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs, const Scalar_T threshold) -> Scalar_T`

Relative or absolute error using the quadratic norm.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::approx_equal (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs, const Scalar_T threshold, const Scalar_T tolerance) -> bool`

Test for approximate equality of multivectors.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::approx_equal (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> bool`

Test for approximate equality of multivectors.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::operator+ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric sum of multivector and scalar.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::operator+ (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric sum of scalar and multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::operator+ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric sum.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::operator- (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric difference of multivector and scalar.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::operator- (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric difference of scalar and multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::operator- (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Geometric difference.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::inv (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`
Geometric multiplicative inverse.
- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, int rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`
Integer power of multivector.
- `template<template< typename, const index_t, const index_t, typename > class Multivector, template< typename, const index_t, const index_t, typename > class RHS, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`
Multivector power of multivector.
- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::outer_pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, int rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`
Outer product power of multivector.
- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::scalar (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`
Scalar part.
- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::real (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`
Real part: synonym for scalar part.
- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::imag (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`
Imaginary part: deprecated (always 0)
- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::pure (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`
Pure part.
- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::even (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`
Even part.
- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::odd (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`
Odd part.
- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::vector_part (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const std::vector< Scalar_T >`
Vector part of multivector, as a `vector_t` with respect to frame()
- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::involute (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Main involution, each $\{i\}$ is replaced by $-\{i\}$ in each term, eg. $\{1\}\{2\} \rightarrow (-\{2\})*(-\{1\})$*

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::reverse (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Reversion, eg. $\{1\}\{2\} \rightarrow \{2\}*\{1\}$.*

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::conj (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Conjugation, rev o invo == invo o rev.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::quad (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

*Scalar_T quadratic form == $(rev(x)*x)(0)$*

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::norm (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

Scalar_T norm == sum of norm of coordinates.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::abs (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

Absolute value == \sqrt{norm}

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::max_abs (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

Maximum of absolute values of components of multivector: multivector infinity norm.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::complexifier (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Square root of -1 which commutes with all members of the frame of the given multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::elliptic (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`
- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::sqrt (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Square root of multivector with specified complexifier.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::sqrt (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Square root of multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::clifford_exp (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Exponential of multivector.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::log (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Natural logarithm of multivector with specified complexifier.

- `template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >`
`auto glucat::log (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

Natural logarithm of multivector.

- ```

• template<template< typename, const index_t, const index_t, typename > class Multivector, typename Scalar_T, const index_t LO,
const index_t HI, typename Tune_P >
auto glucatt::cos (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI,
Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >

```

*Cosine of multivector with specified complexifier.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T, const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::cos (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Cosine of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T, const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::acos (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Inverse cosine of multivector with specified complexifier.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T, const index\_t LO, const index\_t HI, typename Tune_P >  
auto glucat::acos (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Inverse cosine of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T, const index\_t LO, const index\_t HI, typename Tune_P >  
auto glucat::cosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Hyperbolic cosine of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T, const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucatt::acosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Inverse hyperbolic cosine of multivector with specified complexifier.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T, const index\_t LO, const index\_t HI, typename Tune_P >  
auto glucatt::acosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Inverse hyperbolic cosine of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T, const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucats::sin (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Sine of multivector with specified complexifier.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T, const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucats::sin (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Sine of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T, const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::asin (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`





## 7.1.1 Macro Definition Documentation

### 7.1.1.1 \_GLUCAT\_CLIFFORD\_ALGEBRA\_OPERATIONS

#define \_GLUCAT\_CLIFFORD\_ALGEBRA\_OPERATIONS

Definition at line 145 of file clifford\_algebra.h.

## 7.2 clifford\_algebra.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_CLIFFORD_ALGEBRA_H
00002 #define _GLUCAT_CLIFFORD_ALGEBRA_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 clifford_algebra.h : Declare the operations of a Clifford algebra
00006 -----
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2021 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 #include "glucat/global.h"
00035
00036 #include <limits>
00037 #include <string>
00038 #include <utility>
00039 #include <vector>
00040
00041 namespace glucat
00042 {
00043 template< typename Scalar_T, typename Index_Set_T, typename Multivector_T>
00044 class clifford_algebra
00045 {
00046 public:
00047 using scalar_t = Scalar_T;
00048 using index_set_t = Index_Set_T;
00049 static const index_t v_lo = index_set_t::v_lo;
00050 static const index_t v_hi = index_set_t::v_hi;
00051 using multivector_t = Multivector_T;
00052 using pair_t = std::pair<const index_set_t, Scalar_T>;
00053 using vector_t = std::vector<Scalar_T>;
00054
00055 static auto classname() -> const std::string;
00056
00057 static const Scalar_T default_truncation;
00058
00059 virtual ~clifford_algebra() = default;
00060
00061 // clifford_algebra operations
00062 virtual auto operator== (const multivector_t& val) const -> bool = 0;
00063 virtual auto operator== (const Scalar_T& scr) const -> bool = 0;
00064 virtual auto operator+= (const multivector_t& rhs) -> multivector_t& = 0;
00065 virtual auto operator+= (const Scalar_T& scr) -> multivector_t& = 0;
00066 virtual auto operator-= (const multivector_t& rhs) -> multivector_t& = 0;

```

```

00075 virtual auto operator== (const Scalar_T& scr) -> multivector_t& = 0;
00077 virtual auto operator- (const Scalar_T& scr) -> multivector_t = 0;
00079 virtual auto operator* (const Scalar_T& scr) -> multivector_t& = 0;
00081 virtual auto operator* (const multivector_t& rhs) -> multivector_t& = 0;
00083 virtual auto operator% (const multivector_t& rhs) -> multivector_t& = 0;
00085 virtual auto operator& (const multivector_t& rhs) -> multivector_t& = 0;
00087 virtual auto operator^ (const multivector_t& rhs) -> multivector_t& = 0;
00089 virtual auto operator/= (const Scalar_T& scr) -> multivector_t& = 0;
00091 virtual auto operator/= (const multivector_t& rhs) -> multivector_t& = 0;
00093 virtual auto operator|= (const multivector_t& rhs) -> multivector_t& = 0;
00095 virtual auto inv (const multivector_t& rhs) -> multivector_t = 0;
00097 virtual auto pow (int m) const -> const multivector_t = 0;
00099 virtual auto outer_pow (int m) const -> const multivector_t = 0;
00101 virtual auto frame (const multivector_t& rhs) -> multivector_t = 0;
00103 virtual auto grade (const multivector_t& rhs) -> index_t = 0;
00105 virtual auto operator[] (const index_set_t ist) const -> Scalar_T = 0;
00107 virtual auto operator() (const index_t grade) const -> const multivector_t = 0;
00109 virtual auto scalar (const multivector_t& rhs) -> Scalar_T = 0;
00111 virtual auto pure (const multivector_t& rhs) -> multivector_t = 0;
00113 virtual auto even (const multivector_t& rhs) -> multivector_t = 0;
00115 virtual auto odd (const multivector_t& rhs) -> multivector_t = 0;
00117 virtual auto vector_part (const multivector_t& rhs) -> vector_t = 0;
00119 virtual auto vector_part (const index_set_t frm, const bool prechecked) const -> const vector_t =
0;
00121 virtual auto involute (const multivector_t& rhs) -> multivector_t = 0;
00123 virtual auto reverse (const multivector_t& rhs) -> multivector_t = 0;
00125 virtual auto conj (const multivector_t& rhs) -> multivector_t = 0;
00127 virtual auto quad (const multivector_t& rhs) -> Scalar_T = 0;
00129 virtual auto norm (const multivector_t& rhs) -> Scalar_T = 0;
00131 virtual auto max_abs (const multivector_t& rhs) -> Scalar_T = 0;
00133 virtual auto truncated (const Scalar_T& limit = default_truncation) const -> const multivector_t
= 0;
00135 virtual auto isinf (const multivector_t& rhs) -> bool = 0;
00137 virtual auto isnan (const multivector_t& rhs) -> bool = 0;
00139 virtual void write (const std::string& msg="") const = 0;
00141 virtual void write (std::ofstream& ofile, const std::string& msg="") const = 0;
00142 };
00143
00144 #ifndef _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS
00145 #define _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS \
00146 auto operator== (const multivector_t& val) const -> bool override;\
00147 auto operator== (const Scalar_T& scr) const -> bool override;\
00148 auto operator+= (const multivector_t& rhs) -> multivector_t& override;\
00149 auto operator+= (const Scalar_T& scr) -> multivector_t& override;\
00150 auto operator-= (const multivector_t& rhs) -> multivector_t& override;\
00151 auto operator-= (const Scalar_T& scr) -> multivector_t& override;\
00152 auto operator- (const multivector_t& rhs) -> const multivector_t override;\
00153 auto operator* (const Scalar_T& scr) -> multivector_t& override;\
00154 auto operator* (const multivector_t& rhs) -> multivector_t& override;\
00155 auto operator% (const multivector_t& rhs) -> multivector_t& override;\
00156 auto operator& (const multivector_t& rhs) -> multivector_t& override;\
00157 auto operator^ (const multivector_t& rhs) -> multivector_t& override;\
00158 auto operator/= (const Scalar_T& scr) -> multivector_t& override;\
00159 auto operator/= (const multivector_t& rhs) -> multivector_t& override;\
00160 auto operator|= (const multivector_t& rhs) -> multivector_t& override;\
00161 auto inv (const multivector_t& rhs) -> const multivector_t override;\
00162 auto pow (int m) const -> const multivector_t override;\
00163 auto outer_pow (int m) const -> const multivector_t override;\
00164 auto frame (const multivector_t& rhs) -> const index_set_t override;\
00165 auto grade (const multivector_t& rhs) -> index_t override;\
00166 auto operator[] (const index_set_t ist) const -> Scalar_T override;\
00167 auto operator() (const index_t grade) const -> const multivector_t override;\
00168 auto scalar (const multivector_t& rhs) -> Scalar_T override;\
00169 auto pure (const multivector_t& rhs) -> const multivector_t override;\
00170 auto even (const multivector_t& rhs) -> const multivector_t override;\
00171 auto odd (const multivector_t& rhs) -> const multivector_t override;\
00172 auto vector_part (const multivector_t& rhs) -> const vector_t override;\
00173 auto vector_part (const index_set_t frm, const bool prechecked = false) const \
00174 -> const vector_t override;\
00175 auto involute (const multivector_t& rhs) -> const multivector_t override;\
00176 auto reverse (const multivector_t& rhs) -> const multivector_t override;\
00177 auto conj (const multivector_t& rhs) -> const multivector_t override;\
00178 auto quad (const multivector_t& rhs) -> Scalar_T override;\
00179 auto norm (const multivector_t& rhs) -> Scalar_T override;\
00180 auto max_abs (const multivector_t& rhs) -> Scalar_T override;\
00181 auto truncated (const Scalar_T& limit = multivector_t::default_truncation) const \
00182 -> const multivector_t override;\
00183 auto isinf (const multivector_t& rhs) -> bool override;\
00184 auto isnan (const multivector_t& rhs) -> bool override;\
00185 void write (const std::string& msg="") const override;\
00186 void write (std::ofstream& ofile, const std::string& msg="") const override;\
00187 #endif // _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS
00188
00189 template
00190 <
00191 template<typename, const index_t, const index_t, typename> class Multivector,
00192 template<typename, const index_t, const index_t, typename> class RHS,

```

```

00194 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00195 >
00196 auto
00197 operator!= (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
bool;
00198
00200 template
00201 <
00202 template<typename, const index_t, const index_t, typename> class Multivector,
00203 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00204 >
00205 auto
00206 operator!= (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> bool;
00207
00209 template
00210 <
00211 template<typename, const index_t, const index_t, typename> class Multivector,
00212 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00213 >
00214 auto
00215 operator!= (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> bool;
00216
00218 template
00219 <
00220 template<typename, const index_t, const index_t, typename> class Multivector,
00221 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00222 >
00223 auto
00224 error_squared_tol(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00225
00227 template
00228 <
00229 template<typename, const index_t, const index_t, typename> class Multivector,
00230 template<typename, const index_t, const index_t, typename> class RHS,
00231 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00232 >
00233 auto
00234 error_squared(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs,
00235 const RHS<Scalar_T,LO,HI,Tune_P>& rhs,
00236 const Scalar_T threshold) -> Scalar_T;
00237
00239 template
00240 <
00241 template<typename, const index_t, const index_t, typename> class Multivector,
00242 template<typename, const index_t, const index_t, typename> class RHS,
00243 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00244 >
00245 auto
00246 approx_equal(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs,
00247 const RHS<Scalar_T,LO,HI,Tune_P>& rhs,
00248 const Scalar_T threshold,
00249 const Scalar_T tolerance) -> bool;
00250
00252 template
00253 <
00254 template<typename, const index_t, const index_t, typename> class Multivector,
00255 template<typename, const index_t, const index_t, typename> class RHS,
00256 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00257 >
00258 auto
00259 approx_equal(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs,
00260 const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> bool;
00261
00263 template
00264 <
00265 template<typename, const index_t, const index_t, typename> class Multivector,
00266 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00267 >
00268 auto
00269 operator+ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00270
00272 template
00273 <
00274 template<typename, const index_t, const index_t, typename> class Multivector,
00275 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00276 >
00277 auto
00278 operator+ (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00279
00281 template
00282 <
00283 template<typename, const index_t, const index_t, typename> class Multivector,
00284 template<typename, const index_t, const index_t, typename> class RHS,
00285 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00286 >

```

```

00287 auto
00288 operator+ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>;
00289
00291 template
00292 <
00293 template<typename, const index_t, const index_t, typename> class Multivector,
00294 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00295 >
00296 auto
00297 operator- (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00298
00300 template
00301 <
00302 template<typename, const index_t, const index_t, typename> class Multivector,
00303 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00304 >
00305 auto
00306 operator- (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00307
00309 template
00310 <
00311 template<typename, const index_t, const index_t, typename> class Multivector,
00312 template<typename, const index_t, const index_t, typename> class RHS,
00313 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00314 >
00315 auto
00316 operator- (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>;
00317
00319 template
00320 <
00321 template<typename, const index_t, const index_t, typename> class Multivector,
00322 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00323 >
00324 auto
00325 operator* (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00326
00328 template
00329 <
00330 template<typename, const index_t, const index_t, typename> class Multivector,
00331 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00332 >
00333 auto
00334 operator* (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00335
00337 template
00338 <
00339 template<typename, const index_t, const index_t, typename> class Multivector,
00340 template<typename, const index_t, const index_t, typename> class RHS,
00341 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00342 >
00343 auto
00344 operator* (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>;
00345
00347 template
00348 <
00349 template<typename, const index_t, const index_t, typename> class Multivector,
00350 template<typename, const index_t, const index_t, typename> class RHS,
00351 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00352 >
00353 auto
00354 operator^ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>;
00355
00357 template
00358 <
00359 template<typename, const index_t, const index_t, typename> class Multivector,
00360 template<typename, const index_t, const index_t, typename> class RHS,
00361 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00362 >
00363 auto
00364 operator& (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>;
00365
00367 template
00368 <
00369 template<typename, const index_t, const index_t, typename> class Multivector,
00370 template<typename, const index_t, const index_t, typename> class RHS,
00371 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00372 >
00373 auto

```

```

00374 operator% (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>;
00375
00376 template
00377 <
00378 template<typename, const index_t, const index_t, typename> class Multivector,
00379 template<typename, const index_t, const index_t, typename> class RHS,
00380 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00381 >
00382 auto
00383 star (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
Scalar_T;
00384
00385 template
00386 <
00387 template<typename, const index_t, const index_t, typename> class Multivector,
00388 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00389 >
00390 auto
00391 operator/ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00392
00393 template
00394 <
00395 template<typename, const index_t, const index_t, typename> class Multivector,
00396 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00397 >
00398 auto
00399 operator/ (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00400
00401 template
00402 <
00403 template<typename, const index_t, const index_t, typename> class Multivector,
00404 template<typename, const index_t, const index_t, typename> class RHS,
00405 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00406 >
00407 auto
00408 operator/ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>;
00409
00410 template
00411 <
00412 template<typename, const index_t, const index_t, typename> class Multivector,
00413 template<typename, const index_t, const index_t, typename> class RHS,
00414 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00415 >
00416 auto
00417 operator| (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>;
00418
00419 template
00420 <
00421 template<typename, const index_t, const index_t, typename> class Multivector,
00422 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00423 >
00424 auto
00425 inv(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00426
00427 template
00428 <
00429 template<typename, const index_t, const index_t, typename> class Multivector,
00430 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00431 >
00432 auto
00433 pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, int rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00434
00435 template
00436 <
00437 template<typename, const index_t, const index_t, typename> class Multivector,
00438 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00439 >
00440 auto
00441 pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00442
00443 template< template<typename, const index_t, const index_t, typename> class Multivector,
00444 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00445 auto
00446 outer_pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, int rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00447
00448 template
00449 <
00450 template<typename, const index_t, const index_t, typename> class Multivector,

```

```

00462 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00463 >
00464 auto
00465 scalar(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00466
00467 template
00468 <
00469 template<typename, const index_t, const index_t, typename> class Multivector,
00470 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00471 >
00472 auto
00473 real(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00474
00475 template
00476 <
00477 template<typename, const index_t, const index_t, typename> class Multivector,
00478 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00479 >
00480 auto
00481 imag(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00482
00483 template
00484 <
00485 template<typename, const index_t, const index_t, typename> class Multivector,
00486 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00487 >
00488 auto
00489 pure(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00490
00491 template
00492 <
00493 template<typename, const index_t, const index_t, typename> class Multivector,
00494 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00495 >
00496 auto
00497 odd(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00498
00499 template
00500 <
00501 template<typename, const index_t, const index_t, typename> class Multivector,
00502 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00503 >
00504 auto
00505 even(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00506
00507 template
00508 <
00509 template<typename, const index_t, const index_t, typename> class Multivector,
00510 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00511 >
00512 auto
00513 vector_part(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const std::vector<Scalar_T>;
00514
00515 template
00516 <
00517 template<typename, const index_t, const index_t, typename> class Multivector,
00518 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00519 >
00520 auto
00521 involute(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00522
00523 template
00524 <
00525 template<typename, const index_t, const index_t, typename> class Multivector,
00526 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00527 >
00528 auto
00529 reverse(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00530
00531 template
00532 <
00533 template<typename, const index_t, const index_t, typename> class Multivector,
00534 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00535 >
00536 auto
00537 conj(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00538
00539 template
00540 <
00541 template<typename, const index_t, const index_t, typename> class Multivector,
00542 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00543 >
00544 auto
00545 quad(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00546
00547 template
00548 <
00549 template<typename, const index_t, const index_t, typename> class Multivector,
00550 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00551 >
00552 auto
00553 quad(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00554
00555 template
00556 <

```

```

00560 template<typename, const index_t, const index_t, typename> class Multivector,
00561 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00562 >
00563 auto
00564 norm(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00565
00566 template
00567 <
00568 template<typename, const index_t, const index_t, typename> class Multivector,
00569 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00570 >
00571 auto
00572 abs(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00573
00574 template
00575 <
00576 template<typename, const index_t, const index_t, typename> class Multivector,
00577 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00578 >
00579 auto
00580 max_abs(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00581
00582 template
00583 <
00584 template<typename, const index_t, const index_t, typename> class Multivector,
00585 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00586 >
00587 auto
00588 complexifier(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const
00589 Multivector<Scalar_T,LO,HI,Tune_P>;
00590
00591 template
00592 <
00593 template<typename, const index_t, const index_t, typename> class Multivector,
00594 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00595 >
00596 auto
00597 elliptic(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00598
00599 template
00600 <
00601 template<typename, const index_t, const index_t, typename> class Multivector,
00602 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00603 >
00604 auto
00605 sqrt(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00606 const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00607 const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00608
00609 template
00610 <
00611 template<typename, const index_t, const index_t, typename> class Multivector,
00612 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00613 >
00614 auto
00615 sqrt(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00616
00617 // Transcendental functions
00618
00619 template
00620 <
00621 template<typename, const index_t, const index_t, typename> class Multivector,
00622 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00623 >
00624 auto
00625 clifford_exp(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const
00626 Multivector<Scalar_T,LO,HI,Tune_P>;
00627
00628 template
00629 <
00630 template<typename, const index_t, const index_t, typename> class Multivector,
00631 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00632 >
00633 auto
00634 log(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00635 const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00636 const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00637
00638 template
00639 <
00640 template<typename, const index_t, const index_t, typename> class Multivector,
00641 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00642 >
00643 auto
00644 log(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00645
00646 template
00647 <
00648 template<typename, const index_t, const index_t, typename> class Multivector,
00649 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00650 >
00651 auto
00652 log(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00653
00654 template
00655 <

```

```

00656 template<typename, const index_t, const index_t, typename> class Multivector,
00657 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00658 >
00659 auto
00660 cos(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00661 const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00662 const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00663
00665 template
00666 <
00667 template<typename, const index_t, const index_t, typename> class Multivector,
00668 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00669 >
00670 auto
00671 cos(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00672
00674 template
00675 <
00676 template<typename, const index_t, const index_t, typename> class Multivector,
00677 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00678 >
00679 auto
00680 acos(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00681 const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00682 const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00683
00685 template
00686 <
00687 template<typename, const index_t, const index_t, typename> class Multivector,
00688 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00689 >
00690 auto
00691 acos(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00692
00694 template
00695 <
00696 template<typename, const index_t, const index_t, typename> class Multivector,
00697 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00698 >
00699 auto
00700 cosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00701
00703 template
00704 <
00705 template<typename, const index_t, const index_t, typename> class Multivector,
00706 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00707 >
00708 auto
00709 acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00710 const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00711 const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00712
00714 template
00715 <
00716 template<typename, const index_t, const index_t, typename> class Multivector,
00717 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00718 >
00719 auto
00720 acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00721
00723 template
00724 <
00725 template<typename, const index_t, const index_t, typename> class Multivector,
00726 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00727 >
00728 auto
00729 sin(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00730 const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00731 const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00732
00734 template
00735 <
00736 template<typename, const index_t, const index_t, typename> class Multivector,
00737 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00738 >
00739 auto
00740 sin(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00741
00743 template
00744 <
00745 template<typename, const index_t, const index_t, typename> class Multivector,
00746 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00747 >
00748 auto
00749 asin(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00750 const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00751 const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

```



```

00752
00753 template
00754 <
00755 template<typename, const index_t, const index_t, typename> class Multivector,
00756 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00757 >
00758 auto
00759 asin(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00760
00761 template
00762 <
00763 template<typename, const index_t, const index_t, typename> class Multivector,
00764 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00765 >
00766 auto
00767 sinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00768
00769 template
00770 <
00771 template<typename, const index_t, const index_t, typename> class Multivector,
00772 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00773 >
00774 auto
00775 asinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00776 const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00777 const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00778
00779 template
00780 <
00781 template<typename, const index_t, const index_t, typename> class Multivector,
00782 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00783 >
00784 auto
00785 asinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00786
00787 template
00788 <
00789 template<typename, const index_t, const index_t, typename> class Multivector,
00790 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00791 >
00792 auto
00793 tan(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00794 const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00795 const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00796
00797 template
00798 <
00799 template<typename, const index_t, const index_t, typename> class Multivector,
00800 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00801 >
00802 auto
00803 tan(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00804
00805 template
00806 <
00807 template<typename, const index_t, const index_t, typename> class Multivector,
00808 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00809 >
00810 auto
00811 atan(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00812 const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00813 const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00814
00815 template
00816 <
00817 template<typename, const index_t, const index_t, typename> class Multivector,
00818 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00819 >
00820 auto
00821 atan(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00822
00823 template
00824 <
00825 template<typename, const index_t, const index_t, typename> class Multivector,
00826 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00827 >
00828 auto
00829 atan(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00830
00831 template
00832 <
00833 template<typename, const index_t, const index_t, typename> class Multivector,
00834 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00835 >
00836 auto
00837 tanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00838
00839 template
00840 <
00841 template<typename, const index_t, const index_t, typename> class Multivector,
00842 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00843 >
00844 auto
00845 atanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00846 const Multivector<Scalar_T,LO,HI,Tune_P>& i,

```

```

00849 const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00850
00852 template
00853 <
00854 template<typename, const index_t, const index_t, typename> class Multivector,
00855 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00856 >
00857 auto
00858 atanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00859 }
00860 #endif // _GLUCAT_CLIFFORD_ALGEBRA_H

```

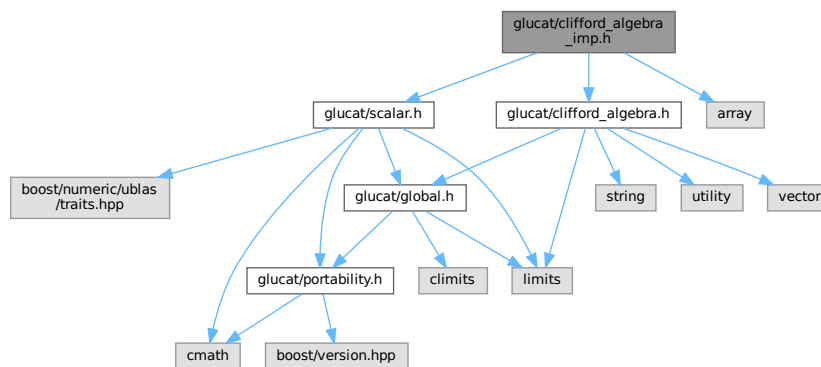
## 7.3 glucat/clifford\_algebra\_imp.h File Reference

```

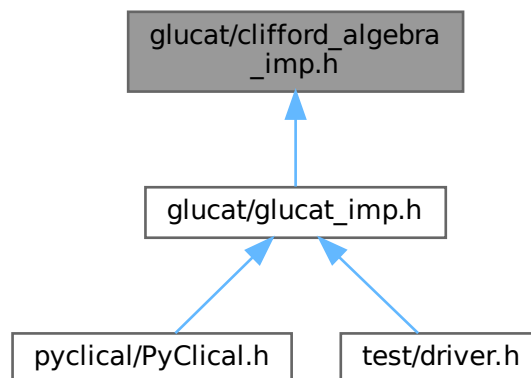
#include "glucat/clifford_algebra.h"
#include "glucat/scalar.h"
#include <array>

```

Include dependency graph for clifford\_algebra\_imp.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [glucat](#)

## Functions

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, template< typename, const index\_t, const index\_t, typename > class RHS, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::operator!= (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> bool`  
*Test for inequality of multivectors.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::operator!= (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> bool`  
*Test for inequality of multivector and scalar.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::operator!= (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> bool`  
*Test for inequality of scalar and multivector.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::error\_squared\_tol (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`  
*Quadratic norm error tolerance relative to a specific multivector.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, template< typename, const index\_t, const index\_t, typename > class RHS, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::error\_squared (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs, const Scalar_T threshold) -> Scalar_T`  
*Relative or absolute error using the quadratic norm.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, template< typename, const index\_t, const index\_t, typename > class RHS, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::approx\_equal (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs, const Scalar_T threshold, const Scalar_T tolerance) -> bool`  
*Test for approximate equality of multivectors.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, template< typename, const index\_t, const index\_t, typename > class RHS, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::approx\_equal (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> bool`  
*Test for approximate equality of multivectors.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::operator+ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const Scalar_T &scr) -> const Multivector< Scalar_T, LO, HI, Tune_P >`  
*Geometric sum of multivector and scalar.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::operator+ (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`  
*Geometric sum of scalar and multivector.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, template< typename, const index\_t, const index\_t, typename > class RHS, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::operator+ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`  
*Geometric sum.*



- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::operator/ (const Scalar_T &scr, const Multivector< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`  
*Quotient of scalar and multivector.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, template< typename, const index\_t, const index\_t, typename > class RHS, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::operator/ (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`  
*Geometric quotient.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, template< typename, const index\_t, const index\_t, typename > class RHS, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::operator| (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`  
*Transformation via twisted adjoint action.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::inv (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`  
*Geometric multiplicative inverse.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, int rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`  
*Integer power of multivector.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, template< typename, const index\_t, const index\_t, typename > class RHS, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, const RHS< Scalar_T, LO, HI, Tune_P > &rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`  
*Multivector power of multivector.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::outer\_pow (const Multivector< Scalar_T, LO, HI, Tune_P > &lhs, int rhs) -> const Multivector< Scalar_T, LO, HI, Tune_P >`  
*Outer product power of multivector.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::scalar (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`  
*Scalar part.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::real (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`  
*Real part: synonym for scalar part.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::imag (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`  
*Imaginary part: deprecated (always 0)*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::pure (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`  
*Pure part.*
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::even (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Even part.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::odd (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Odd part.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::vector\_part (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const std::vector< Scalar_←_T >`

*Vector part of multivector, as a [vector\\_t](#) with respect to frame()*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::involute (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Main involution, each {i} is replaced by -{i} in each term, eg. {1}\*{2} -> (-{2})\*(-{1})*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::reverse (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Reversion, eg. {1}\*{2} -> {2}\*{1}.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::conj (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Conjugation, rev o invo == invo o rev.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::quad (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

*Scalar\_T quadratic form == (rev(x)\*x)(0)*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::norm (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

*Scalar\_T norm == sum of norm of coordinates.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::abs (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

*Absolute value == sqrt(norm)*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::max\_abs (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> Scalar_T`

*Maximum of absolute values of components of multivector: multivector infinity norm.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::complexifier (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Square root of -1 which commutes with all members of the frame of the given multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::elliptic (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`
- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`static void glucat::check\_complex (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false)`

*Check that  $i$  is a valid complexifier for  $val$ .*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::sqrt (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Square root of multivector with specified complexifier.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::sqrt (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Square root of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::clifford\_exp (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Exponential of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::log (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Natural logarithm of multivector with specified complexifier.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::log (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Natural logarithm of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::cosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Hyperbolic cosine of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::acosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Inverse hyperbolic cosine of multivector with specified complexifier.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::acosh (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Inverse hyperbolic cosine of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::cos (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Cosine of multivector with specified complexifier.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::cos (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Cosine of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::acos (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`







*Inverse hyperbolic tangent of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::tan (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Tangent of multivector with specified complexifier.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::tan (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Tangent of multivector.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::atan (const Multivector< Scalar_T, LO, HI, Tune_P > &val, const Multivector< Scalar_T, LO, HI, Tune_P > &i, const bool prechecked=false) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Inverse tangent of multivector with specified complexifier.*

- `template<template< typename, const index\_t, const index\_t, typename > class Multivector, typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::atan (const Multivector< Scalar_T, LO, HI, Tune_P > &val) -> const Multivector< Scalar_T, LO, HI, Tune_P >`

*Inverse tangent of multivector.*

## 7.4 clifford\_algebra\_imp.h

[Go to the documentation of this file.](#)

```
00001 #ifndef _GLUCAT_CLIFFORD_ALGEBRA_IMP_H
00002 #define _GLUCAT_CLIFFORD_ALGEBRA_IMP_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 clifford_algebra_imp.h : Implement common Clifford algebra functions
00006 -----
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2021 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 // References for algorithms:
00035 // [AS]:
00036 // Milton Abramowicz and Irene A. Stegun, "Handbook of mathematical functions",
00037 // Dover 1972, first published 1965.
00038 // [CHKL]:
00039 // Sheung Hun Cheng, Nicholas J. Higham, Charles S. Kenney and Alan J. Laub,
00040 // "Approximating the Logarithm of a Matrix to Specified Accuracy", 1999.
00041 // ftp://ftp.ma.man.ac.uk/pub/narep/narep353.ps.gz
00042 // [GL]:
00043 // Gene H. Golub and Charles F. van Loan,
00044 // "Matrix Computations", 3rd ed., Johns Hopkins UP, 1996.
```

```

00045 // [GW]:
00046 // C.F. Gerald and P.O. Wheatley, "Applied Numerical Analysis",
00047 // 6th Edition, Addison-Wesley, 1999.
00048 // [H]:
00049 // Nicholas J. Higham
00050 // "The Scaling and Squaring Method for the Matrix Exponential Revisited",
00051 // SIAM Journal on Matrix Analysis and Applications,
00052 // Vol. 26, Issue 4 (2005), pp. 1179-1193.
00053 // [Z]:
00054 // Doron Zeilberger, "PADE" (Maple code), 2002.
00055 // http://www.math.rutgers.edu/~zeilberg/tokhniot/PADE
00056
00057 #include "glucat/clifford_algebra.h"
00058 #include "glucat/scalar.h"
00059
00060 #include <array>
00061
00062 namespace glucat
00063 {
00064 template< typename Scalar_T, typename Index_Set_T, typename Multivector_T>
00065 auto
00066 clifford_algebra<Scalar_T, Index_Set_T, Multivector_T>::
00067 classname() -> const std::string
00068 { return "clifford_algebra"; }
00069
00070 template< typename Scalar_T, typename Index_Set_T, typename Multivector_T>
00071 const
00072 Scalar_T
00073 clifford_algebra<Scalar_T, Index_Set_T, Multivector_T>::
00074 default_truncation = std::numeric_limits<Scalar_T>::epsilon();
00075
00076 template
00077 <
00078 template<typename, const index_t, const index_t, typename> class Multivector,
00079 template<typename, const index_t, const index_t, typename> class RHS,
00080 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00081 >
00082 inline
00083 auto
00084 operator!= (const Multivector<Scalar_T, LO, HI, Tune_P>& lhs, const RHS<Scalar_T, LO, HI, Tune_P>& rhs) ->
00085 bool
00086 { return !(lhs == rhs); }
00087
00088 template< template<typename, const index_t, const index_t, typename> class Multivector,
00089 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00090 inline
00091 auto
00092 operator!= (const Multivector<Scalar_T, LO, HI, Tune_P>& lhs, const Scalar_T& scr) -> bool
00093 { return !(lhs == scr); }
00094
00095 template< template<typename, const index_t, const index_t, typename> class Multivector,
00096 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00097 inline
00098 auto
00099 operator!= (const Scalar_T& scr, const Multivector<Scalar_T, LO, HI, Tune_P>& rhs) -> bool
00100 { return !(rhs == scr); }
00101
00102 template
00103 <
00104 template<typename, const index_t, const index_t, typename> class Multivector,
00105 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00106 >
00107 auto
00108 error_squared_tol(const Multivector<Scalar_T, LO, HI, Tune_P>& val) -> Scalar_T
00109 {
00110 using multivector_t = Multivector<Scalar_T, LO, HI, Tune_P>;
00111 static const auto scalar_eps = std::numeric_limits<Scalar_T>::epsilon();
00112 static const auto nbr_different_bits =
00113 std::numeric_limits<Scalar_T>::digits / Tune_P::denom_different_bits +
00114 Tune_P::extra_different_bits;
00115 static const auto abs_tol = scalar_eps *
00116 numeric_traits<Scalar_T>::pow(Scalar_T(2), nbr_different_bits);
00117 using framed_multi_t = typename multivector_t::framed_multi_t;
00118 const auto nbr_terms = double(framed_multi_t(val).truncated(scalar_eps).nbr_terms());
00119 return abs_tol * abs_tol * std::max(Scalar_T(nbr_terms), Scalar_T(1));
00120 }
00121
00122 template
00123 <
00124 template<typename, const index_t, const index_t, typename> class Multivector,
00125 template<typename, const index_t, const index_t, typename> class RHS,
00126 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00127 >
00128 inline
00129 auto
00130 error_squared(const Multivector<Scalar_T, LO, HI, Tune_P>& lhs,
00131 const RHS<Scalar_T, LO, HI, Tune_P>& rhs,

```

```

00136 const Scalar_T threshold) -> Scalar_T
00137 {
00138 const auto relative = norm(rhs) > threshold;
00139 const auto abs_norm_diff = norm(rhs-lhs);
00140 return (relative)
00141 ? abs_norm_diff/norm(rhs)
00142 : abs_norm_diff;
00143 }
00144
00146 template
00147 <
00148 template<typename, const index_t, const index_t, typename> class Multivector,
00149 template<typename, const index_t, const index_t, typename> class RHS,
00150 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00151 >
00152 inline
00153 auto
00154 approx_equal(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs,
00155 const RHS<Scalar_T,LO,HI,Tune_P>& rhs,
00156 const Scalar_T threshold,
00157 const Scalar_T tolerance) -> bool
00158 { return error_squared(lhs, rhs, threshold) < tolerance; }
00159
00161 template
00162 <
00163 template<typename, const index_t, const index_t, typename> class Multivector,
00164 template<typename, const index_t, const index_t, typename> class RHS,
00165 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00166 >
00167 inline
00168 auto
00169 approx_equal(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs,
00170 const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> bool
00171 {
00172 const Scalar_T rhs_tol = error_squared_tol(rhs);
00173 return approx_equal(lhs, rhs, rhs_tol, rhs_tol);
00174 }
00175
00177 template< template<typename, const index_t, const index_t, typename> class Multivector,
00178 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00179 inline
00180 auto
00181 operator+ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const
00182 Multivector<Scalar_T,LO,HI,Tune_P>
00183 {
00184 auto result = lhs;
00185 return result += scr;
00186 }
00187
00188 template< template<typename, const index_t, const index_t, typename> class Multivector,
00189 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00190 inline
00191 auto
00192 operator+ (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const
00193 Multivector<Scalar_T,LO,HI,Tune_P>
00194 {
00195 return rhs + scr;
00196 }
00197
00198 template
00199 <
00200 template<typename, const index_t, const index_t, typename> class Multivector,
00201 template<typename, const index_t, const index_t, typename> class RHS,
00202 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00203 >
00204 inline
00205 auto
00206 operator+ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
00207 const Multivector<Scalar_T,LO,HI,Tune_P>
00208 {
00209 auto result = lhs;
00210 return result += rhs;
00211 }
00212
00213 template< template<typename, const index_t, const index_t, typename> class Multivector,
00214 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00215 inline
00216 auto
00217 operator- (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const
00218 Multivector<Scalar_T,LO,HI,Tune_P>
00219 {
00220 auto result = lhs;
00221 return result -= scr;
00222 }
00223
00224 template< template<typename, const index_t, const index_t, typename> class Multivector,
00225 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >

```

```

00226 inline
00227 auto
00228 operator- (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>
00229 { return -rhs + scr; }
00230
00232 template
00233 <
00234 template<typename, const index_t, const index_t, typename> class Multivector,
00235 template<typename, const index_t, const index_t, typename> class RHS,
00236 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00237 >
00238 inline
00239 auto
00240 operator- (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>
00241 {
00242 auto result = lhs;
00243 return result -= rhs;
00244 }
00245
00247 template< template<typename, const index_t, const index_t, typename> class Multivector,
00248 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00249 inline
00250 auto
00251 operator* (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const
Multivector<Scalar_T,LO,HI,Tune_P>
00252 {
00253 auto result = lhs;
00254 return result *= scr;
00255 }
00256
00258 template< template<typename, const index_t, const index_t, typename> class Multivector,
00259 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00260 inline
00261 auto
00262 operator* (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>
00263 { // Note: this assumes that scalar commutes with multivector.
00264 // This excludes Clifford algebras over non-commuting rings.
00265 return rhs * scr;
00266 }
00267
00269 template
00270 <
00271 template<typename, const index_t, const index_t, typename> class Multivector,
00272 template<typename, const index_t, const index_t, typename> class RHS,
00273 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00274 >
00275 inline
00276 auto
00277 operator* (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>
00278 {
00279 using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00280 return lhs * multivector_t(rhs);
00281 }
00282
00284 template
00285 <
00286 template<typename, const index_t, const index_t, typename> class Multivector,
00287 template<typename, const index_t, const index_t, typename> class RHS,
00288 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00289 >
00290 inline
00291 auto
00292 operator^ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>
00293 {
00294 using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00295 return lhs ^ multivector_t(rhs);
00296 }
00297
00299 template
00300 <
00301 template<typename, const index_t, const index_t, typename> class Multivector,
00302 template<typename, const index_t, const index_t, typename> class RHS,
00303 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00304 >
00305 inline
00306 auto
00307 operator& (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>
00308 {
00309 using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00310 return lhs & multivector_t(rhs);
00311 }

```

```

00312
00314 template
00315 <
00316 template<typename, const index_t, const index_t, typename> class Multivector,
00317 template<typename, const index_t, const index_t, typename> class RHS,
00318 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00319 >
00320 inline
00321 auto
00322 operator% (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>
00323 {
00324 using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00325 return lhs % multivector_t(rhs);
00326 }
00327
00329 template
00330 <
00331 template<typename, const index_t, const index_t, typename> class Multivector,
00332 template<typename, const index_t, const index_t, typename> class RHS,
00333 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00334 >
00335 inline
00336 auto
00337 star (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
Scalar_T
00338 {
00339 using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00340 return star(lhs, multivector_t(rhs));
00341 }
00342
00344 template< template<typename, const index_t, const index_t, typename> class Multivector,
00345 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00346 inline
00347 auto
00348 operator/ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const
Multivector<Scalar_T,LO,HI,Tune_P>
00349 {
00350 auto result = lhs;
00351 return result /= scr;
00352 }
00353
00355 template< template<typename, const index_t, const index_t, typename> class Multivector,
00356 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00357 inline
00358 auto
00359 operator/ (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>
00360 {
00361 Multivector<Scalar_T,LO,HI,Tune_P> result = scr;
00362 return result /= rhs;
00363 }
00364
00366 template
00367 <
00368 template<typename, const index_t, const index_t, typename> class Multivector,
00369 template<typename, const index_t, const index_t, typename> class RHS,
00370 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00371 >
00372 inline
00373 auto
00374 operator/ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>
00375 {
00376 using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00377 return lhs / multivector_t(rhs);
00378 }
00379
00381 template
00382 <
00383 template<typename, const index_t, const index_t, typename> class Multivector,
00384 template<typename, const index_t, const index_t, typename> class RHS,
00385 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00386 >
00387 inline
00388 auto
00389 operator| (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) ->
const Multivector<Scalar_T,LO,HI,Tune_P>
00390 {
00391 using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00392 return lhs | multivector_t(rhs);
00393 }
00394
00396 template< template<typename, const index_t, const index_t, typename> class Multivector,
00397 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00398 inline
00399 auto

```

```

00400 inv(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00401 { return val.inv(); }
00402
00404 template< template<typename, const index_t, const index_t, typename> class Multivector,
00405 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00406 auto
00407 pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, int rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>
00408 {
00409 using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00410 if (lhs == Scalar_T(0))
00411 {
00412 using traits_t = numeric_traits<Scalar_T>;
00413 return
00414 (rhs < 0)
00415 ? traits_t::NaN()
00416 : (rhs == 0)
00417 ? Scalar_T(1)
00418 : Scalar_T(0);
00419 }
00420 auto result = multivector_t(Scalar_T(1));
00421 auto power =
00422 (rhs < 0)
00423 ? lhs.inv()
00424 : lhs;
00425 for (auto
00426 k = std::abs(rhs);
00427 k != 0;
00428 k /= 2)
00429 {
00430 if (k % 2)
00431 result *= power;
00432 power *= power;
00433 }
00434 return result;
00435 }
00436
00438 template
00439 <
00440 template<typename, const index_t, const index_t, typename> class Multivector,
00441 template<typename, const index_t, const index_t, typename> class RHS,
00442 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00443 >
00444 inline
00445 auto
00446 pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const RHS<Scalar_T,LO,HI,Tune_P>& rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>
00447 {
00448 using traits_t = numeric_traits<Scalar_T>;
00449
00450 if (lhs == Scalar_T(0))
00451 {
00452 const Scalar_T m = rhs.scalar();
00453 if (rhs == m)
00454 return
00455 (m < 0)
00456 ? traits_t::NaN()
00457 : (m == 0)
00458 ? Scalar_T(1)
00459 : Scalar_T(0);
00460 else
00461 return Scalar_T(0);
00462 }
00463 return exp(log(lhs) * rhs);
00464 }
00465
00467 template< template<typename, const index_t, const index_t, typename> class Multivector,
00468 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00469 auto
00470 outer_pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, int rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>
00471 { return lhs.outer_pow(rhs); }
00472
00474 template< template<typename, const index_t, const index_t, typename> class Multivector,
00475 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00476 inline
00477 auto
00478 scalar(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
00479 { return val.scalar(); }
00480
00482 template< template<typename, const index_t, const index_t, typename> class Multivector,
00483 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00484 inline
00485 auto
00486 real(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
00487 { return val.scalar(); }
00488

```

```

00490 template
00491 <
00492 template<typename, const index_t, const index_t, typename> class Multivector,
00493 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00494 >
00495 inline
00496 auto
00497 imag(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
00498 { return Scalar_T(0); }
00499
00500 template< template<typename, const index_t, const index_t, typename> class Multivector,
00501 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00502 inline
00503 auto
00504 pure(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00505 { return val - val.scalar(); }
00506
00507 template< template<typename, const index_t, const index_t, typename> class Multivector,
00508 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00509 inline
00510 auto
00511 even(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00512 { return val.even(); }
00513
00514 template< template<typename, const index_t, const index_t, typename> class Multivector,
00515 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00516 inline
00517 auto
00518 odd(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00519 { return val.odd(); }
00520
00521 template< template<typename, const index_t, const index_t, typename> class Multivector,
00522 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00523 inline
00524 auto
00525 vector_part(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const std::vector<Scalar_T>
00526 { return val.vector_part(); }
00527
00528 template< template<typename, const index_t, const index_t, typename> class Multivector,
00529 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00530 inline
00531 auto
00532 involute(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00533 { return val.involute(); }
00534
00535 template< template<typename, const index_t, const index_t, typename> class Multivector,
00536 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00537 inline
00538 auto
00539 reverse(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00540 { return val.reverse(); }
00541
00542 template< template<typename, const index_t, const index_t, typename> class Multivector,
00543 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00544 inline
00545 auto
00546 conj(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00547 { return val.conj(); }
00548
00549 template< template<typename, const index_t, const index_t, typename> class Multivector,
00550 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00551 inline
00552 auto
00553 quad(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
00554 { return val.quad(); }
00555
00556 template< template<typename, const index_t, const index_t, typename> class Multivector,
00557 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00558 inline
00559 auto
00560 norm(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
00561 { return val.norm(); }
00562
00563 template< template<typename, const index_t, const index_t, typename> class Multivector,
00564 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00565 inline
00566 auto
00567 abs(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
00568 { return numeric_traits<Scalar_T>::sqrt(val.norm()); }
00569
00570 template< template<typename, const index_t, const index_t, typename> class Multivector,
00571 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00572 inline
00573 auto
00574 max_abs(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
00575 { return val.max_abs(); }
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587

```

```

00589 template< template<typename, const index_t, const index_t, typename> class Multivector,
00590 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00591 auto
00592 complexifier(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const
Multivector<Scalar_T,LO,HI,Tune_P>
00593 {
00594 using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00595 using traits_t = numeric_traits<Scalar_T>;
00596
00597 auto frm = val.frame();
00598 using array_t = std::array<index_t, 4>;
00599 auto incp = array_t{0, 2, 1, 0};
00600 auto incq = array_t{1, 0, 0, 0};
00601 auto bott = pos_mod((frm.count_pos() - frm.count_neg()), 4);
00602 for (auto
00603 k = index_t(0);
00604 k != incp[bott];
00605 k++)
00606 for (auto
00607 idx = index_t(1);
00608 idx != HI+1;
00609 ++idx)
00610 if (!frm[idx])
00611 {
00612 frm.set(idx);
00613 break;
00614 }
00615 for (auto
00616 k = index_t(0);
00617 k != incq[bott];
00618 k++)
00619 for (auto
00620 idx = index_t(-1);
00621 idx != LO-1;
00622 --idx)
00623 if (!frm[idx])
00624 {
00625 frm.set(idx);
00626 break;
00627 }
00628 auto new_bott = pos_mod(frm.count_pos() - frm.count_neg(), 4);
00629
00630 if ((incp[new_bott] == 0) && (incq[new_bott] == 0))
00631 return multivector_t(frm, Scalar_T(1));
00632 else
00633 // Return IEEE NaN or -Inf
00634 return traits_t::NaN();
00635 }
00636
00637 template< template<typename, const index_t, const index_t, typename> class Multivector,
00638 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00639 inline
00640 auto
00641 elliptic(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00642 { return complexifier(val); }
00643
00644 template< template<typename, const index_t, const index_t, typename> class Multivector,
00645 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00646 inline
00647 static
00648 void
00649 check_complex(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00650 const Multivector<Scalar_T,LO,HI,Tune_P>& i, const bool prechecked = false)
00651 {
00652 if (!prechecked)
00653 {
00654 using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00655 using index_set_t = typename multivector_t::index_set_t;
00656 using error_t = typename multivector_t::error_t;
00657
00658 const auto i_frame = i.frame();
00659 // We need i to be a complexifier whose frame is large enough to represent val
00660 if (complexifier(i) != i ||
00661 (val.frame() | i_frame) != i_frame ||
00662 complexifier(val).frame().count() > i_frame.count())
00663 throw error_t("check_complex(val, i): i is not a valid complexifier for val");
00664 }
00665 }
00666
00667 template< template<typename, const index_t, const index_t, typename> class Multivector,
00668 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00669 inline
00670 auto
00671 sqrt(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00672 bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00673 { return sqrt(val, i, prechecked); }
00674
00675
00676
00677

```



```

00679 template< template<typename, const index_t, const index_t, typename> class Multivector,
00680 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00681 inline
00682 auto
00683 sqrt(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00684 { return sqrt(val, complexifier(val), true); }
00685
00686 template< template<typename, const index_t, const index_t, typename> class Multivector,
00687 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00688 auto
00689 clifford_exp(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const
00690 Multivector<Scalar_T,LO,HI,Tune_P>
00691 {
00692 // Scaling and squaring Pade' approximation of matrix exponential
00693 // Reference: [GL], Section 11.3, p572-576
00694 // Reference: [H]
00695
00696 using traits_t = numeric_traits<Scalar_T>;
00697
00698 const auto scalar_val = val.scalar();
00699 const auto scalar_exp = traits_t::exp(scalar_val);
00700 if (traits_t::isNaN_or_isInf(scalar_exp))
00701 return traits_t::NaN();
00702 if (val == scalar_val)
00703 return scalar_exp;
00704
00705 using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00706 auto A = val - scalar_val;
00707 const auto pure_scale2 = A.norm();
00708
00709 if (traits_t::isNaN_or_isInf(pure_scale2))
00710 return traits_t::NaN();
00711 if (pure_scale2 == Scalar_T(0))
00712 return scalar_exp;
00713
00714 const auto ilog2_scale =
00715 std::max(0, traits_t::to_int(ceil((log2(pure_scale2) +
00716 Scalar_T(A.frame().count())/Scalar_T(2))) - 3));
00717 const auto i_scale = traits_t::pow(Scalar_T(2), ilog2_scale);
00718 if (traits_t::isNaN_or_isInf(i_scale))
00719 return traits_t::NaN();
00720
00721 A /= i_scale;
00722 multivector_t pure_exp;
00723 {
00724 using limits_t = std::numeric_limits<Scalar_T>;
00725 const auto nbr_even_powers = 2*(limits_t::digits / 32) + 4;
00726 using nbr_t = decltype(nbr_even_powers);
00727
00728 // Create an array of coefficients
00729 const auto max_power = 2*nbr_even_powers + 1;
00730 static std::array<Scalar_T, max_power+1> c;
00731 if (c[0] != Scalar_T(1))
00732 {
00733 c[0] = Scalar_T(1);
00734 for (auto
00735 k = decltype(max_power)(0);
00736 k != max_power;
00737 ++k)
00738 c[k+1] = c[k]*(max_power-k) / ((2*max_power-k)*(k+1));
00739 }
00740
00741 // Create an array of even powers
00742 std::array<multivector_t, nbr_even_powers> AA;
00743 AA[0] = A * A;
00744 AA[1] = AA[0] * AA[0];
00745 for (auto
00746 k = nbr_t(2);
00747 k != nbr_even_powers;
00748 ++k)
00749 AA[k] = AA[k-2] * AA[1];
00750
00751 // Use compensated summation to calculate U and AV
00752 auto residual = multivector_t();
00753 auto U = multivector_t(c[0]);
00754 for (auto
00755 k = nbr_t(0);
00756 k != nbr_even_powers;
00757 ++k)
00758 {
00759 const auto& term = AA[k]*c[2*k + 2] - residual;
00760 const auto& sum = U + term;
00761 residual = (sum - U) - term;
00762 U = sum;
00763 }
00764 residual = multivector_t();
00765 auto AV = multivector_t(c[1]);

```

```

00765 for (auto
00766 k = nbr_t(0);
00767 k != nbr_even_powers;
00768 ++k)
00769 {
00770 const auto& term = AA[k]*c[2*k + 3] - residual;
00771 const auto& sum = AV + term;
00772 residual = (sum - AV) - term;
00773 AV = sum;
00774 }
00775 AV *= A;
00776 pure_exp = (U+AV) / (U-AV);
00777 }
00778 for (auto
00779 k = decltype(ilog2_scale)(0);
00780 k != ilog2_scale;
00781 ++k)
00782 pure_exp *= pure_exp;
00783 return pure_exp * scalar_exp;
00784 }
00785
00786 template< template<typename, const index_t, const index_t, typename> class Multivector,
00787 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00788 inline
00789 auto
00790 log(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool
00791 prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00792 { return log(val, i, prechecked); }
00793
00794 template< template<typename, const index_t, const index_t, typename> class Multivector,
00795 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00796 inline
00797 auto
00798 log(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00799 { return log(val, complexifier(val), true); }
00800
00801 template< template<typename, const index_t, const index_t, typename> class Multivector,
00802 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00803 inline
00804 auto
00805 cosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00806 {
00807 using traits_t = numeric_traits<Scalar_T>;
00808 if (val.isnan())
00809 return traits_t::NaN();
00810 const auto& s = val.scalar();
00811 if (val == s)
00812 return traits_t::cosh(s);
00813 return (exp(val)+exp(-val)) / Scalar_T(2);
00814 }
00815
00816 // Reference: [AS], Section 4.6, p86-89
00817 template< template<typename, const index_t, const index_t, typename> class Multivector,
00818 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00819 inline
00820 auto
00821 acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00822 bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00823 {
00824 using traits_t = numeric_traits<Scalar_T>;
00825 check_complex(val, i, prechecked);
00826 if (val.isnan())
00827 return traits_t::NaN();
00828 const auto radical = sqrt(val*val - Scalar_T(1), i, true);
00829 return (norm(val + radical) >= norm(val))
00830 ? log(val + radical, i, true)
00831 : -log(val - radical, i, true);
00832 }
00833
00834 // Reference: [AS], Section 4.6, p86-89
00835 template< template<typename, const index_t, const index_t, typename> class Multivector,
00836 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00837 inline
00838 auto
00839 acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00840 { return acosh(val, complexifier(val), true); }
00841
00842 template< template<typename, const index_t, const index_t, typename> class Multivector,
00843 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00844 auto
00845 cos(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool
00846 prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00847 {
00848 using traits_t = numeric_traits<Scalar_T>;
00849 if (val.isnan())

```

```

00855 return traits_t::NaN();
00856
00857 const auto& s = val.scalar();
00858 if (val == s)
00859 return traits_t::cos(s);
00860
00861 check_complex(val, i, prechecked);
00862
00863 static const auto& twopi = Scalar_T(2) * traits_t::pi();
00864 const auto& z = i *
00865 (val - s + traits_t::fmod(s, twopi));
00866 return (exp(z)+exp(-z)) / Scalar_T(2);
00867 }
00868
00870 template< template<typename, const index_t, const index_t, typename> class Multivector,
00871 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00872 inline
00873 auto
00874 cos(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00875 { return cos(val, complexifier(val), true); }
00876
00878 // Reference: [AS], Section 4.4, p79-83
00879 template< template<typename, const index_t, const index_t, typename> class Multivector,
00880 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00881 inline
00882 auto
00883 acos(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00884 bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00885 {
00886 using traits_t = numeric_traits<Scalar_T>;
00887 if (val.isnan())
00888 return traits_t::NaN();
00889
00890 const auto& s = val.scalar();
00891 if (val == s && traits_t::abs(s) <= Scalar_T(1))
00892 return traits_t::acos(s);
00893
00894 check_complex(val, i, prechecked);
00895 return i * acosh(val, i, true);
00896 }
00897
00898 // Reference: [AS], Section 4.4, p79-83
00899 template< template<typename, const index_t, const index_t, typename> class Multivector,
00900 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00901 inline
00902 auto
00903 acos(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00904 { return acos(val, complexifier(val), true); }
00905
00907 template< template<typename, const index_t, const index_t, typename> class Multivector,
00908 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00909 inline
00910 auto
00911 sinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00912 {
00913 using traits_t = numeric_traits<Scalar_T>;
00914 if (val.isnan())
00915 return traits_t::NaN();
00916
00917 const auto& s = val.scalar();
00918 if (val == s)
00919 return traits_t::sinh(s);
00920
00921 return (exp(val)-exp(-val)) / Scalar_T(2);
00922 }
00923
00925 // Reference: [AS], Section 4.6, p86-89
00926 template< template<typename, const index_t, const index_t, typename> class Multivector,
00927 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00928 inline
00929 auto
00930 asinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00931 bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00932 {
00933 using traits_t = numeric_traits<Scalar_T>;
00934 check_complex(val, i, prechecked);
00935 if (val.isnan())
00936 return traits_t::NaN();
00937
00938 const auto radical = sqrt(val*val + Scalar_T(1), i, true);
00939 return (norm(val + radical) >= norm(val))
00940 ? log(val + radical, i, true)
00941 : -log(-val + radical, i, true);
00942 }
00943
00944 // Reference: [AS], Section 4.6, p86-89
00945 template< template<typename, const index_t, const index_t, typename> class Multivector,

```

```

00946 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00947 inline
00948 auto
00949 asinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00950 { return asinh(val, complexifier(val), true); }
00951
00952 template< template<typename, const index_t, const index_t, typename> class Multivector,
00953 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00954 auto
00955 sin(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool
00956 prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00957 {
00958 using traits_t = numeric_traits<Scalar_T>;
00959 if (val.isnan())
00960 return traits_t::NaN();
00961
00962 const auto& s = val.scalar();
00963 if (val == s)
00964 return traits_t::sin(s);
00965
00966 check_complex(val, i, prechecked);
00967
00968 static const auto& twopi = Scalar_T(2) * traits_t::pi();
00969 const auto& z = i *
00970 (val - s + traits_t::fmod(s, twopi));
00971 return i * (exp(-z)-exp(z)) / Scalar_T(2);
00972 }
00973
00974 template< template<typename, const index_t, const index_t, typename> class Multivector,
00975 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00976 inline
00977 auto
00978 sin(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00979 { return sin(val, complexifier(val), true); }
00980
00981 // Reference: [AS], Section 4.4, p79-83
00982 template< template<typename, const index_t, const index_t, typename> class Multivector,
00983 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00984 inline
00985 auto
00986 asin(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00987 bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00988 {
00989 using traits_t = numeric_traits<Scalar_T>;
00990 if (val.isnan())
00991 return traits_t::NaN();
00992
00993 const auto& s = val.scalar();
00994 if (val == s && traits_t::abs(s) <= Scalar_T(1))
00995 return traits_t::asin(s);
00996
00997 check_complex(val, i, prechecked);
00998 return -i * asinh(i * val, i, true);
00999 }
01000
01001 // Reference: [AS], Section 4.4, p79-83
01002 template< template<typename, const index_t, const index_t, typename> class Multivector,
01003 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01004 inline
01005 auto
01006 asin(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
01007 { return asin(val, complexifier(val), true); }
01008
01009 template< template<typename, const index_t, const index_t, typename> class Multivector,
01010 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01011 inline
01012 auto
01013 tanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
01014 {
01015 using traits_t = numeric_traits<Scalar_T>;
01016 if (val.isnan())
01017 return traits_t::NaN();
01018
01019 const auto& s = val.scalar();
01020 if (val == s)
01021 return traits_t::tanh(s);
01022
01023 return sinh(val) / cosh(val);
01024 }
01025
01026 // Reference: [AS], Section 4.6, p86-89
01027 template< template<typename, const index_t, const index_t, typename> class Multivector,
01028 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01029 inline
01030 auto
01031 atanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
01032 bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>

```

```

01036 {
01037 using traits_t = numeric_traits<Scalar_T>;
01038 check_complex(val, i, prechecked);
01039 return val.isnan()
01040 ? traits_t::NaN()
01041 : (norm(val + Scalar_T(1)) > norm(val - Scalar_T(1)))
01042 ? (log(val + Scalar_T(1), i, true) - log(-val + Scalar_T(1), i, true)) / Scalar_T(2)
01043 : log((val + Scalar_T(1)) / (-val + Scalar_T(1)), i, true) / Scalar_T(2);
01044 }
01045
01046 // Reference: [AS], Section 4.6, p86-89
01047 template< template<typename, const index_t, const index_t, typename> class Multivector,
01048 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01049 inline
01050 auto
01051 atanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
01052 { return atanh(val, complexifier(val), true); }
01053
01054 template< template<typename, const index_t, const index_t, typename> class Multivector,
01055 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01056 inline
01057 auto
01058 tan(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool
01059 prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
01060 {
01061 {
01062 using traits_t = numeric_traits<Scalar_T>;
01063 if (val.isnan())
01064 return traits_t::NaN();
01065
01066 const auto& s = val.scalar();
01067 if (val == s)
01068 return traits_t::tan(s);
01069
01070 check_complex(val, i, prechecked);
01071 return sin(val, i, true) / cos(val, i, true);
01072 }
01073
01074 template< template<typename, const index_t, const index_t, typename> class Multivector,
01075 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01076 inline
01077 auto
01078 tan(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
01079 { return tan(val, complexifier(val), true); }
01080
01081 // Reference: [AS], Section 4.4, p79-83
01082 template< template<typename, const index_t, const index_t, typename> class Multivector,
01083 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01084 inline
01085 auto
01086 atan(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
01087 bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
01088 {
01089 {
01090 using traits_t = numeric_traits<Scalar_T>;
01091 if (val.isnan())
01092 return traits_t::NaN();
01093
01094 const auto& s = val.scalar();
01095 if (val == s)
01096 return traits_t::atan(s);
01097
01098 check_complex(val, i, prechecked);
01099 return -i * atanh(i * val, i, true);
01100 }
01101
01102 // Reference: [AS], Section 4.4, p79-83
01103 template< template<typename, const index_t, const index_t, typename> class Multivector,
01104 typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01105 inline
01106 auto
01107 atan(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
01108 { return atan(val, complexifier(val), true); }
01109
01110 }
01111 }
01112 #endif // _GLUCAT_CLIFFORD_ALGEBRA_IMP_H

```

## 7.5 glucat/errors.h File Reference

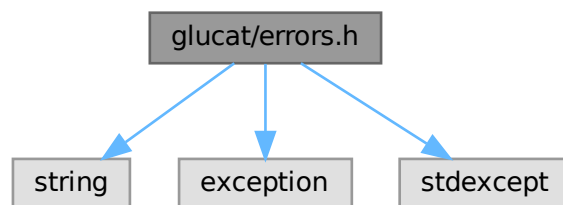
```

#include <string>
#include <exception>

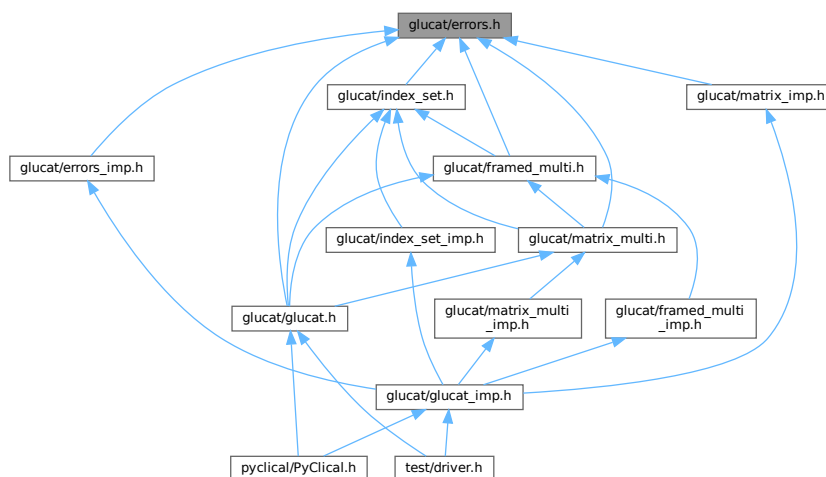
```

```
#include <stdexcept>
```

Include dependency graph for errors.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [glucat::glucat\\_error](#)  
*Abstract exception class.*
- class [glucat::error< Class\\_T >](#)  
*Specific exception class.*

## Namespaces

- namespace [glucat](#)

## 7.6 errors.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_ERRORS_H
00002 #define _GLUCAT_ERRORS_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 errors.h : Declare error classes and functions
00006 -----
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2012 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 #include <string>
00035 #include <exception>
00036 #include <stdexcept>
00037
00038 namespace glucat
00039 {
00040 class glucat_error : public std::logic_error
00041 {
00042 public:
00043 glucat_error(const std::string& context, const std::string& msg)
00044 : logic_error(msg), name(context)
00045 { }
00046 ~glucat_error() noexcept override = default;
00047 virtual auto heading() const noexcept -> const std::string =0;
00048 virtual auto classname() const noexcept -> const std::string =0;
00049 virtual void print_error_msg() const =0;
00050 std::string name;
00051 };
00052
00053 template< class Class_T >
00054 class error : public glucat_error
00055 {
00056 public:
00057 error(const std::string& msg);
00058 error(const std::string& context, const std::string& msg);
00059 auto heading() const noexcept -> const std::string override;
00060 auto classname() const noexcept -> const std::string override;
00061 void print_error_msg() const override;
00062 };
00063
00064 #endif // _GLUCAT_ERRORS_H

```

## 7.7 glucat/errors\_imp.h File Reference

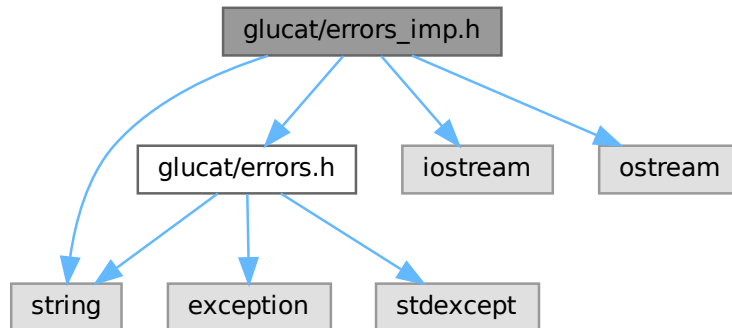
```

#include "glucat/errors.h"
#include <string>
#include <iostream>

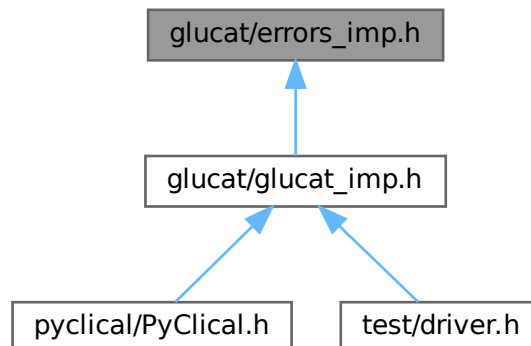
```

```
#include <ostream>
```

Include dependency graph for errors\_imp.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace `glucat`

## 7.8 errors\_imp.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_ERRORS_IMP_H
00002 #define _GLUCAT_ERRORS_IMP_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 errors_imp.h : Define error functions
00006 -----

```



```

00007 begin : Sun 2001-12-20
00008 copyright : (C) 2001-2007 by Paul C. Leopardi
00009 *****
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 #include "glucat/errors.h"
00035
00036 #include <string>
00037 #include <iostream>
00038 #include <ostream>
00039
00040 namespace glucat
00041 {
00042 template< class Class_T >
00043 error<Class_T>::
00044 error(const std::string& msg)
00045 : glucat_error(Class_T::classname(), msg)
00046 { }
00047
00048 template< class Class_T >
00049 error<Class_T>::
00050 error(const std::string& context, const std::string& msg)
00051 : glucat_error(context, msg)
00052 { }
00053
00054 template< class Class_T >
00055 auto
00056 error<Class_T>::
00057 heading() const noexcept -> const std::string
00058 { return "Error in glucat: "; }
00059
00060 template< class Class_T >
00061 auto
00062 error<Class_T>::
00063 classname() const noexcept -> const std::string
00064 { return name; }
00065
00066 template< class Class_T >
00067 void
00068 error<Class_T>::
00069 print_error_msg() const
00070 { std::cerr << heading() << classname() << std::endl << what() << std::endl; }
00071 }
00072 #endif // _GLUCAT_ERRORS_IMP_H
00073

```

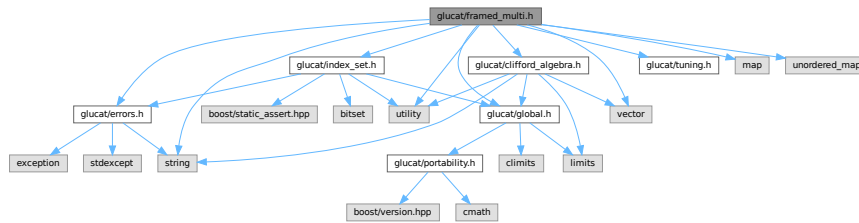
## 7.9 glucat/framed\_multi.h File Reference

```

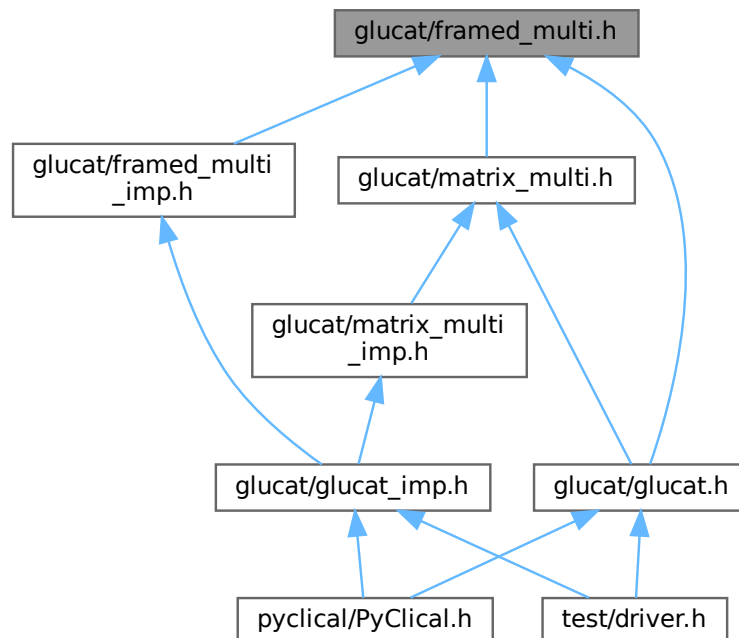
#include "glucat/global.h"
#include "glucat/errors.h"
#include "glucat/index_set.h"
#include "glucat/clifford_algebra.h"
#include "glucat/tuning.h"
#include <string>
#include <utility>
#include <map>

```

```
#include <unordered_map>
#include <vector>
Include dependency graph for framed_multi.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [glucat::index\\_set\\_hash< LO, HI >](#)
- class [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >](#)  
*A framed\_multi<Scalar\_T,LO,HI,Tune\_P> is a framed approximation to a multivector.*
- class [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::hash\\_size\\_t](#)
- class [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::var\\_term](#)  
*Variable term.*
- struct [std::numeric\\_limits< glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P > >](#)  
*Numeric limits for framed\_multi inherit limits for the corresponding scalar type.*

## Namespaces

- namespace [glucat](#)
- namespace [std](#)

## Functions

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator\\*](#) (const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >  
*Geometric product.*
- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator^](#) (const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >  
*Outer product.*
- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator&](#) (const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >  
*Inner product.*
- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator%](#) (const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >  
*Left contraction.*
- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::star](#) (const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> Scalar\_T  
*Hestenes scalar product.*
- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator/](#) (const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >  
*Geometric quotient.*
- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator|](#) (const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >  
*Transformation via twisted adjoint action.*
- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator>>](#) (std::istream &s, [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val) -> std::istream &  
*Read multivector from input.*
- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator<<](#) (std::ostream &os, const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val) -> std::ostream &  
*Write multivector to output.*
- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI>  
auto [glucat::operator<<](#) (std::ostream &os, const std::pair< const [index\\_set](#)< LO, HI >, Scalar\_T > &term) -> std::ostream &  
*Write term to output.*
- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::exp](#) (const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val) -> const [framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >  
*Exponential of multivector.*
- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI>  
static auto [glucat::crd\\_of\\_mult](#) (const std::pair< const [index\\_set](#)< LO, HI >, Scalar\_T > &lhs, const std::pair< const [index\\_set](#)< LO, HI >, Scalar\_T > &rhs) -> Scalar\_T

*Coordinate of product of terms.*

- `template<typename Scalar_T, const index_t LO, const index_t HI>`  
`auto glucat::operator* (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std::pair< const index_set< LO, HI >, Scalar_T > &rhs) -> const std::pair< const index_set< LO, HI >, Scalar_T >`

*Product of terms.*

- `template<typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::sqrt (const framed_multi< Scalar_T, LO, HI, Tune_P > &val, const framed_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`

*Square root of multivector with specified complexifier.*

- `template<typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::log (const framed_multi< Scalar_T, LO, HI, Tune_P > &val, const framed_multi< Scalar_T, LO, HI, Tune_P > &i, bool prechecked) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`

*Natural logarithm of multivector with specified complexifier.*

## 7.10 framed\_multi.h

[Go to the documentation of this file.](#)

```
00001 #ifndef _GLUCAT_FRAMED_MULTI_H
00002 #define _GLUCAT_FRAMED_MULTI_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 framed_multi.h : Declare a class for the framed representation of a multivector
00006 -----
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2021 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 #include "glucat/global.h"
00035 #include "glucat/errors.h"
00036 #include "glucat/index_set.h"
00037 #include "glucat/clifford_algebra.h"
00038 #include "glucat/tuning.h"
00039
00040 #if defined(_GLUCAT_USE_BOOST_POOL_ALLOC)
00041 // Use the Boost pool allocator
00042 #include <boost/pool/pool_fwd.hpp>
00043 #endif
00044
00045 #include <string>
00046 #include <utility>
00047 #include <map>
00048 #include <unordered_map>
00049 #include <vector>
00050
00051 namespace glucat
00052 {
00053 // Forward declarations for friends
00054
00055 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00056 class framed_multi; // forward
00057 }
```

```

00058 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00059 class matrix_multi; // forward
00060
00062 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00063 auto
00064 operator* (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00065
00067 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00068 auto
00069 operator^ (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00070
00072 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00073 auto
00074 operator& (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00075
00077 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00078 auto
00079 operator% (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00080
00082 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00083 auto
00084 star(const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const framed_multi<Scalar_T,LO,HI,Tune_P>& rhs)
-> Scalar_T;
00085
00087 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00088 auto
00089 operator/ (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00090
00092 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00093 auto
00094 operator| (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00095
00097 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00098 auto
00099 operator> (std::istream& s, framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::istream&;
00100
00102 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00103 auto
00104 operator< (std::ostream& os, const framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::ostream&;
00105
00107 template< typename Scalar_T, const index_t LO, const index_t HI >
00108 auto
00109 operator<< (std::ostream& os, const std::pair< const index_set<LO,HI>, Scalar_T >& term) ->
std::ostream&;
00110
00112 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00113 auto
00114 exp(const framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00115
00116 template< const index_t LO, const index_t HI>
00117 class index_set_hash
00118 {
00119 public:
00120 using index_set_t = index_set<LO, HI>;
00121 inline auto operator()(index_set_t val) const -> size_t { return val.hash_fn(); }
00122 };
00123
00125 template< typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI,
typename Tune_P = tuning<> >
00126 class framed_multi :
00127 public clifford_algebra< Scalar_T, index_set<LO,HI>, framed_multi<Scalar_T,LO,HI,Tune_P> >,
00128 private std::unordered_map< index_set<LO,HI>, Scalar_T, index_set_hash<LO,HI> >
00129 {
00130 public:
00131 using multivector_t = framed_multi;
00132 using framed_multi_t = multivector_t;
00133 using scalar_t = Scalar_T;
00134 using tune_p = Tune_P;
00135 using index_set_t = index_set<LO, HI>;
00136 using term_t = std::pair<const index_set_t, Scalar_T>;
00137 using vector_t = std::vector<Scalar_T>;
00138 using error_t = error<multivector_t>;
00139 using matrix_multi_t = matrix_multi<Scalar_T,LO,HI,Tune_P >;
00140 template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
00141 friend class matrix_multi;
00142 template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
00143 friend class framed_multi;
00144 private:

```

```

00146 class var_term; // forward
00147 using var_term_t = class var_term;
00148 using matrix_t = typename matrix_multi_t::matrix_t;
00149 using sorted_map_t = std::map< index_set_t, Scalar_T, std::less<const index_set_t> >;
00150 using map_t = std::unordered_map<index_set_t, Scalar_T, index_set_hash<LO, HI>;
00151
00152 class hash_size_t
00153 {
00154 public:
00155 hash_size_t(size_t hash_size)
00156 : n(hash_size)
00157 { };
00158 auto operator()() const -> size_t
00159 { return n; }
00160 private:
00161 size_t n;
00162 };
00163
00164 using framed_pair_t = std::pair<const multivector_t, const multivector_t>;
00165 using size_type = typename map_t::size_type;
00166 using iterator = typename map_t::iterator;
00167 using const_iterator = typename map_t::const_iterator;
00168
00169 public:
00170 static auto classname() -> const std::string;
00171 ~framed_multi() override = default;
00172 framed_multi();
00173
00174 private:
00175 framed_multi(const hash_size_t& hash_size);
00176
00177 public:
00178 template< typename Other_Scalar_T >
00179 framed_multi(const framed_multi<Other_Scalar_T, LO, HI, Tune_P>& val);
00180 template< typename Other_Scalar_T >
00181 framed_multi(const framed_multi<Other_Scalar_T, LO, HI, Tune_P>& val,
00182 const index_set_t frm, const bool prechecked = false);
00183 framed_multi(const framed_multi_t& val,
00184 const index_set_t frm, const bool prechecked = false);
00185 framed_multi(const index_set_t ist, const Scalar_T& crd = Scalar_T(1));
00186 framed_multi(const index_set_t ist, const Scalar_T& crd,
00187 const index_set_t frm, const bool prechecked = false);
00188 framed_multi(const Scalar_T& scr, const index_set_t frm = index_set_t());
00189 framed_multi(const int scr, const index_set_t frm = index_set_t());
00190 framed_multi(const vector_t& vec,
00191 const index_set_t frm, const bool prechecked = false);
00192 framed_multi(const std::string& str);
00193 framed_multi(const std::string& str,
00194 const index_set_t frm, const bool prechecked = false);
00195 framed_multi(const char* str)
00196 { *this = framed_multi(std::string(str)); };
00197 framed_multi(const char* str,
00198 const index_set_t frm, const bool prechecked = false)
00199 { *this = framed_multi(std::string(str), frm, prechecked); };
00200 template< typename Other_Scalar_T >
00201 framed_multi(const matrix_multi<Other_Scalar_T, LO, HI, Tune_P >& val);
00202 template< typename Other_Scalar_T >
00203 auto fast_matrix_multi(const index_set_t frm) const -> const
00204 matrix_multi<Other_Scalar_T, LO, HI, Tune_P >;
00205 auto fast_framed_multi() const -> const framed_multi_t;
00206
00207 _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS
00208
00209 auto nbr_terms() const -> unsigned long;
00210
00211 static auto random(const index_set_t frm, Scalar_T fill = Scalar_T(1)) -> const multivector_t;
00212
00213 // Friend declarations
00214
00215 friend auto
00216 operator* <>(const multivector_t& lhs, const multivector_t& rhs) -> const multivector_t;
00217 friend auto
00218 operator^ <>(const multivector_t& lhs, const multivector_t& rhs) -> const multivector_t;
00219 friend auto
00220 operator& <>(const multivector_t& lhs, const multivector_t& rhs) -> const multivector_t;
00221 friend auto
00222 operator% <>(const multivector_t& lhs, const multivector_t& rhs) -> const multivector_t;
00223 friend auto
00224 star <>(const multivector_t& lhs, const multivector_t& rhs) -> Scalar_T;
00225 friend auto
00226 operator/ <>(const multivector_t& lhs, const multivector_t& rhs) -> const multivector_t;
00227 friend auto
00228 operator| <>(const multivector_t& lhs, const multivector_t& rhs) -> const multivector_t;
00229
00230 friend auto
00231 operator<< <>(std::istream& s, multivector_t& val) -> std::istream&;
00232 friend auto
00233 operator<< <>(std::ostream& os, const multivector_t& val) -> std::ostream&;

```

```

00253 friend auto
00254 operator<< (<std::ostream& os, const term_t& term) -> std::ostream&;
00255
00256 friend auto
00257 exp<>(const multivector_t& val) -> const multivector_t;
00258
00260 auto operator+= (const term_t& term) -> multivector_t&;
00261
00262 private:
00264 auto fold(const index_set_t frm) const -> multivector_t;
00266 auto unfold(const index_set_t frm) const -> multivector_t;
00268 auto centre_pm4_qp4(index_t& p, index_t& q) -> multivector_t&;
00270 auto centre_pp4_qm4(index_t& p, index_t& q) -> multivector_t&;
00272 auto centre_qp1_pm1(index_t& p, index_t& q) -> multivector_t&;
00274 auto divide(const index_set_t ist) const -> const framed_pair_t;
00276 auto fast(const index_t level, const bool odd) const -> const matrix_t;
00277
00279 class var_term :
00280 public std::pair<index_set<LO,HI>, Scalar_T>
00281 {
00282 public:
00283 using var_pair_t = std::pair<index_set<LO, HI>, Scalar_T>;
00284
00286 static auto classname() -> const std::string
00287 { return "var_term"; };
00289 ~var_term() = default;
00291 var_term()
00292 : var_pair_t(index_set_t(), Scalar_T(1))
00293 { };
00295 var_term(const index_set_t ist, const Scalar_T& crd = Scalar_T(1))
00296 : var_pair_t(ist, crd)
00297 { };
00299 auto operator*= (const term_t& rhs) -> var_term_t&
00300 {
00301 this->second *= rhs.second * this->first.sign_of_mult(rhs.first);
00302 this->first ^= rhs.first;
00303 return *this;
00304 }
00305 };
00306 };
00307
00308 // Non-members
00309
00311 template< typename Scalar_T, const index_t LO, const index_t HI >
00312 inline
00313 static
00314 auto
00315 crd_of_mult(const std::pair<const index_set<LO,HI>, Scalar_T>& lhs,
00316 const std::pair<const index_set<LO,HI>, Scalar_T>& rhs) -> Scalar_T;
00317
00319 template< typename Scalar_T, const index_t LO, const index_t HI >
00320 auto
00321 operator*
00322 (const std::pair<const index_set<LO,HI>, Scalar_T>& lhs,
00323 const std::pair<const index_set<LO,HI>, Scalar_T>& rhs) -> const std::pair<const index_set<LO,HI>,
00324 Scalar_T>;
00326 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00327 auto
00328 sqrt(const framed_multi<Scalar_T,LO,HI,Tune_P>& val, const framed_multi<Scalar_T,LO,HI,Tune_P>& i,
00329 bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00331 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00332 auto
00333 exp(const framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00334
00336 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00337 auto
00338 log(const framed_multi<Scalar_T,LO,HI,Tune_P>& val, const framed_multi<Scalar_T,LO,HI,Tune_P>& i,
00339 bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00340
00341 namespace std
00342 {
00344 template < typename Scalar_T, const glucat::index_t LO, const glucat::index_t HI, typename Tune_P >
00345 struct numeric_limits< glucat::framed_multi<Scalar_T,LO,HI,Tune_P> > :
00346 public numeric_limits<Scalar_T>
00347 { };
00348 }
00349 #endif // _GLUCAT_FRAMED_MULTI_H

```





## Macros

- `#define _GLUCAT_HASH_N(x)`
- `#define _GLUCAT_HASH_SIZE_T(x)`

## Functions

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::operator* (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_↵`  
`_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`  
*Geometric product.*
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::operator^ (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_↵`  
`_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`  
*Outer product.*
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::operator& (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_↵`  
`_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`  
*Inner product.*
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::operator% (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi<`  
`Scalar_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`  
*Left contraction.*
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::star (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_T, LO,`  
`HI, Tune_P > &rhs) -> Scalar_T`  
*Hestenes scalar product.*
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::operator/ (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_↵`  
`_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`  
*Geometric quotient.*
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::operator| (const framed_multi< Scalar_T, LO, HI, Tune_P > &lhs, const framed_multi< Scalar_↵`  
`_T, LO, HI, Tune_P > &rhs) -> const framed_multi< Scalar_T, LO, HI, Tune_P >`  
*Transformation via twisted adjoint action.*
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::operator<< (std::ostream &os, const framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> std_↵`  
`::ostream &`  
*Write multivector to output.*
- `template<typename Scalar_T , const index_t LO, const index_t HI>`  
`auto glucat::operator<< (std::ostream &os, const std::pair< const index_set< LO, HI >, Scalar_T > &term)`  
`-> std::ostream &`  
*Write term to output.*
- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::operator>> (std::istream &s, framed_multi< Scalar_T, LO, HI, Tune_P > &val) -> std::istream`  
`&`  
*Read multivector from input.*
- `template<typename Scalar_T , const index_t LO, const index_t HI>`  
`static auto glucat::crd_of_mult (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std_↵`  
`::pair< const index_set< LO, HI >, Scalar_T > &rhs) -> Scalar_T`  
*Coordinate of product of terms.*
- `template<typename Scalar_T , const index_t LO, const index_t HI>`  
`auto glucat::operator* (const std::pair< const index_set< LO, HI >, Scalar_T > &lhs, const std::pair< const`  
`index_set< LO, HI >, Scalar_T > &rhs) -> const std::pair< const index_set< LO, HI >, Scalar_T >`

*Product of terms.*

- `template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::sqrt (const framed\_multi< Scalar_T, LO, HI, Tune_P > &val, const framed\_multi< Scalar_T, LO,`  
`HI, Tune_P > &i, bool prechecked) -> const framed\_multi< Scalar_T, LO, HI, Tune_P >`

*Square root of multivector with specified complexifier.*

- `template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::exp (const framed\_multi< Scalar_T, LO, HI, Tune_P > &val) -> const framed\_multi< Scalar_T,`  
`LO, HI, Tune_P >`

*Exponential of multivector.*

- `template<typename Scalar_T , const index\_t LO, const index\_t HI, typename Tune_P >`  
`auto glucat::log (const framed\_multi< Scalar_T, LO, HI, Tune_P > &val, const framed\_multi< Scalar_T, LO,`  
`HI, Tune_P > &i, bool prechecked) -> const framed\_multi< Scalar_T, LO, HI, Tune_P >`

*Natural logarithm of multivector with specified complexifier.*

## 7.11.1 Macro Definition Documentation

### 7.11.1.1 `_GLUCAT_HASH_N`

```
#define _GLUCAT_HASH_N(
 x)
```

**Value:**

(x)

Definition at line 54 of file [framed\\_multi\\_imp.h](#).

### 7.11.1.2 `_GLUCAT_HASH_SIZE_T`

```
#define _GLUCAT_HASH_SIZE_T(
 x)
```

**Value:**

([typename](#) multivector\_t::hash\_size\_t) (x)

Definition at line 55 of file [framed\\_multi\\_imp.h](#).

Referenced by [glucat::framed\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >::[framed\\_multi](#)(), [glucat::operator%](#)(), [glucat::operator&](#)(), [glucat::operator\\*](#)(), and [glucat::operator^](#)() .

## 7.12 `framed_multi_imp.h`

[Go to the documentation of this file.](#)

```
00001 #ifndef _GLUCAT_FRAMED_MULTI_IMP_H
00002 #define _GLUCAT_FRAMED_MULTI_IMP_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 framed_multi_imp.h : Implement the coordinate map representation of a
00006 Clifford algebra element
00007 -----
00008 begin : Sun 2001-12-09
00009 copyright : (C) 2001-2021 by Paul C. Leopardi
00010 *****/
00011
00012 This library is free software: you can redistribute it and/or modify
00013 it under the terms of the GNU Lesser General Public License as published
00014 by the Free Software Foundation, either version 3 of the License, or
```

```

00015 (at your option) any later version.
00016
00017 This library is distributed in the hope that it will be useful,
00018 but WITHOUT ANY WARRANTY; without even the implied warranty of
00019 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00020 GNU Lesser General Public License for more details.
00021
00022 You should have received a copy of the GNU Lesser General Public License
00023 along with this library. If not, see <http://www.gnu.org/licenses/>.
00024
00025 *****
00026 This library is based on a prototype written by Arvind Raja and was
00027 licensed under the LGPL with permission of the author. See Arvind Raja,
00028 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00029 in Ablamowicz, Lounesto and Parra (eds.)
00030 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00031 *****
00032 See also Arvind Raja's original header comments in glucat.h
00033 *****/
00034
00035 #include "glucat/framed_multi.h"
00036
00037 #include "glucat/scalar.h"
00038 #include "glucat/random.h"
00039 #include "glucat/generation.h"
00040 #include "glucat/matrix.h"
00041
00042 #include <sstream>
00043 #include <fstream>
00044
00045 namespace glucat
00046 {
00047 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00048 auto
00049 framed_multi<Scalar_T,LO,HI,Tune_P>::
00050 classname() -> const std::string
00051 { return "framed_multi"; }
00052
00053 #define _GLUCAT_HASH_N(x) (x)
00054 #define _GLUCAT_HASH_SIZE_T(x) (typename multivector_t::hash_size_t)(x)
00055
00056 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00057 framed_multi<Scalar_T,LO,HI,Tune_P>::
00058 framed_multi()
00059 : map_t(_GLUCAT_HASH_N(0))
00060 { }
00061
00062 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00063 framed_multi<Scalar_T,LO,HI,Tune_P>::
00064 framed_multi(const hash_size_t& hash_size)
00065 : map_t(_GLUCAT_HASH_N(hash_size()))
00066 { }
00067
00068 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00069 template< typename Other_Scalar_T >
00070 framed_multi<Scalar_T,LO,HI,Tune_P>::
00071 framed_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val)
00072 : map_t(_GLUCAT_HASH_N(val.size()))
00073 {
00074 for (auto& val_term : val)
00075 this->insert(term_t(val_term.first, numeric_traits<Scalar_T>::to_scalar_t(val_term.second)));
00076 }
00077
00078 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00079 template< typename Other_Scalar_T >
00080 framed_multi<Scalar_T,LO,HI,Tune_P>::
00081 framed_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val,
00082 const index_set_t frm, const bool prechecked)
00083 : map_t(_GLUCAT_HASH_N(val.size()))
00084 {
00085 if (!prechecked && (val.frame() | frm) != frm)
00086 throw error_t("multivector_t(val,frm): cannot initialize with value outside of frame");
00087 for (auto& val_term : val)
00088 this->insert(term_t(val_term.first, numeric_traits<Scalar_T>::to_scalar_t(val_term.second)));
00089 }
00090
00091 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00092 framed_multi<Scalar_T,LO,HI,Tune_P>::
00093 framed_multi(const multivector_t& val,
00094 const index_set_t frm, const bool prechecked)
00095 : map_t(_GLUCAT_HASH_N(val.size()))
00096 {
00097 if (!prechecked && (val.frame() | frm) != frm)
00098 throw error_t("multivector_t(val,frm): cannot initialize with value outside of frame");
00099 for (auto& val_term : val)
00100 this->insert(val_term);
00101 }
00102 }

```

```

00108
00110 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00111 framed_multi<Scalar_T,LO,HI,Tune_P>::
00112 framed_multi(const index_set_t ist, const Scalar_T& crd)
00113 : map_t(_GLUCAT_HASH_N(1))
00114 {
00115 if (crd != Scalar_T(0))
00116 this->insert(term_t(ist, crd));
00117 }
00118
00120 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00121 framed_multi<Scalar_T,LO,HI,Tune_P>::
00122 framed_multi(const index_set_t ist, const Scalar_T& crd,
00123 const index_set_t frm, const bool prechecked)
00124 : map_t(_GLUCAT_HASH_N(1))
00125 {
00126 if (!prechecked && (ist | frm) != frm)
00127 throw error_t("multivector_t(ist,crd,frm): cannot initialize with value outside of frame");
00128 if (crd != Scalar_T(0))
00129 this->insert(term_t(ist, crd));
00130 }
00131
00133 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00134 framed_multi<Scalar_T,LO,HI,Tune_P>::
00135 framed_multi(const Scalar_T& scr, const index_set_t frm)
00136 : map_t(_GLUCAT_HASH_N(1))
00137 {
00138 if (scr != Scalar_T(0))
00139 this->insert(term_t(index_set_t(), scr));
00140 }
00141
00143 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00144 framed_multi<Scalar_T,LO,HI,Tune_P>::
00145 framed_multi(const int scr, const index_set_t frm)
00146 : map_t(_GLUCAT_HASH_N(1))
00147 {
00148 if (scr != Scalar_T(0))
00149 this->insert(term_t(index_set_t(), Scalar_T(scr)));
00150 }
00151
00153 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00154 framed_multi<Scalar_T,LO,HI,Tune_P>::
00155 framed_multi(const vector_t& vec,
00156 const index_set_t frm, const bool prechecked)
00157 : map_t(_GLUCAT_HASH_N(vec.size()))
00158 {
00159 if (!prechecked && index_t(vec.size()) != frm.count())
00160 throw error_t("multivector_t(vec,frm): cannot initialize with vector not matching frame");
00161 auto idx = frm.min();
00162 const auto frm_end = frm.max()+1;
00163 for (auto& crd : vec)
00164 {
00165 *this += term_t(index_set_t(idx), crd);
00166 for (
00167 ++idx;
00168 idx != frm_end && !frm[idx];
00169 ++idx)
00170 ;
00171 }
00172 }
00173
00175 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00176 framed_multi<Scalar_T,LO,HI,Tune_P>::
00177 framed_multi(const std::string& str)
00178 : map_t(_GLUCAT_HASH_N(0))
00179 {
00180 std::istringstream ss(str);
00181 ss » *this;
00182 if (!ss)
00183 throw error_t("multivector_t(str): could not parse string");
00184 // Peek to see if the end of the string has been reached.
00185 ss.peek();
00186 if (!ss.eof())
00187 throw error_t("multivector_t(str): could not parse entire string");
00188 }
00189
00191 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00192 framed_multi<Scalar_T,LO,HI,Tune_P>::
00193 framed_multi(const std::string& str, const index_set_t frm, const bool prechecked)
00194 : map_t(_GLUCAT_HASH_N(0))
00195 {
00196 if (prechecked)
00197 *this = multivector_t(str);
00198 else
00199 *this = multivector_t(multivector_t(str), frm, false);
00200 }
00201

```

```

00203 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00204 template< typename Other_Scalar_T >
00205 framed_multi<Scalar_T,LO,HI,Tune_P>::
00206 framed_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val)
00207 : map_t(_GLUCAT_HASH_N(1))
00208 {
00209 if (val == Other_Scalar_T(0))
00210 return;
00211
00212 const auto dim = val.m_matrix.size1();
00213 using traits_t = numeric_traits<Scalar_T>;
00214 if (dim == 1)
00215 {
00216 this->insert(term_t(index_set_t(), traits_t::to_scalar_t(val.m_matrix(0, 0))));
00217 return;
00218 }
00219 if (dim >= Tune_P::inv_fast_dim_threshold)
00220 {
00221 try
00222 {
00223 *this = (val.template fast_framed_multi<Scalar_T>()).truncated();
00224 return;
00225 }
00226 catch (const glucat_error& e)
00227 { }
00228
00229 const auto val_norm = traits_t::to_scalar_t(val.norm());
00230 if (traits_t::isNaN_or_isInf(val_norm))
00231 {
00232 *this = traits_t::NaN();
00233 return;
00234 }
00235 const auto frm = val.frame();
00236 const auto algebra_dim = set_value_t(1) << frm.count();
00237 auto result = multivector_t(
00238 _GLUCAT_HASH_SIZE_T(std::min<size_t>(algebra_dim, matrix::nnz(val.m_matrix))));
00239 for (auto
00240 stv = set_value_t(0);
00241 stv != algebra_dim;
00242 stv++)
00243 {
00244 const auto ist = index_set_t(stv, frm, true);
00245 const auto crd =
00246 traits_t::to_scalar_t(matrix::inner<Other_Scalar_T>(val.basis_element(ist), val.m_matrix));
00247 if (crd != Scalar_T(0))
00248 result.insert(term_t(ist, crd));
00249 }
00250 *this = result.truncated();
00251 }
00252
00253 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00254 auto
00255 framed_multi<Scalar_T,LO,HI,Tune_P>::
00256 operator==(const multivector_t& rhs) const -> bool
00257 {
00258 if (this->size() != rhs.size())
00259 return false;
00260 const auto rhs_end = rhs.end();
00261 for (auto& this_term : *this)
00262 {
00263 const const_iterator& rhs_it = rhs.find(this_term.first);
00264 if (rhs_it == rhs_end || rhs_it->second != this_term.second)
00265 return false;
00266 }
00267 return true;
00268 }
00269
00271 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00272 inline
00273 auto
00274 framed_multi<Scalar_T,LO,HI,Tune_P>::
00275 operator==(const Scalar_T& scr) const -> bool
00276 {
00277 switch (this->size())
00278 {
00279 case 0:
00280 return scr == Scalar_T(0);
00281 case 1:
00282 {
00283 const auto& this_it = this->begin();
00284 return this_it->first == index_set_t() && this_it->second == scr;
00285 }
00286 default:
00287 return false;
00288 }
00289 }
00290
00292 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >

```

```

00293 inline
00294 auto
00295 framed_multi<Scalar_T,LO,HI,Tune_P>::
00296 operator+= (const Scalar_T& scr) -> multivector_t&
00297 {
00298 *this += term_t(index_set_t(), scr);
00299 return *this;
00300 }
00301
00303 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00304 inline
00305 auto
00306 framed_multi<Scalar_T,LO,HI,Tune_P>::
00307 operator+= (const multivector_t& rhs) -> multivector_t&
00308 { // simply add terms
00309 for (auto& rhs_term : rhs)
00310 *this += rhs_term;
00311 return *this;
00312 }
00313
00315 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00316 inline
00317 auto
00318 framed_multi<Scalar_T,LO,HI,Tune_P>::
00319 operator-= (const Scalar_T& scr) -> multivector_t&
00320 {
00321 *this += term_t(index_set_t(), -scr);
00322 return *this;
00323 }
00324
00326 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00327 inline
00328 auto
00329 framed_multi<Scalar_T,LO,HI,Tune_P>::
00330 operator-= (const multivector_t& rhs) -> multivector_t&
00331 {
00332 for (auto& rhs_term : rhs)
00333 *this += term_t(rhs_term.first, -(rhs_term.second));
00334 return *this;
00335 }
00336
00338 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00339 inline
00340 auto
00341 framed_multi<Scalar_T,LO,HI,Tune_P>::
00342 operator- () const -> const multivector_t
00343 { // multiply coordinates of all terms by -1
00344 auto result = *this;
00345 for (auto& result_term : result)
00346 result_term.second *= Scalar_T(-1);
00347 return result;
00348 }
00349
00351 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00352 auto
00353 framed_multi<Scalar_T,LO,HI,Tune_P>::
00354 operator*= (const Scalar_T& scr) -> multivector_t&
00355 { // multiply coordinates of all terms by scalar
00356 using traits_t = numeric_traits<Scalar_T>;
00357
00358 if (traits_t::isNaN_or_isInf(scr))
00359 return *this = traits_t::NaN();
00360 if (scr == Scalar_T(0))
00361 if (this->isnan())
00362 *this = traits_t::NaN();
00363 else
00364 this->clear();
00365 else
00366 for (auto& this_term : *this)
00367 this_term.second *= scr;
00368 return *this;
00369 }
00370
00372 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00373 auto
00374 operator* (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
00375 framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
00376 {
00377 using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00378 using traits_t = numeric_traits<Scalar_T>;
00379
00380 if (lhs.isnan() || rhs.isnan())
00381 return traits_t::NaN();
00382
00383 const double lhs_size = lhs.size();
00384 const double rhs_size = rhs.size();
00385 const auto our_frame = lhs.frame() | rhs.frame();

```

```

00385 const auto frm_count = our_frame.count();
00386 const auto algebra_dim = set_value_t(1) << frm_count;
00387 const auto direct_mult = lhs_size * rhs_size <= double(algebra_dim);
00388 if (direct_mult)
00389 { // If we have a sparse multiply, store the result directly
00390 auto result = multivector_t(
00391 _GLUCAT_HASH_SIZE_T(size_t(std::min(lhs_size * rhs_size, double(algebra_dim)))));
00392 for (auto& lhs_term : lhs)
00393 for (auto& rhs_term : rhs)
00394 result += lhs_term * rhs_term;
00395 return result;
00396 }
00397 else
00398 { // Past a certain threshold, the matrix algorithm is fastest
00399 using matrix_multi_t = typename multivector_t::matrix_multi_t;
00400 return matrix_multi_t(lhs, our_frame, true) *
00401 matrix_multi_t(rhs, our_frame, true);
00402 }
00403 }
00404
00406 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00407 inline
00408 auto
00409 framed_multi<Scalar_T,LO,HI,Tune_P>::
00410 operator*= (const multivector_t& rhs) -> multivector_t&
00411 { return *this = *this * rhs; }
00412
00414 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00415 auto
00416 operator^ (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
00417 framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
00418 { // Arvind Raja's original reference:
00419 // "old clical, outerproduct(p,q:pterm) in file compmod.pas"
00420 if (lhs.empty() || rhs.empty())
00421 return Scalar_T(0);
00422
00423 using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00424 using index_set_t = typename multivector_t::index_set_t;
00425 using term_t = typename multivector_t::term_t;
00426
00427 const auto empty_set = index_set_t();
00428
00429 const double lhs_size = lhs.size();
00430 const double rhs_size = rhs.size();
00431 const auto lhs_frame = lhs.frame();
00432 const auto rhs_frame = rhs.frame();
00433 const auto our_frame = lhs_frame | rhs_frame;
00434 const auto algebra_dim = set_value_t(1) << our_frame.count();
00435 auto result = multivector_t(
00436 _GLUCAT_HASH_SIZE_T(size_t(std::min(lhs_size * rhs_size, double(algebra_dim)))));
00437 const auto lhs_end = lhs.end();
00438 const auto rhs_end = rhs.end();
00439
00440 if (lhs_size * rhs_size > double(Tune_P::products_size_threshold))
00441 {
00442 for (auto
00443 result_stv = set_value_t(0);
00444 result_stv != algebra_dim;
00445 ++result_stv)
00446 {
00447 const auto result_ist = index_set_t(result_stv, our_frame, true);
00448 const auto lhs_result_frame = lhs_frame & result_ist;
00449 const auto lhs_result_dim = set_value_t(1) << lhs_result_frame.count();
00450 auto result_crd = Scalar_T(0);
00451 for (auto
00452 lhs_stv = set_value_t(0);
00453 lhs_stv != lhs_result_dim;
00454 ++lhs_stv)
00455 {
00456 const auto lhs_ist = index_set_t(lhs_stv, lhs_result_frame, true);
00457 const auto rhs_ist = result_ist ^ lhs_ist;
00458 if ((rhs_ist | rhs_frame) == rhs_frame)
00459 {
00460 const auto lhs_it = lhs.find(lhs_ist);
00461 if (lhs_it != lhs_end)
00462 {
00463 const auto rhs_it = rhs.find(rhs_ist);
00464 if (rhs_it != rhs_end)
00465 result_crd += crd_of_mult(*lhs_it, *rhs_it);
00466 }
00467 }
00468 }
00469 if (result_crd != Scalar_T(0))
00470 result.insert(term_t(result_ist, result_crd));
00471 }
00472 return result;

```

```

00473 }
00474 else
00475 {
00476 for (auto& lhs_term : lhs)
00477 for (auto& rhs_term : rhs)
00478 if ((lhs_term.first & rhs_term.first) == empty_set)
00479 result += lhs_term * rhs_term;
00480 return result;
00481 }
00482 }
00483
00484 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00485 inline
00486 auto
00487 framed_multi<Scalar_T,LO,HI,Tune_P>::
00488 operator^= (const multivector_t& rhs) -> multivector_t&
00489 { return *this = *this ^ rhs; }
00490
00491 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00492 auto
00493 operator& (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
00494 framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
00495 { // Arvind Raja's original reference:
00496 // "old clical, innerproduct(p,q:pterm):pterm in file compmod.pas"
00497 if (lhs.empty() || rhs.empty())
00498 return Scalar_T(0);
00499
00500 using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00501 using index_set_t = typename multivector_t::index_set_t;
00502 using term_t = typename multivector_t::term_t;
00503
00504 const auto lhs_end = lhs.end();
00505 const auto rhs_end = rhs.end();
00506 const double lhs_size = lhs.size();
00507 const double rhs_size = rhs.size();
00508
00509 const auto lhs_frame = lhs.frame();
00510 const auto rhs_frame = rhs.frame();
00511
00512 const auto our_frame = lhs_frame | rhs_frame;
00513 const auto algebra_dim = set_value_t(1) << our_frame.count();
00514 auto result = multivector_t(
00515 _GLUCAT_HASH_SIZE_T(size_t(std::min(lhs_size * rhs_size, double(algebra_dim)))));
00516 if (lhs_size * rhs_size > double(Tune_P::products_size_threshold))
00517 {
00518 for (auto
00519 result_stv = set_value_t(0);
00520 result_stv != algebra_dim;
00521 ++result_stv)
00522 {
00523 const auto result_ist = index_set_t(result_stv, our_frame, true);
00524 const auto comp_frame = our_frame & ~result_ist;
00525 const auto comp_dim = set_value_t(1) << comp_frame.count();
00526 auto result_crd = Scalar_T(0);
00527 for (auto
00528 comp_stv = set_value_t(1);
00529 comp_stv != comp_dim;
00530 ++comp_stv)
00531 {
00532 const auto comp_ist = index_set_t(comp_stv, comp_frame, true);
00533 const auto our_ist = result_ist ^ comp_ist;
00534 if ((our_ist | lhs_frame) == lhs_frame)
00535 {
00536 const auto lhs_it = lhs.find(our_ist);
00537 if (lhs_it != lhs_end)
00538 {
00539 const auto rhs_it = rhs.find(comp_ist);
00540 if (rhs_it != rhs_end)
00541 result_crd += crd_of_mult(*lhs_it, *rhs_it);
00542 }
00543 }
00544 }
00545 if (result_stv != 0)
00546 {
00547 if ((our_ist | rhs_frame) == rhs_frame)
00548 {
00549 const auto rhs_it = rhs.find(our_ist);
00550 if (rhs_it != rhs_end)
00551 {
00552 const auto lhs_it = lhs.find(comp_ist);
00553 if (lhs_it != lhs_end)
00554 result_crd += crd_of_mult(*lhs_it, *rhs_it);
00555 }
00556 }
00557 }
00558 }
00559 }
00560 if (result_crd != Scalar_T(0))

```



```

00561 result.insert(term_t(result_ist, result_crd));
00562 }
00563 }
00564 else
00565 {
00566 const auto empty_set = index_set_t();
00567 for (auto& lhs_term : lhs)
00568 {
00569 const auto lhs_ist = lhs_term.first;
00570 if (lhs_ist != empty_set)
00571 for (auto& rhs_term : rhs)
00572 {
00573 const auto rhs_ist = rhs_term.first;
00574 if (rhs_ist != empty_set)
00575 {
00576 const auto our_ist = lhs_ist | rhs_ist;
00577 if ((lhs_ist == our_ist) || (rhs_ist == our_ist))
00578 result += lhs_term * rhs_term;
00579 }
00580 }
00581 }
00582 }
00583 return result;
00584 }
00585
00586 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00587 inline
00588 auto
00589 framed_multi<Scalar_T,LO,HI,Tune_P>::
00590 operator+=(const multivector_t& rhs) -> multivector_t&
00591 { return *this = *this & rhs; }
00592
00593 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00594 auto
00595 operator% (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
00596 framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
00597 {
00598 // Reference: Leo Dorst, "Honing geometric algebra for its use in the computer sciences",
00599 // in Geometric Computing with Clifford Algebras, ed. G. Sommer,
00600 // Springer 2001, Chapter 6, pp. 127-152.
00601 // http://staff.science.uva.nl/~leo/clifford/index.html
00602
00603 if (lhs.empty() || rhs.empty())
00604 return Scalar_T(0);
00605
00606 using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00607 using index_set_t = typename multivector_t::index_set_t;
00608 using term_t = typename multivector_t::term_t;
00609 using map_t = typename multivector_t::map_t;
00610
00611 const auto lhs_end = lhs.end();
00612 const auto rhs_end = rhs.end();
00613 const double lhs_size = lhs.size();
00614 const double rhs_size = rhs.size();
00615 const auto lhs_frame = lhs.frame();
00616 const auto rhs_frame = rhs.frame();
00617
00618 const auto our_frame = lhs_frame | rhs_frame;
00619 const auto algebra_dim = set_value_t(1) << our_frame.count();
00620 auto result = multivector_t(
00621 _GLUCAT_HASH_SIZE_T(size_t(std::min(lhs_size * rhs_size, double(algebra_dim)))));
00622
00623 if (lhs_size * rhs_size > double(Tune_P::products_size_threshold))
00624 {
00625 for (auto
00626 result_stv = set_value_t(0);
00627 result_stv != algebra_dim;
00628 ++result_stv)
00629 {
00630 const auto result_ist = index_set_t(result_stv, our_frame, true);
00631 const auto comp_frame = lhs_frame & ~result_ist;
00632 const auto comp_dim = set_value_t(1) << comp_frame.count();
00633 auto result_crd = Scalar_T(0);
00634 for (auto
00635 comp_stv = set_value_t(0);
00636 comp_stv != comp_dim;
00637 ++comp_stv)
00638 {
00639 const auto comp_ist = index_set_t(comp_stv, comp_frame, true);
00640 const auto rhs_ist = result_ist ^ comp_ist;
00641 if ((rhs_ist | rhs_frame) == rhs_frame)
00642 {
00643 const auto rhs_it = rhs.find(rhs_ist);
00644 if (rhs_it != rhs_end)
00645 {
00646 const auto lhs_it = lhs.find(comp_ist);
00647 if (lhs_it != lhs_end)

```

```

00649 result_crd += crd_of_mult(*lhs_it, *rhs_it);
00650 }
00651 }
00652 }
00653 if (result_crd != Scalar_T(0))
00654 result.insert(term_t(result_ist, result_crd));
00655 }
00656 }
00657 else
00658 {
00659 for (auto& rhs_term : rhs)
00660 {
00661 const auto rhs_ist = rhs_term.first;
00662 for (auto& lhs_term : lhs)
00663 {
00664 const index_set_t lhs_ist = lhs_term.first;
00665 if ((lhs_ist | rhs_ist) == rhs_ist)
00666 result += lhs_term * rhs_term;
00667 }
00668 }
00669 }
00670 return result;
00671 }
00672
00673 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00674 inline
00675 auto
00676 framed_multi<Scalar_T,LO,HI,Tune_P>::
00677 operator%=(const multivector_t& rhs) -> multivector_t&
00678 { return *this = *this % rhs; }
00679
00680 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00681 auto
00682 star(const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const framed_multi<Scalar_T,LO,HI,Tune_P>& rhs)
00683 -> Scalar_T
00684 {
00685 {
00686 using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00687
00688 auto result = Scalar_T(0);
00689 const auto small_star_large = lhs.size() < rhs.size();
00690 const auto* smallp =
00691 small_star_large
00692 ? &lhs
00693 : &rhs;
00694 const auto* largep =
00695 small_star_large
00696 ? &rhs
00697 : &lhs;
00698
00699 for (auto& small_term : *smallp)
00700 {
00701 const auto small_ist = small_term.first;
00702 const auto large_crd = (*largep)[small_ist];
00703 if (large_crd != Scalar_T(0))
00704 result += small_ist.sign_of_square() * small_term.second * large_crd;
00705 }
00706 return result;
00707 }
00708
00709 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00710 auto
00711 framed_multi<Scalar_T,LO,HI,Tune_P>::
00712 operator/=(const Scalar_T& scr) -> multivector_t&
00713 { // Divide coordinates of all terms by scr
00714 using traits_t = numeric_traits<Scalar_T>;
00715
00716 if (traits_t::isNan(scr))
00717 return *this = traits_t::NaN();
00718 if (traits_t::isInf(scr))
00719 if (this->isnan())
00720 *this = traits_t::NaN();
00721 else
00722 this->clear();
00723 else
00724 for (auto& this_term : *this)
00725 this_term.second /= scr;
00726 return *this;
00727 }
00728
00729 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00730 inline
00731 auto
00732 operator/ (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
00733 framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
00734 {
00735 {
00736 using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00737 using traits_t = numeric_traits<Scalar_T>;

```

```

00738 using index_set_t = typename multivector_t::index_set_t;
00739 using matrix_multi_t = typename multivector_t::matrix_multi_t;
00740
00741 if (rhs == Scalar_T(0))
00742 return traits_t::NaN();
00743
00744 const auto our_frame = lhs.frame() | rhs.frame();
00745 return matrix_multi_t(lhs, our_frame, true) / matrix_multi_t(rhs, our_frame, true);
00746 }
00747
00748 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00749 inline
00750 auto
00751 framed_multi<Scalar_T,LO,HI,Tune_P>::
00752 operator/= (const multivector_t& rhs) -> multivector_t&
00753 { return *this = *this / rhs; }
00754
00755 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00756 inline
00757 auto
00758 operator| (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
00759 framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
00760 {
00761 using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00762 using matrix_multi_t = typename multivector_t::matrix_multi_t;
00763
00764 return matrix_multi_t(rhs) * matrix_multi_t(lhs) / matrix_multi_t(rhs.involute());
00765 }
00766
00767 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00768 inline
00769 auto
00770 framed_multi<Scalar_T,LO,HI,Tune_P>::
00771 operator|= (const multivector_t& rhs) -> multivector_t&
00772 { return *this = *this | rhs; }
00773
00774 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00775 inline
00776 auto
00777 framed_multi<Scalar_T,LO,HI,Tune_P>::
00778 inv() const -> const multivector_t
00779 {
00780 auto result = matrix_multi_t(Scalar_T(1), this->frame());
00781 return result /= matrix_multi_t(*this);
00782 }
00783
00784 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00785 auto
00786 framed_multi<Scalar_T,LO,HI,Tune_P>::
00787 pow(int m) const -> const multivector_t
00788 { return glucat::pow(*this, m); }
00789
00790 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00791 auto
00792 framed_multi<Scalar_T,LO,HI,Tune_P>::
00793 outer_pow(int m) const -> const multivector_t
00794 {
00795 if (m < 0)
00796 throw error_t("outer_pow(int): negative exponent");
00797 auto result = multivector_t(Scalar_T(1));
00798 auto a = *this;
00799 for (;
00800 m != 0;
00801 m >>= 1, a = a ^ a)
00802 if (m & 1)
00803 result ^= a;
00804 return result;
00805 }
00806
00807 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00808 inline
00809 auto
00810 framed_multi<Scalar_T,LO,HI,Tune_P>::
00811 frame() const -> const index_set_t
00812 {
00813 auto result = index_set_t();
00814 for (auto& this_term : *this)
00815 result |= this_term.first;
00816 return result;
00817 }
00818
00819 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00820 inline
00821 auto
00822 framed_multi<Scalar_T,LO,HI,Tune_P>::
00823 grade() const -> index_t
00824 {

```

```

00832 auto result = index_t(0);
00833 for (auto& this_term : *this)
00834 result = std::max(result, this_term.first.count());
00835 return result;
00836 }
00837
00838 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00839 inline
00840 auto
00841 framed_multi<Scalar_T,LO,HI,Tune_P>::
00842 operator[] (const index_set_t ist) const -> Scalar_T
00843 {
00844 {
00845 const auto& this_it = this->find(ist);
00846 if (this_it == this->end())
00847 return Scalar_T(0);
00848 else
00849 return this_it->second;
00850 }
00851
00852 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00853 auto
00854 framed_multi<Scalar_T,LO,HI,Tune_P>::
00855 operator() (index_t grade) const -> const multivector_t
00856 {
00857 {
00858 if ((grade < 0) || (grade > HI-LO))
00859 return Scalar_T(0);
00860 else
00861 {
00862 auto result = multivector_t();
00863 for (auto& this_term : *this)
00864 if (this_term.first.count() == grade)
00865 result += this_term;
00866 return result;
00867 }
00868 }
00869
00870 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00871 inline
00872 auto
00873 framed_multi<Scalar_T,LO,HI,Tune_P>::
00874 scalar() const -> Scalar_T
00875 { return (*this)[index_set_t()]; }
00876
00877 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00878 inline
00879 auto
00880 framed_multi<Scalar_T,LO,HI,Tune_P>::
00881 pure() const -> const multivector_t
00882 { return *this - this->scalar(); }
00883
00884 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00885 auto
00886 framed_multi<Scalar_T,LO,HI,Tune_P>::
00887 even() const -> const multivector_t
00888 { // even part of x, sum of the pure(count) with even count
00889 auto result = multivector_t();
00890 for (auto& this_term : *this)
00891 if ((this_term.first.count() % 2) == 0)
00892 result.insert(this_term);
00893 return result;
00894 }
00895
00896 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00897 auto
00898 framed_multi<Scalar_T,LO,HI,Tune_P>::
00899 odd() const -> const multivector_t
00900 { // even part of x, sum of the pure(count) with even count
00901 auto result = multivector_t();
00902 for (auto& this_term : *this)
00903 if ((this_term.first.count() % 2) == 1)
00904 result.insert(this_term);
00905 return result;
00906 }
00907
00908 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00909 auto
00910 framed_multi<Scalar_T,LO,HI,Tune_P>::
00911 vector_part() const -> const vector_t
00912 { return this->vector_part(this->frame(), true); }
00913
00914 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00915 auto
00916 framed_multi<Scalar_T,LO,HI,Tune_P>::
00917 vector_part(const index_set_t frm, const bool prechecked) const -> const vector_t
00918 {
00919 if (!prechecked && (this->frame() | frm) != frm)
00920 throw error_t("vector_part(frm): value is outside of requested frame");

```

```

00927 auto result = vector_t();
00928 result.reserve(frm.count());
00929 const auto frm_end = frm.max()+1;
00930 for (auto
00931 idx = frm.min();
00932 idx != frm_end;
00933 ++idx)
00934 // Frame may contain indices which do not correspond to a grade 1 term but
00935 // frame cannot omit any index corresponding to a grade 1 term
00936 if (frm[idx])
00937 result.push_back((*this)[index_set_t(idx)]);
00938 return result;
00939 }
00940
00942 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00943 auto
00944 framed_multi<Scalar_T,LO,HI,Tune_P>::
00945 involute() const -> const multivector_t
00946 {
00947 auto result = *this;
00948 for (auto& result_term : result)
00949 { // for a k-vector u, involute(u) == (-1)^k * u
00950 if ((result_term.first.count() % 2) == 1)
00951 result_term.second *= Scalar_T(-1);
00952 }
00953 return result;
00954 }
00955
00957 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00958 auto
00959 framed_multi<Scalar_T,LO,HI,Tune_P>::
00960 reverse() const -> const multivector_t
00961 {
00962 auto result = *this;
00963 for (auto& result_term : result)
00964 // For a k-vector u, reverse(u) = { -u, k == 2,3 (mod 4)
00965 // { u, k == 0,1 (mod 4)
00966 switch (result_term.first.count() % 4)
00967 {
00968 case 2:
00969 case 3:
00970 result_term.second *= Scalar_T(-1);
00971 break;
00972 default:
00973 break;
00974 }
00975 return result;
00976 }
00977
00979 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00980 auto
00981 framed_multi<Scalar_T,LO,HI,Tune_P>::
00982 conj() const -> const multivector_t
00983 {
00984 auto result = *this;
00985 for (auto& result_term : result)
00986 // For a k-vector u, conj(u) = { -u, k == 1,2 (mod 4)
00987 // { u, k == 0,3 (mod 4)
00988 switch (result_term.first.count() % 4)
00989 {
00990 case 1:
00991 case 2:
00992 result_term.second *= Scalar_T(-1);
00993 break;
00994 default:
00995 break;
00996 }
00997 return result;
00998 }
00999
01001 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01002 auto
01003 framed_multi<Scalar_T,LO,HI,Tune_P>::
01004 quad() const -> Scalar_T
01005 {
01006 // scalar(conj(x)*x) = 2*quad(even(x)) - quad(x)
01007 // ref: old clical: quadfunction(p:pter):pterm in file compmod.pas
01008 auto result = Scalar_T(0);
01009 for (auto& this_term : *this)
01010 {
01011 const auto sign =
01012 (this_term.first.count_neg() % 2)
01013 ? -Scalar_T(1)
01014 : Scalar_T(1);
01015 result += sign * (this_term.second) * (this_term.second);
01016 }
01017 return result;

```

```

01018 }
01019
01020 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01021 auto
01022 framed_multi<Scalar_T,LO,HI,Tune_P>::
01023 norm() const -> Scalar_T
01024 {
01025 using traits_t = numeric_traits<Scalar_T>;
01026
01027 auto result = Scalar_T(0);
01028 for (auto& this_term : *this)
01029 {
01030 const auto abs_crd = traits_t::abs(this_term.second);
01031 result += abs_crd * abs_crd;
01032 }
01033 return result;
01034 }
01035 }
01036
01037 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01038 auto
01039 framed_multi<Scalar_T,LO,HI,Tune_P>::
01040 max_abs() const -> Scalar_T
01041 {
01042 using traits_t = numeric_traits<Scalar_T>;
01043
01044 auto result = Scalar_T(0);
01045 for (auto& this_term : *this)
01046 {
01047 const auto abs_crd = traits_t::abs(this_term.second);
01048 if (abs_crd > result)
01049 result = abs_crd;
01050 }
01051 return result;
01052 }
01053 }
01054
01055 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01056 auto
01057 framed_multi<Scalar_T,LO,HI,Tune_P>::
01058 random(const index_set_t frm, Scalar_T fill) -> const multivector_t
01059 {
01060 using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
01061 using index_set_t = typename multivector_t::index_set_t;
01062 using term_t = typename multivector_t::term_t;
01063
01064 using random_generator_t = random_generator<Scalar_T>;
01065 auto& generator = random_generator_t::generator();
01066
01067 fill =
01068 (fill < Scalar_T(0))
01069 ? Scalar_T(0)
01070 : (fill > Scalar_T(1))
01071 ? Scalar_T(1)
01072 : fill;
01073 const auto algebra_dim = set_value_t(1) << frm.count();
01074 using traits_t = numeric_traits<Scalar_T>;
01075 const auto mean_abs = traits_t::sqrt(Scalar_T(double(algebra_dim)));
01076 auto result = multivector_t();
01077 for (auto
01078 stv = set_value_t(0);
01079 stv != algebra_dim;
01080 ++stv)
01081 {
01082 if (generator.uniform() < fill)
01083 {
01084 const auto& result_crd = generator.normal() / mean_abs;
01085 result.insert(term_t(index_set_t(stv, frm, true), result_crd));
01086 }
01087 }
01088 return result;
01089 }
01090 }
01091
01092 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01093 inline
01094 void
01095 framed_multi<Scalar_T,LO,HI,Tune_P>::
01096 write(const std::string& msg) const
01097 { std::cout << msg << std::endl << " " << (*this) << std::endl; }
01098
01099 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01100 inline
01101 void
01102 framed_multi<Scalar_T,LO,HI,Tune_P>::
01103 write(std::ofstream& ofile, const std::string& msg) const
01104 {
01105 if (!ofile)
01106 throw error_t("write(ofile,msg): cannot write to output file");
01107 ofile << msg << std::endl << " " << (*this) << std::endl;
01108 }
01109 }

```

```

01111 template< typename Map_T,typename Sorted_Map_T >
01112 class sorted_range
01113 {
01114 public:
01115 using map_t = Map_T;
01116 using sorted_map_t = Sorted_Map_T;
01117 using sorted_iterator = typename Sorted_Map_T::const_iterator;
01118
01119 sorted_range (Sorted_Map_T &sorted_val, const Map_T& val)
01120 {
01121 for (auto& val_term : val)
01122 sorted_val.insert(val_term);
01123 sorted_begin = sorted_val.begin();
01124 sorted_end = sorted_val.end();
01125 }
01126 sorted_iterator sorted_begin;
01127 sorted_iterator sorted_end;
01128 };
01129
01130 template< typename Sorted_Map_T >
01131 class sorted_range< Sorted_Map_T, Sorted_Map_T >
01132 {
01133 public:
01134 using map_t = Sorted_Map_T;
01135 using sorted_map_t = Sorted_Map_T;
01136 using sorted_iterator = typename Sorted_Map_T::const_iterator;
01137
01138 sorted_range (Sorted_Map_T &sorted_val, const Sorted_Map_T& val)
01139 : sorted_begin(val.begin()),
01140 sorted_end(val.end())
01141 { }
01142 sorted_iterator sorted_begin;
01143 sorted_iterator sorted_end;
01144 };
01145
01147 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01148 auto
01149 operator<< (std::ostream& os, const framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::ostream&
01150 {
01151 using limits_t = std::numeric_limits<Scalar_T>;
01152 if (val.empty())
01153 os << 0;
01154 else if (val.isnan())
01155 os << limits_t::quiet_NaN();
01156 else if (val.isinf())
01157 {
01158 const Scalar_T& inf = limits_t::infinity();
01159 os << (scalar(val) < 0.0 ? -inf : inf);
01160 }
01161 else
01162 {
01163 using traits_t = numeric_traits<Scalar_T>;
01164 using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
01165 Scalar_T truncation;
01166 switch (os.flags() & std::ios::floatfield)
01167 {
01168 case std::ios_base::scientific:
01169 truncation = Scalar_T(1) / traits_t::pow(Scalar_T(10), int(os.precision()) + 1);
01170 break;
01171 case std::ios_base::fixed:
01172 truncation = Scalar_T(1) / (traits_t::pow(Scalar_T(10), int(os.precision())) *
01173 val.max_abs());
01174 break;
01175 case std::ios_base::fixed | std::ios_base::scientific:
01176 truncation = multivector_t::default_truncation;
01177 break;
01178 default:
01179 truncation = Scalar_T(1) / traits_t::pow(Scalar_T(10), int(os.precision()));
01180 break;
01181 }
01182 auto truncated_val = val.truncated(truncation);
01183 if (truncated_val.empty())
01184 os << 0;
01185 else
01186 {
01187 using map_t = typename multivector_t::map_t;
01188 using sorted_map_t = typename multivector_t::sorted_map_t;
01189 using sorted_iterator = typename sorted_map_t::const_iterator;
01190 auto sorted_val = sorted_map_t();
01191 const auto sorted_val_range = sorted_range< map_t, sorted_map_t >(sorted_val, truncated_val);
01192 auto sorted_it = sorted_val_range.sorted_begin;
01193 os << *sorted_it;
01194 for (++sorted_it;
01195 sorted_it != sorted_val_range.sorted_end;
01196 ++sorted_it)
01197 {
01198 const Scalar_T& scr = sorted_it->second;

```

```

01198 if (scr >= 0.0)
01199 os << '+';
01200 os << *sorted_it;
01201 }
01202 }
01203 }
01204 return os;
01205 }
01206
01208 template< typename Scalar_T, const index_t LO, const index_t HI >
01209 auto
01210 operator<< (std::ostream& os, const std::pair< const index_set<LO,HI>, Scalar_T >& term) ->
std::ostream&
01211 {
01212 const auto second_as_double = numeric_traits<Scalar_T>::to_double(term.second);
01213 const auto use_double =
01214 (os.precision() <= std::numeric_limits<double>::digits10) ||
01215 (term.second == Scalar_T(second_as_double));
01216 if (term.first.count() == 0)
01217 if (use_double)
01218 os << second_as_double;
01219 else
01220 os << term.second;
01221 else if (term.second == Scalar_T(-1))
01222 {
01223 os << '-';
01224 os << term.first;
01225 }
01226 else if (term.second != Scalar_T(1))
01227 {
01228 if (use_double)
01229 {
01230 auto tol = std::pow(10.0,-os.precision());
01231 if (std::fabs(second_as_double + 1.0) < tol)
01232 os << '-';
01233 else if (std::fabs(second_as_double - 1.0) >= tol)
01234 os << second_as_double;
01235 }
01236 else
01237 os << term.second;
01238 os << term.first;
01239 }
01240 else
01241 os << term.first;
01242 return os;
01243 }
01244
01246 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01247 auto
01248 operator>> (std::istream& s, framed_multi<Scalar_T,LO,HI,Tune_P> & val) -> std::istream&
01249 { // Input looks like 1.0-2.0{1,2}+3.2{3,4}.
01250 using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
01251 // Parsing variables.
01252 auto local_val = multivector_t();
01253 auto c = 0;
01254 // Parsing control variables.
01255 auto negative = false;
01256 auto expect_term = true;
01257 // The multivector may begin with '+' or '-'. Check for this.
01258 c = s.peek();
01259 if (s.good() && (c == int('+') || c == int('-')))
01260 { // A '-' here negates the following term.
01261 negative = (c == int('-'));
01262 // Consume the '+' or '-'.
01263 s.get();
01264 }
01265 while (s.good())
01266 { // Parse a term.
01267 // A term consists of an optional scalar, followed by an optional index set.
01268 // At least one of the two must be present.
01269 // Default coordinate is Scalar_T(1).
01270 auto coordinate = Scalar_T(1);
01271 // Default index set is empty.
01272 auto ist = index_set<LO,HI>();
01273 // First, check for an opening brace.
01274 c = s.peek();
01275 if (s.good())
01276 { // If the character is not an opening brace,
01277 // a coordinate value is expected here.
01278 if (c != int('{'))
01279 { // Try to read a coordinate value.
01280 double coordinate_as_double;
01281 s >> coordinate_as_double;
01282 // Reading the coordinate may have resulted in an end of file condition.
01283 // This is not a failure.
01284 if (s)
01285 coordinate = Scalar_T(coordinate_as_double);

```



```

01286 }
01287 }
01288 else
01289 { // End of file here ends parsing while a term may still be expected.
01290 break;
01291 }
01292 // Coordinate is now Scalar_T(1) or a Scalar_T value.
01293 // Parse an optional index set.
01294 if (s.good())
01295 {
01296 c = s.peek();
01297 if (s.good() && c == int('{'))
01298 { // Try to read index set.
01299 s » ist;
01300 }
01301 }
01302 // Reading the term may have resulted in an end of file condition.
01303 // This is not a failure.
01304 if (s)
01305 {
01306 // Immediately after parsing a term, another term is not expected.
01307 expect_term = false;
01308 if (coordinate != Scalar_T(0))
01309 {
01310 // Add the term to the local multivector.
01311 coordinate =
01312 negative
01313 ? -coordinate
01314 : coordinate;
01315 using term_t = typename multivector_t::term_t;
01316 local_val += term_t(ist, coordinate);
01317 }
01318 }
01319 // Check if anything follows the current term.
01320 if (s.good())
01321 {
01322 c = s.peek();
01323 if (s.good())
01324 { // Only '+' and '-' are valid here.
01325 if (c == int('+') || c == int('-'))
01326 { // A '-' here negates the following term.
01327 negative = (c == int('-'));
01328 // Consume the '+' or '-'.
01329 s.get();
01330 // Immediately after '+' or '-',
01331 // expect another term.
01332 expect_term = true;
01333 }
01334 else
01335 { // Any other character here is a not failure,
01336 // but still ends the parsing of the multivector.
01337 break;
01338 }
01339 }
01340 }
01341 }
01342 // If a term is still expected, this is a failure.
01343 if (expect_term)
01344 s.clear(std::istream::failbit);
01345 // End of file is not a failure.
01346 if (s)
01347 { // The multivector has been successfully parsed.
01348 val = local_val;
01349 }
01350 return s;
01351 }
01352
01353 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01354 auto
01355 framed_multi<Scalar_T,LO,HI,Tune_P>::
01356 nbr_terms () const -> unsigned long
01357 { return this->size(); }
01358
01359 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01360 inline
01361 auto
01362 framed_multi<Scalar_T,LO,HI,Tune_P>::
01363 operator+= (const term_t& term) -> multivector_t&
01364 { // Do not insert terms with 0 coordinate
01365 if (term.second != Scalar_T(0))
01366 {
01367 const auto& this_it = this->find(term.first);
01368 if (this_it == this->end())
01369 this->insert(term);
01370 else if (this_it->second + term.second == Scalar_T(0))
01371 // Erase term if resulting coordinate is 0
01372 this->erase(this_it);
01373 }
01374 }

```

```

01375 else
01376 this_it->second += term.second;
01377 }
01378 return *this;
01379 }
01380
01382 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01383 auto
01384 framed_multi<Scalar_T,LO,HI,Tune_P>::
01385 isinf() const -> bool
01386 {
01387 using traits_t = numeric_traits<Scalar_T>;
01388
01389 if (std::numeric_limits<Scalar_T>::has_infinity)
01390 for (auto& this_term : *this)
01391 if (traits_t::isInf(this_term.second))
01392 return true;
01393 return false;
01394 }
01395
01397 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01398 auto
01399 framed_multi<Scalar_T,LO,HI,Tune_P>::
01400 isnan() const -> bool
01401 {
01402 using traits_t = numeric_traits<Scalar_T>;
01403
01404 if (std::numeric_limits<Scalar_T>::has_quiet_NaN)
01405 for (auto& this_term : *this)
01406 if (traits_t::isNaN(this_term.second))
01407 return true;
01408 return false;
01409 }
01410
01412 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01413 auto
01414 framed_multi<Scalar_T,LO,HI,Tune_P>::
01415 truncated(const Scalar_T& limit) const -> const multivector_t
01416 {
01417 using traits_t = numeric_traits<Scalar_T>;
01418
01419 if (this->isnan() || this->isinf())
01420 return *this;
01421 const auto truncation = traits_t::abs(limit);
01422 const auto top = max_abs();
01423 auto result = multivector_t();
01424 if (top != Scalar_T(0))
01425 for (auto& this_term : *this)
01426 if (traits_t::abs(this_term.second) > top * truncation)
01427 result.insert(this_term);
01428 return result;
01429 }
01430
01432 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01433 auto
01434 framed_multi<Scalar_T,LO,HI,Tune_P>::
01435 fold(const index_set_t frm) const -> multivector_t
01436 {
01437 if (frm.is_contiguous())
01438 return *this;
01439 else
01440 {
01441 auto result = multivector_t();
01442 for (auto& this_term : *this)
01443 result.insert(term_t(this_term.first.fold(frm), this_term.second));
01444 return result;
01445 }
01446 }
01447
01449 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01450 auto
01451 framed_multi<Scalar_T,LO,HI,Tune_P>::
01452 unfold(const index_set_t frm) const -> multivector_t
01453 {
01454 if (frm.is_contiguous())
01455 return *this;
01456 else
01457 {
01458 auto result = multivector_t();
01459 for (auto& this_term : *this)
01460 result.insert(term_t(this_term.first.unfold(frm), this_term.second));
01461 return result;
01462 }
01463 }
01464
01466 // Reference: [L] 16.4 Periodicity of 8, p216
01467 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >

```

```

01468 auto
01469 framed_multi<Scalar_T,LO,HI,Tune_P>::
01470 centre_pm4_qp4(index_t& p, index_t& q) -> multivector_t&
01471 {
01472 // We add 4 to q by subtracting 4 from p
01473 if (q+4 > -LO)
01474 throw error_t("centre_pm4_qp4(p,q): LO is too high to represent this value");
01475 if (this->frame().max() > p-4)
01476 {
01477 using index_pair_t = typename index_set_t::index_pair_t;
01478 const auto pm3210 = index_set_t(index_pair_t(p-3,p), true);
01479 const auto qm4321 = index_set_t(index_pair_t(-q-4,-q-1), true);
01480 const auto& tqm4321 = term_t(qm4321, Scalar_T(1));
01481 auto result = multivector_t();
01482 for (auto& this_term : *this)
01483 {
01484 const auto ist = this_term.first;
01485 if (ist.max() > p-4)
01486 {
01487 auto var_term = var_term_t();
01488 for (auto
01489 n = index_t(0);
01490 n != index_t(4);
01491 ++n)
01492 if (ist[n+p-3])
01493 var_term *= term_t(index_set_t(n-q-4), Scalar_T(1)) * tqm4321;
01494 // Mask out {p-3}..{p}
01495 result.insert(term_t(ist & ~pm3210, this_term.second) *
01496 term_t(var_term.first, var_term.second));
01497 }
01498 else
01499 result.insert(this_term);
01500 }
01501 *this = result;
01502 }
01503 p -=4; q += 4;
01504 return *this;
01505 }
01506
01508 // Reference: [L] 16.4 Periodicity of 8, p216
01509 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01510 auto
01511 framed_multi<Scalar_T,LO,HI,Tune_P>::
01512 centre_pp4_qm4(index_t& p, index_t& q) -> multivector_t&
01513 {
01514 // We add 4 to p by subtracting 4 from q
01515 if (p+4 > HI)
01516 throw error_t("centre_pp4_qm4(p,q): HI is too low to represent this value");
01517 if (this->frame().min() < -q+4)
01518 {
01519 using index_pair_t = typename index_set_t::index_pair_t;
01520 const auto qp0123 = index_set_t(index_pair_t(-q,-q+3), true);
01521 const auto pp1234 = index_set_t(index_pair_t(p+1,p+4), true);
01522 const auto& tpp1234 = term_t(pp1234, Scalar_T(1));
01523 auto result = multivector_t();
01524 for (auto& this_term : *this)
01525 {
01526 index_set_t ist = this_term.first;
01527 if (ist.min() < -q+4)
01528 {
01529 auto var_term = var_term_t();
01530 for (auto
01531 n = index_t(0);
01532 n != index_t(4);
01533 ++n)
01534 if (ist[n-q])
01535 var_term *= term_t(index_set_t(n+p+1), Scalar_T(1)) * tpp1234;
01536 // Mask out {-q}..{-q+3}
01537 result.insert(term_t(var_term.first, var_term.second) *
01538 term_t(ist & ~qp0123, this_term.second));
01539 }
01540 else
01541 result.insert(this_term);
01542 }
01543 *this = result;
01544 }
01545 p +=4; q -= 4;
01546 return *this;
01547 }
01548
01550 // Reference: [P] Proposition 15.20, p 131
01551 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01552 auto
01553 framed_multi<Scalar_T,LO,HI,Tune_P>::
01554 centre_qp1_pm1(index_t& p, index_t& q) -> multivector_t&
01555 {
01556 if (q+1 > HI)

```

```

01557 throw error_t("centre_qpl_pml(p,q): HI is too low to represent this value");
01558 if (p-1 > -LO)
01559 throw error_t("centre_qpl_pml(p,q): LO is too high to represent this value");
01560 const auto qpl = index_set_t(q+1);
01561 const auto& tqpl = term_t(qpl, Scalar_T(1));
01562 auto result = multivector_t();
01563 for (auto& this_term : *this)
01564 {
01565 const auto ist = this_term.first;
01566 auto var_term = var_term_t(index_set_t(), this_term.second);
01567 for (auto
01568 n = -q;
01569 n != p;
01570 ++n)
01571 if (n != 0 && ist[n])
01572 var_term *= term_t(index_set_t(-n) | qpl, Scalar_T(1));
01573 if (p != 0 && ist[p])
01574 var_term *= tqpl;
01575 result.insert(term_t(var_term.first, var_term.second));
01576 }
01577 index_t orig_p = p;
01578 p = q+1;
01579 q = orig_p-1;
01580 return *this = result;
01581 }
01582
01583 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01584 auto
01585 framed_multi<Scalar_T,LO,HI,Tune_P>::
01586 divide(const index_set_t ist) const -> const framed_pair_t
01587 {
01588 auto quo = multivector_t();
01589 auto rem = multivector_t();
01590 for (auto& this_term : *this)
01591 if ((this_term.first | ist) == this_term.first)
01592 quo.insert(term_t(this_term.first ^ ist, this_term.second));
01593 else
01594 rem.insert(this_term);
01595 return framed_pair_t(quo, rem);
01596 }
01597
01598 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01599 auto
01600 framed_multi<Scalar_T,LO,HI,Tune_P>::
01601 fast(const index_t level, const bool odd) const -> const matrix_t
01602 {
01603 // Assume val is already folded and centred
01604 if (this->empty())
01605 {
01606 using matrix_index_t = typename matrix_multi_t::matrix_index_t;
01607 const auto dim = matrix_index_t(1) << level;
01608 auto result = matrix_t(dim, dim);
01609 result.clear();
01610 return result;
01611 }
01612 if (level == 0)
01613 return matrix::unit<matrix_t>(1) * this->scalar();
01614
01615 using basis_matrix_t = typename matrix_multi_t::basis_matrix_t;
01616 using basis_scalar_t = typename basis_matrix_t::value_type;
01617
01618 const auto& I = matrix::unit<basis_matrix_t>(2);
01619 auto J = basis_matrix_t(2,2,2);
01620 J.clear();
01621 J(0,1) = basis_scalar_t(-1);
01622 J(1,0) = basis_scalar_t(1);
01623 auto K = J;
01624 K(0,1) = basis_scalar_t(1);
01625 auto JK = I;
01626 JK(0,0) = basis_scalar_t(-1);
01627
01628 const auto ist_mn = index_set_t(-level);
01629 const auto ist_pn = index_set_t(level);
01630 if (level == 1)
01631 {
01632 if (odd)
01633 return matrix_t(J) * (*this)[ist_mn] + matrix_t(K) * (*this)[ist_pn];
01634 else
01635 return matrix_t(I) * this->scalar() + matrix_t(JK) * (*this)[ist_mn ^ ist_pn];
01636 }
01637 else
01638 {
01639 const auto& pair_mn = this->divide(ist_mn);
01640 const auto& quo_mn = pair_mn.first;
01641 const auto& rem_mn = pair_mn.second;
01642 const auto& pair_quo_mnpn = quo_mn.divide(ist_pn);
01643 const auto& val_mnpn = pair_quo_mnpn.first;

```

```

01646 const auto& val_mn = pair_quo_mnpn.second;
01647 const auto& pair_rem_mnpn = rem_mn.divide(ist_pn);
01648 const auto& val_pn = pair_rem_mnpn.first;
01649 const auto& val_l = pair_rem_mnpn.second;
01650 using matrix::kron;
01651 if (odd)
01652 return - kron(JK, val_l.fast (level-1, 1))
01653 + kron(I, val_mnpn.fast (level-1, 1))
01654 + kron(J, val_mn.fast (level-1, 0))
01655 + kron(K, val_pn.fast (level-1, 0));
01656 else
01657 return kron(I, val_l.fast (level-1, 0))
01658 + kron(JK, val_mnpn.fast (level-1, 0))
01659 + kron(K, val_mn.fast (level-1, 1))
01660 - kron(J, val_pn.fast (level-1, 1));
01661 }
01662 }
01663
01665 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01666 template< typename Other_Scalar_T >
01667 auto
01668 framed_multi<Scalar_T,LO,HI,Tune_P>::
01669 fast_matrix_multi(const index_set_t frm) const -> const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>
01670 {
01671 // Fold val
01672 auto val = this->fold(frm);
01673 auto p = frm.count_pos();
01674 auto q = frm.count_neg();
01675 const auto bott_offset = gen::offset_to_super[pos_mod(p - q, 8)];
01676 p += std::max(bott_offset, index_t(0));
01677 q -= std::min(bott_offset, index_t(0));
01678 if (p > HI)
01679 throw error_t("fast_matrix_multi(frm): HI is too low to represent this value");
01680 if (q > -LO)
01681 throw error_t("fast_matrix_multi(frm): LO is too high to represent this value");
01682 // Centre val
01683 while (p - q > 4)
01684 val.centre_pm4_qp4(p, q);
01685 while (p - q < -3)
01686 val.centre_pp4_qm4(p, q);
01687 if (p - q > 1)
01688 val.centre_qp1_pm1(p, q);
01689 const index_t level = (p + q)/2;
01690
01691 // Do the fast transform
01692 const auto& ev_val = val.even();
01693 const auto& od_val = val.odd();
01694 return matrix_multi<Other_Scalar_T,LO,HI,Tune_P>(ev_val.fast(level, 0) + od_val.fast(level, 1),
01695 frm);
01696 }
01697
01697 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01698 inline
01699 auto
01700 framed_multi<Scalar_T,LO,HI,Tune_P>::
01701 fast_framed_multi() const -> const multivector_t
01702 { return *this; }
01703
01705 template< typename Scalar_T, const index_t LO, const index_t HI >
01706 inline
01707 static
01708 auto
01709 crd_of_mult(const std::pair<const index_set<LO,HI>, Scalar_T>& lhs,
01710 const std::pair<const index_set<LO,HI>, Scalar_T>& rhs) -> Scalar_T
01711 { return lhs.first.sign_of_mult(rhs.first) * lhs.second * rhs.second; }
01712
01714 template< typename Scalar_T, const index_t LO, const index_t HI >
01715 inline
01716 auto
01717 operator* (const std::pair<const index_set<LO,HI>, Scalar_T>& lhs,
01718 const std::pair<const index_set<LO,HI>, Scalar_T>& rhs) -> const std::pair<const
01719 index_set<LO,HI>, Scalar_T>
01720 {
01720 using term_t = std::pair<const index_set<LO,HI>, Scalar_T>;
01721 return term_t(lhs.first ^ rhs.first, crd_of_mult(lhs, rhs));
01722 }
01723
01725 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01726 auto
01727 sqrt(const framed_multi<Scalar_T,LO,HI,Tune_P>& val, const framed_multi<Scalar_T,LO,HI,Tune_P>& i,
01728 bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
01729 {
01729 using traits_t = numeric_traits<Scalar_T>;
01730 if (val.isnan())
01731 return traits_t::NaN();
01732 check_complex(val, i, prechecked);
01733 }

```

```

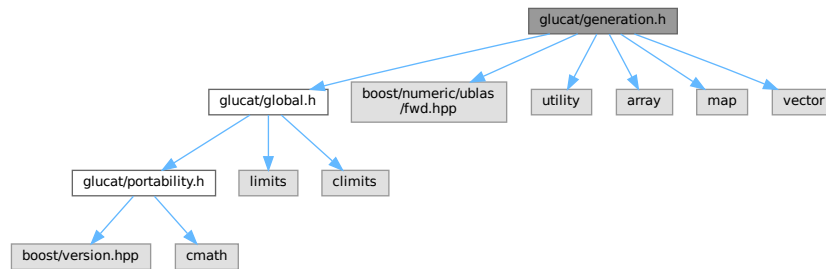
01734
01735 const auto realval = val.scalar();
01736 if (val == realval)
01737 {
01738 if (realval < Scalar_T(0))
01739 return i * traits_t::sqrt(-realval);
01740 else
01741 return traits_t::sqrt(realval);
01742 }
01743 using matrix_multi_t = typename framed_multi<Scalar_T,LO,HI,Tune_P>::matrix_multi_t;
01744 return sqrt(matrix_multi_t(val), matrix_multi_t(i), prechecked);
01745 }
01746
01747 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01748 auto
01749 exp(const framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
01750 {
01751 using traits_t = numeric_traits<Scalar_T>;
01752 if (val.isnan())
01753 return traits_t::NaN();
01754
01755 const auto s = scalar(val);
01756 if (val == s)
01757 return traits_t::exp(s);
01758
01759 const double size = val.size();
01760 const auto frm_count = val.frame().count();
01761 const auto algebra_dim = set_value_t(1) << frm_count;
01762
01763 if((size * size <= double(algebra_dim)) || (frm_count < Tune_P::mult_matrix_threshold))
01764 {
01765 switch (Tune_P::function_precision)
01766 {
01767 case precision_demoted:
01768 {
01769 using demoted_scalar_t = typename traits_t::demoted::type;
01770 using demoted_multivector_t = framed_multi<demoted_scalar_t,LO,HI,Tune_P>;
01771
01772 const auto& demoted_val = demoted_multivector_t(val);
01773 return clifford_exp(demoted_val);
01774 }
01775 break;
01776 case precision_promoted:
01777 {
01778 using promoted_scalar_t = typename traits_t::promoted::type;
01779 using promoted_multivector_t = framed_multi<promoted_scalar_t,LO,HI,Tune_P>;
01780
01781 const auto& promoted_val = promoted_multivector_t(val);
01782 return clifford_exp(promoted_val);
01783 }
01784 break;
01785 default:
01786 return clifford_exp(val);
01787 }
01788 }
01789 else
01790 {
01791 using matrix_multi_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
01792 return exp(matrix_multi_t(val));
01793 }
01794 }
01795
01796 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01797 auto
01800 log(const framed_multi<Scalar_T,LO,HI,Tune_P>& val, const framed_multi<Scalar_T,LO,HI,Tune_P>& i,
 bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
01801 {
01802 using traits_t = numeric_traits<Scalar_T>;
01803 if (val == Scalar_T(0) || val.isnan())
01804 return traits_t::NaN();
01805
01806 check_complex(val, i, prechecked);
01807
01808 const auto realval = val.scalar();
01809 if (val == realval)
01810 {
01811 if (realval < Scalar_T(0))
01812 return i * traits_t::pi() + traits_t::log(-realval);
01813 else
01814 return traits_t::log(realval);
01815 }
01816 using matrix_multi_t = typename framed_multi<Scalar_T,LO,HI,Tune_P>::matrix_multi_t;
01817 return log(matrix_multi_t(val), matrix_multi_t(i), prechecked);
01818 }
01819 }
01820 #endif // _GLUCAT_FRAMED_MULTI_IMP_H

```

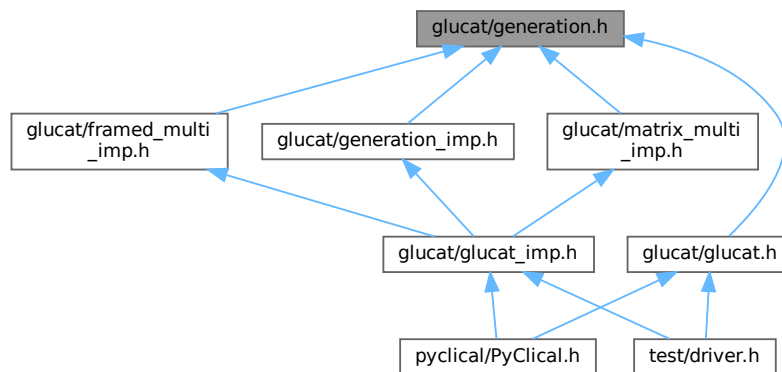
## 7.13 glucat/generation.h File Reference

```
#include "glucat/global.h"
#include <boost/numeric/ublas/fwd.hpp>
#include <utility>
#include <array>
#include <map>
#include <vector>
```

Include dependency graph for generation.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class `glucat::gen::generator_table< Matrix_T >`  
*Table of generators for specific signatures.*

### Namespaces

- namespace `glucat`
- namespace `glucat::gen`

## Typedefs

- using `glucat::gen::signature_t = std::pair<index_t, index_t>`  
*A signature is a pair of indices, p, q, with  $p == \text{frame.max}()$ ,  $q == -\text{frame.min}()$*

## Variables

- static const `std::array< index_t, 8 > glucat::gen::offset_to_super = {0,-1, 0,-1,-2, 3, 2, 1}`  
*Offsets between the current signature and that of the real superalgebra.*

## 7.14 generation.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_GENERATION_H
00002 #define _GLUCAT_GENERATION_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 generation.h : Declare functions for generation of the matrix representation
00006 -----
00007 begin : Wed Jan 23 2002
00008 copyright : (C) 2002-2012 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 #include "glucat/global.h"
00035
00036 #include <boost/numeric/ublas/fwd.hpp>
00037
00038 #include <utility>
00039 #include <array>
00040 #include <map>
00041 #include <vector>
00042
00043 namespace glucat { namespace gen
00044 {
00045 namespace ublas = boost::numeric::ublas;
00046
00047 using signature_t = std::pair<index_t, index_t>;
00048
00049 template< class Matrix_T >
00050 class generator_table :
00051 private std::map< signature_t, std::vector<Matrix_T> >
00052 {
00053 public:
00054 auto operator() (const index_t p, const index_t q) -> const Matrix_T*;
00055 static auto generator() -> generator_table<Matrix_T>&;
00056 private:
00057 auto gen_vector(const index_t p, const index_t q) -> const std::vector<Matrix_T>&;
00058 void gen_from_pml_qml(const std::vector<Matrix_T>& old, const signature_t sig);
00059 void gen_from_pm4_qp4(const std::vector<Matrix_T>& old, const signature_t sig);
00060 void gen_from_pp4_qm4(const std::vector<Matrix_T>& old, const signature_t sig);
00061 void gen_from_qp1_pml(const std::vector<Matrix_T>& old, const signature_t sig);
00062 }
00063 }
00064 }
```



```

00075 friend class friend_for_private_destructor;
00076 // Enforce singleton
00077 // Reference: A. Alexandrescu, "Modern C++ Design", Chapter 6
00078 generator_table() = default;
00079 ~generator_table() = default;
00080 public:
00081 generator_table(const generator_table&) = delete;
00082 auto operator= (const generator_table&) -> generator_table& = delete;
00083 };
00084
00086 static const std::array<index_t, 8> offset_to_super = {0,-1, 0,-1,-2, 3, 2, 1};
00087
00088 } }
00089 #endif // _GLUCAT_GENERATION_H

```

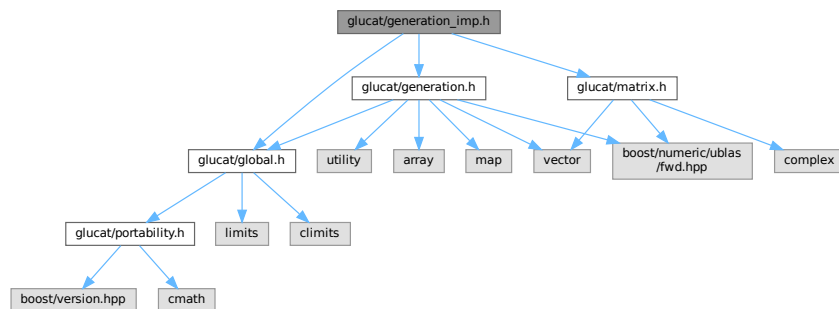
## 7.15 glucat/generation\_imp.h File Reference

```

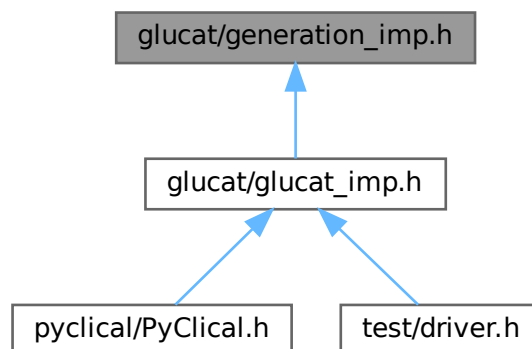
#include "glucat/global.h"
#include "glucat/generation.h"
#include "glucat/matrix.h"

```

Include dependency graph for generation\_imp.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [glucat](#)
- namespace [glucat::gen](#)

## 7.16 generation\_imp.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_GENERATION_IMP_H
00002 #define _GLUCAT_GENERATION_IMP_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 generation_imp.h : Implement functions for generation of the matrix representation
00006 -----
00007 begin : Wed Jan 23 2002
00008 copyright : (C) 2002-2012 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 #include "glucat/global.h"
00035 #include "glucat/generation.h"
00036 #include "glucat/matrix.h"
00037
00038 namespace glucat { namespace gen
00039 {
00040 // References for algorithms:
00041 // [M]: Scott Meyers, "Effective C++" Second Edition, Addison-Wesley, 1998.
00042 // [P]: Ian R. Porteous, "Clifford algebras and the classical groups", Cambridge UP, 1995.
00043 // [L]: Pertti Lounesto, "Clifford algebras and spinors", Cambridge UP, 1997.
00044
00045 // Reference: [M] Item 47
00046 template< class Matrix_T >
00047 auto
00048 generator_table<Matrix_T>::
00049 generator() -> generator_table<Matrix_T>&
00050 { static generator_table<Matrix_T> g; return g;}
00051
00052 // Reference: [P] Table 15.27, p 133
00053 template< class Matrix_T >
00054 inline
00055 auto
00056 generator_table<Matrix_T>::
00057 operator() (const index_t p, const index_t q) -> const Matrix_T*
00058 {
00059 const auto bott = pos_mod(p-q, 8);
00060 switch(bott)
00061 {
00062 case 0:
00063 case 2:
00064 // Construct generators
00065 return &(gen_vector(p, q)[q]);
00066 default:
00067 // Select generators from the vector for a larger frame
00068 const auto super_p = p + std::max(offset_to_super[bott], index_t(0));
00069 const auto super_q = q - std::min(offset_to_super[bott], index_t(0));
00070 return &(gen_vector(super_p, super_q)[super_q]);
00071 }
00072 }
00073 }
00074 }

```

```

00075
00076 template< class Matrix_T >
00077 auto
00078 generator_table<Matrix_T>::
00079 gen_vector(const index_t p, const index_t q) -> const std::vector<Matrix_T>&
00080 {
00081 using result_t = std::vector<Matrix_T>;
00082 const auto card = p + q;
00083 const auto bias = p - q;
00084 const auto bott = pos_mod(bias, 8);
00085 const auto sig = signature_t(p, q);
00086 if (this->find(sig) == this->end())
00087 switch(bott)
00088 {
00089 case 0:
00090 if (bias < 0)
00091 // Construct generators for p,q given generators for p+4,q-4
00092 gen_from_pp4_qm4(gen_vector(p+4, q-4), sig);
00093 else if (bias > 0)
00094 // Construct generators for p,q given generators for p-4,q+4
00095 gen_from_pm4_qp4(gen_vector(p-4, q+4), sig);
00096 else if (card == 0)
00097 { // Base case. Save a generator vector containing one matrix, size 1.
00098 auto result = result_t(1, matrix::unit<Matrix_T>(1));
00099 this->insert(make_pair(sig, result));
00100 }
00101 else
00102 // Construct generators for p,q given generators for p-1,q-1
00103 gen_from_pml_qml(gen_vector(p-1, q-1), sig);
00104 break;
00105 case 2:
00106 if (bias < 2)
00107 // Construct generators for p,q given generators for p+4,q-4
00108 gen_from_pp4_qm4(gen_vector(p+4, q-4), sig);
00109 else if (bias > 2)
00110 // Construct generators for p,q given generators for p-4,q+4
00111 gen_from_pm4_qp4(gen_vector(p-4, q+4), sig);
00112 else
00113 // Construct generators for p,q given generators for q+1,p-1
00114 gen_from_qp1_pml(gen_vector(q+1, p-1), sig);
00115 break;
00116 default:
00117 break;
00118 }
00119 return (*this)[sig];
00120 }
00121
00122 // Reference: [P] Proposition 15.17, p 131
00123 template< class Matrix_T >
00124 void
00125 generator_table<Matrix_T>::
00126 gen_from_pml_qml(const std::vector<Matrix_T>& old, const signature_t sig)
00127 {
00128 const auto new_size = old.size() + 2;
00129 using size_t = decltype(new_size);
00130 using result_t = std::vector<Matrix_T>;
00131 auto result = result_t(new_size);
00132
00133 const auto old_dim = old[0].size1();
00134 const auto& eye = matrix::unit<Matrix_T>(old_dim);
00135
00136 auto neg = Matrix_T(2,2,2);
00137 neg(0,1) = -1;
00138 neg(1,0) = 1;
00139
00140 auto pos = neg;
00141 pos(0,1) = 1;
00142
00143 auto dup = Matrix_T(2,2,2);
00144 dup(0,0) = 1;
00145 dup(1,1) = -1;
00146
00147 result[0] = matrix::mono_kron(neg, eye);
00148 for (auto
00149 k = size_t(1);
00150 k != new_size-1;
00151 ++k)
00152 result[k] = matrix::mono_kron(dup, old[k-1]);
00153 result[new_size-1] = matrix::mono_kron(pos, eye);
00154
00155 // Save the resulting generator array.
00156 this->insert(make_pair(sig, result));
00157 }
00158
00159 // Reference: [L] 16.4 Periodicity of 8, p216
00160 template< class Matrix_T >
00161 void

```

```

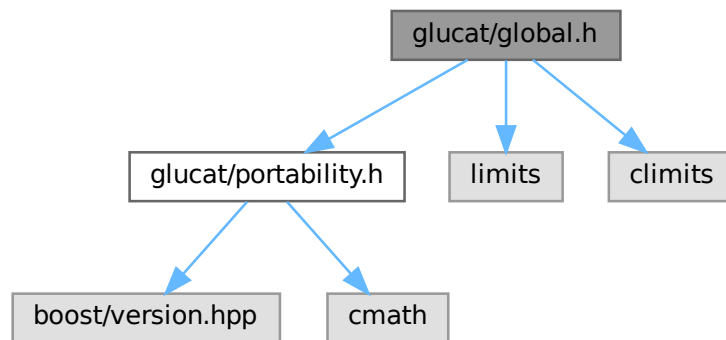
00165 generator_table<Matrix_T>::
00166 gen_from_pm4_qp4(const std::vector<Matrix_T>& old, const signature_t sig)
00167 {
00168 const auto old_size = old.size();
00169 using size_t = decltype(old_size);
00170 using result_t = std::vector<Matrix_T>;
00171 auto result = result_t(old_size);
00172
00173 auto h = old[0];
00174 for (auto
00175 k = size_t(1);
00176 k != size_t(4);
00177 ++k)
00178 h = matrix::mono_prod(old[k], h);
00179
00180 for (auto
00181 k = size_t(0);
00182 k != old_size-4;
00183 ++k)
00184 result[k] = old[k+4];
00185 for (auto
00186 k = old_size-4;
00187 k != old_size;
00188 ++k)
00189 result[k] = matrix::mono_prod(old[k+4-old_size], h);
00190 // Save the resulting generator array.
00191 this->insert(make_pair(sig, result));
00192 }
00193
00194 // Reference: [L] 16.4 Periodicity of 8, p216
00195 template< class Matrix_T >
00196 void
00197 generator_table<Matrix_T>::
00198 gen_from_pp4_qm4(const std::vector<Matrix_T>& old, const signature_t sig)
00199 {
00200 const auto old_size = old.size();
00201 using size_t = decltype(old_size);
00202 using result_t = std::vector<Matrix_T>;
00203 auto result = result_t(old_size);
00204
00205 auto h = old[old_size-1];
00206 for (auto
00207 k = size_t(1);
00208 k != size_t(4);
00209 ++k)
00210 h = matrix::mono_prod(old[old_size-1-k], h);
00211
00212 for (auto
00213 k = size_t(0);
00214 k != size_t(4);
00215 ++k)
00216 result[k] = matrix::mono_prod(old[k+old_size-4], h);
00217 for (auto
00218 k = size_t(4);
00219 k != old_size;
00220 ++k)
00221 result[k] = old[k-4];
00222 // Save the resulting generator array.
00223 this->insert(make_pair(sig, result));
00224 }
00225
00226 // Reference: [P] Proposition 15.20, p 131
00227 template< class Matrix_T >
00228 void
00229 generator_table<Matrix_T>::
00230 gen_from_qp1_pml(const std::vector<Matrix_T>& old, const signature_t sig)
00231 {
00232 const auto old_size = old.size();
00233 using size_t = decltype(old_size);
00234 using result_t = std::vector<Matrix_T>;
00235 auto result = result_t(old_size);
00236
00237 const auto& h = old[old_size-1];
00238 for (auto
00239 k = size_t(0);
00240 k != old_size-1;
00241 ++k)
00242 result[k] = matrix::mono_prod(old[old_size-2-k], h);
00243 result[old_size-1] = h;
00244
00245 // Save the resulting generator array.
00246 this->insert(make_pair(sig, result));
00247 }
00248 } }
00249
00250 #endif // _GLUCAT_GENERATION_IMP_H

```

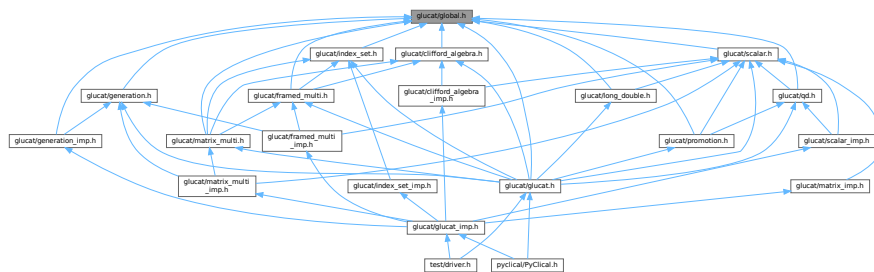
## 7.17 glucat/global.h File Reference

```
#include "glucat/portability.h"
#include <limits>
#include <climits>
```

Include dependency graph for global.h:



This graph shows which files directly or indirectly include this file:



### Classes

- struct `glucat::CTAssertion< true >`
- class `glucat::compare_types< LHS_T, RHS_T >`  
*Type comparison.*
- class `glucat::compare_types< T, T >`
- class `glucat::bool_to_type< truth_value >`  
*Bool to type.*

### Namespaces

- namespace `glucat`

## Macros

- `#define _GLUCAT_CTAssert(expr, msg)`

## Typedefs

- using `glucat::index_t` = int  
*Size of index\_t should be enough to represent LO, HI.*
- using `glucat::set_value_t` = unsigned long  
*Size of set\_value\_t should be enough to contain index\_set<LO,HI>*

## Functions

- `glucat::_GLUCAT_CTAssert` (std::numeric\_limits< unsigned char >::radix==2, CannotDetermineBitsPerChar) const `index_t` BITS\_PER\_CHAR  
*If radix of unsigned char is not 2, we can't easily determine number of bits from sizeof.*
- `glucat::_GLUCAT_CTAssert` (\_GLUCAT\_BITS\_PER\_ULONG==BITS\_PER\_SET\_VALUE, BitsPerULongDoesNotMatchSetValueT) const `index_t` DEFAULT\_LO  
*Default lowest index in an index set.*
- template<typename LHS\_T, typename RHS\_T >  
auto `glucat::pos_mod` (LHS\_T lhs, RHS\_T rhs) -> LHS\_T  
*Modulo function which works reliably for lhs < 0.*

## Variables

- const double `glucat::MS_PER_S` = 1000.0  
*Timing constant: deprecated here - moved to [test/timing.h](#).*
- const `index_t` `glucat::BITS_PER_SET_VALUE` = std::numeric\_limits<set\_value\_t>::digits  
*Number of bits in set\_value\_t.*
- const `index_t` `glucat::DEFAULT_HI` = `index_t`(BITS\_PER\_SET\_VALUE / 2)  
*Default highest index in an index set.*

## 7.17.1 Macro Definition Documentation

### 7.17.1.1 \_GLUCAT\_CTAssert

```
#define _GLUCAT_CTAssert(
 expr,
 msg)
```

#### Value:

```
namespace { struct msg { glucat::CTAssertion<(expr)> ERROR_##msg; }; }
```

Definition at line 48 of file [global.h](#).

## 7.18 global.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_GLOBAL_H
00002 #define _GLUCAT_GLOBAL_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 global.h : Global declarations
00006
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2021 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
00030 *****/
00031 See also Arvind Raja's original header comments and references in glucat.h
00032 *****/
00033
00034 #include "glucat/portability.h"
00035
00036 #include <limits>
00037 #include <climits>
00038
00039 namespace glucat
00040 {
00041 // References:
00042 // [AA]: A. Alexandrescu, "Modern C++ Design", Addison-Wesley, 2001.
00043
00044 // Reference: [AA], p. 25
00045 template<bool> struct CTAAssertion;
00046 template<> struct CTAAssertion<true> { };
00047 #define _GLUCAT_CTAssert(expr, msg) \
00048 namespace { struct msg { glucat::CTAAssertion<(expr)> ERROR_##msg; }; }
00049
00050
00051 // Reference: [AA], pp. 34--37
00052 template < typename LHS_T, typename RHS_T >
00053 class compare_types
00054 {
00055 public:
00056 enum { are_same = false };
00057 };
00058 template < typename T >
00059 class compare_types<T, T>
00060 {
00061 public:
00062 enum { are_same = true };
00063 };
00064
00065
00066 // Reference: [AA], 2.4, p. 29
00067 template< bool truth_value >
00068 class bool_to_type
00069 {
00070 private:
00071 enum { value = truth_value };
00072 };
00073
00074
00075 // Global types which determine sizes
00076 using index_t = int;
00077 using set_value_t = unsigned long;
00078
00079
00080
00081 // Global constants
00082 const double MS_PER_S = 1000.0;
00083
00084
00085 // Constants which determine sizes
00086
00087 // Bits per unsigned long
00088 #if (ULONG_MAX == (4294967295UL))

```

```

00089 #define _GLUCAT_BITS_PER_ULONG 32
00090 #elif (ULONG_MAX == (18446744073709551615UL))
00091 #define _GLUCAT_BITS_PER_ULONG 64
00092 #elif defined(__WORDSIZE)
00093 #define _GLUCAT_BITS_PER_ULONG __WORDSIZE
00094 #endif
00095
00097 _GLUCAT_CTAssert(std::numeric_limits<unsigned char>::radix == 2, CannotDetermineBitsPerChar)
00098
00099
00100 const index_t BITS_PER_CHAR = std::numeric_limits<unsigned char>::digits;
00101
00103 const index_t BITS_PER_SET_VALUE = std::numeric_limits<set_value_t>::digits;
00104
00105 _GLUCAT_CTAssert(_GLUCAT_BITS_PER_ULONG == BITS_PER_SET_VALUE, BitsPerUlongDoesNotMatchSetValueT)
00106
00107 // Constants which are determined by size
00109 const index_t DEFAULT_LO = -index_t(BITS_PER_SET_VALUE / 2);
00111 const index_t DEFAULT_HI = index_t(BITS_PER_SET_VALUE / 2);
00112
00114 template< typename LHS_T, typename RHS_T >
00115 inline
00116 auto
00117 pos_mod(LHS_T lhs, RHS_T rhs) -> LHS_T
00118 { return lhs > 0? lhs % rhs : (-lhs) % rhs == 0 ? 0 : rhs - (-lhs) % rhs; }
00119
00120 }
00121 #endif // _GLUCAT_GLOBAL_H

```

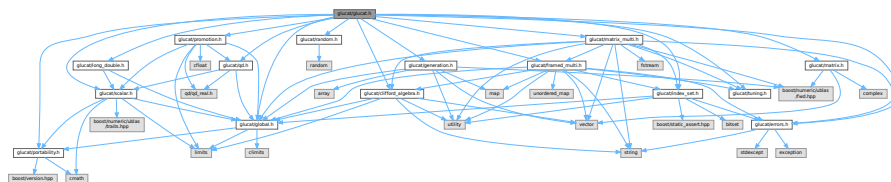
## 7.19 glucat/glucat.h File Reference

```

#include "glucat/portability.h"
#include "glucat/global.h"
#include "glucat/errors.h"
#include "glucat/index_set.h"
#include "glucat/scalar.h"
#include "glucat/long_double.h"
#include "glucat/qd.h"
#include "glucat/promotion.h"
#include "glucat/random.h"
#include "glucat/clifford_algebra.h"
#include "glucat/tuning.h"
#include "glucat/framed_multi.h"
#include "glucat/generation.h"
#include "glucat/matrix.h"
#include "glucat/matrix_multi.h"

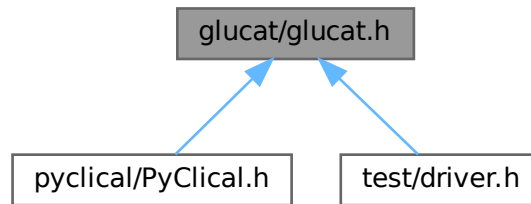
```

Include dependency graph for glucat.h:





This graph shows which files directly or indirectly include this file:



## 7.20 glucat.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_GLUCAT_H
00002 #define _GLUCAT_GLUCAT_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 glucat.h : Organize GluCat header files for applications
00006 -----
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2021 by Paul C. Leopardi
00009 ****
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 ****
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 ****
00031 Arvind Raja's original header comments and references follow.
00032 ****
00033 // clifford algebra package, Arvind.Raja@hut.fi
00034 // ref: Press et.al. "Numerical Recipes in C", 2nd ed., C.U.P., 1992.
00035 // ref: LEDA, v 3.0, Stefan N\aher, Max-Planck-Institut f\ur Informatik
00036 // ref: Stroustrup B., "The C++ Programming Language", 2nd ed.,
00037 // Addison-Wesley, 1991.
00038 // ref: R. Sedgewick, "Algorithms in C++", Addison-Wesley, 1992.
00039 // ref: S. Meyers, "Effective C++ ", Addison-Wesley, 1992.
00040 *****/
00041
00042 #include "glucat/portability.h"
00043
00044 #include "glucat/global.h"
00045
00046 #include "glucat/errors.h"
00047
00048 #include "glucat/index_set.h"
00049
00050 #include "glucat/scalar.h"
00051
00052 #include "glucat/long_double.h"
00053
00054 #include "glucat/qd.h"

```

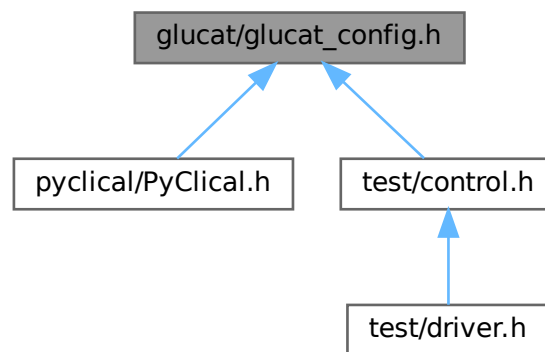
```

00055
00056 #include "glucat/promotion.h"
00057
00058 #include "glucat/random.h"
00059
00060 #include "glucat/clifford_algebra.h"
00061
00062 #include "glucat/tuning.h"
00063
00064 #include "glucat/framed_multi.h"
00065
00066 #include "glucat/generation.h"
00067
00068 #include "glucat/matrix.h"
00069
00070 #include "glucat/matrix_multi.h"
00071
00072 #endif // _GLUCAT_GLUCAT_H

```

## 7.21 glucat/glucat\_config.h File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- #define GLUCAT\_HAVE\_CXX11 1
- #define GLUCAT\_HAVE\_INTTYPES\_H 1
- #define GLUCAT\_HAVE\_STDINT\_H 1
- #define GLUCAT\_HAVE\_STDIO\_H 1
- #define GLUCAT\_HAVE\_STDLIB\_H 1
- #define GLUCAT\_HAVE\_STRINGS\_H 1
- #define GLUCAT\_HAVE\_STRING\_H 1
- #define GLUCAT\_HAVE\_SYS\_STAT\_H 1
- #define GLUCAT\_HAVE\_SYS\_TYPES\_H 1
- #define GLUCAT\_HAVE\_UNISTD\_H 1
- #define GLUCAT\_PACKAGE "glucat"
- #define GLUCAT\_PACKAGE\_BUGREPORT ""
- #define GLUCAT\_PACKAGE\_NAME "glucat"
- #define GLUCAT\_PACKAGE\_STRING "glucat 0.12.0"
- #define GLUCAT\_PACKAGE\_TARNAME "glucat"

- `#define GLUCAT_PACKAGE_URL ""`
- `#define GLUCAT_PACKAGE_VERSION "0.12.0"`
- `#define GLUCAT_STDC_HEADERS 1`
- `#define GLUCAT_VERSION "0.12.0"`

## 7.21.1 Macro Definition Documentation

### 7.21.1.1 GLUCAT\_HAVE\_CXX11

```
#define GLUCAT_HAVE_CXX11 1
```

Definition at line 20 of file [glucat\\_config.h](#).

### 7.21.1.2 GLUCAT\_HAVE\_INTTYPES\_H

```
#define GLUCAT_HAVE_INTTYPES_H 1
```

Definition at line 28 of file [glucat\\_config.h](#).

### 7.21.1.3 GLUCAT\_HAVE\_STDINT\_H

```
#define GLUCAT_HAVE_STDINT_H 1
```

Definition at line 39 of file [glucat\\_config.h](#).

### 7.21.1.4 GLUCAT\_HAVE\_STDIO\_H

```
#define GLUCAT_HAVE_STDIO_H 1
```

Definition at line 44 of file [glucat\\_config.h](#).

### 7.21.1.5 GLUCAT\_HAVE\_STDLIB\_H

```
#define GLUCAT_HAVE_STDLIB_H 1
```

Definition at line 49 of file [glucat\\_config.h](#).

### 7.21.1.6 GLUCAT\_HAVE\_STRING\_H

```
#define GLUCAT_HAVE_STRING_H 1
```

Definition at line 59 of file [glucat\\_config.h](#).

#### 7.21.1.7 GLUCAT\_HAVE\_STRINGS\_H

```
#define GLUCAT_HAVE_STRINGS_H 1
```

Definition at line 54 of file [glucat\\_config.h](#).

#### 7.21.1.8 GLUCAT\_HAVE\_SYS\_STAT\_H

```
#define GLUCAT_HAVE_SYS_STAT_H 1
```

Definition at line 64 of file [glucat\\_config.h](#).

#### 7.21.1.9 GLUCAT\_HAVE\_SYS\_TYPES\_H

```
#define GLUCAT_HAVE_SYS_TYPES_H 1
```

Definition at line 69 of file [glucat\\_config.h](#).

#### 7.21.1.10 GLUCAT\_HAVE\_UNISTD\_H

```
#define GLUCAT_HAVE_UNISTD_H 1
```

Definition at line 74 of file [glucat\\_config.h](#).

#### 7.21.1.11 GLUCAT\_PACKAGE

```
#define GLUCAT_PACKAGE "glucat"
```

Definition at line 79 of file [glucat\\_config.h](#).

#### 7.21.1.12 GLUCAT\_PACKAGE\_BUGREPORT

```
#define GLUCAT_PACKAGE_BUGREPORT ""
```

Definition at line 84 of file [glucat\\_config.h](#).

#### 7.21.1.13 GLUCAT\_PACKAGE\_NAME

```
#define GLUCAT_PACKAGE_NAME "glucat"
```

Definition at line 89 of file [glucat\\_config.h](#).

Referenced by [glucat::control\\_t::control\\_t\(\)](#).

#### 7.21.1.14 GLUCAT\_PACKAGE\_STRING

```
#define GLUCAT_PACKAGE_STRING "glucat 0.12.0"
```

Definition at line 94 of file [glucat\\_config.h](#).

#### 7.21.1.15 GLUCAT\_PACKAGE\_TARNAME

```
#define GLUCAT_PACKAGE_TARNAME "glucat"
```

Definition at line 99 of file [glucat\\_config.h](#).

#### 7.21.1.16 GLUCAT\_PACKAGE\_URL

```
#define GLUCAT_PACKAGE_URL ""
```

Definition at line 104 of file [glucat\\_config.h](#).

#### 7.21.1.17 GLUCAT\_PACKAGE\_VERSION

```
#define GLUCAT_PACKAGE_VERSION "0.12.0"
```

Definition at line 109 of file [glucat\\_config.h](#).

#### 7.21.1.18 GLUCAT\_STDC\_HEADERS

```
#define GLUCAT_STDC_HEADERS 1
```

Definition at line 116 of file [glucat\\_config.h](#).

#### 7.21.1.19 GLUCAT\_VERSION

```
#define GLUCAT_VERSION "0.12.0"
```

Definition at line 121 of file [glucat\\_config.h](#).

Referenced by [glucat::control\\_t::control\\_t\(\)](#).

## 7.22 glucat\_config.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_GLUCAT_CONFIG_H
00002 #define _GLUCAT_GLUCAT_CONFIG_H 1
00003
00004 /* glucat/glucat_config.h. Generated automatically at end of configure. */
00005 /* config.h. Generated from config.h.in by configure. */
00006 /* config.h.in. Generated from configure.ac by autoheader. */
00007
00008 /* Define to dummy 'main' function (if any) required to link to the Fortran
00009 libraries. */
00010 /* #undef F77_DUMMY_MAIN */
00011
00012 /* Define if F77 and FC dummy 'main' functions are identical. */
00013 /* #undef FC_DUMMY_MAIN_EQ_F77 */
00014
00015 /* Define if you have a BLAS library. */
00016 /* #undef HAVE_BLAS */
00017
00018 /* define if the compiler supports basic C++11 syntax */
00019 #ifndef GLUCAT_HAVE_CXX11
00020 #define GLUCAT_HAVE_CXX11 1
00021 #endif
00022
00023 /* define if the compiler supports basic C++14 syntax */
00024 /* #undef HAVE_CXX14 */
00025
00026 /* Define to 1 if you have the <inttypes.h> header file. */
00027 #ifndef GLUCAT_HAVE_INTTYPES_H
00028 #define GLUCAT_HAVE_INTTYPES_H 1
00029 #endif
00030
00031 /* Define if you have LAPACK library. */
00032 /* #undef HAVE_LAPACK */
00033
00034 /* Define to 1 if you have the 'lmf' library (-lmf). */
00035 /* #undef HAVE_LIBIMF */
00036
00037 /* Define to 1 if you have the <stdint.h> header file. */
00038 #ifndef GLUCAT_HAVE_STDINT_H
00039 #define GLUCAT_HAVE_STDINT_H 1
00040 #endif
00041
00042 /* Define to 1 if you have the <stdio.h> header file. */
00043 #ifndef GLUCAT_HAVE_STDIO_H
00044 #define GLUCAT_HAVE_STDIO_H 1
00045 #endif
00046
00047 /* Define to 1 if you have the <stdlib.h> header file. */
00048 #ifndef GLUCAT_HAVE_STDLIB_H
00049 #define GLUCAT_HAVE_STDLIB_H 1
00050 #endif
00051
00052 /* Define to 1 if you have the <strings.h> header file. */
00053 #ifndef GLUCAT_HAVE_STRINGS_H
00054 #define GLUCAT_HAVE_STRINGS_H 1
00055 #endif
00056
00057 /* Define to 1 if you have the <string.h> header file. */
00058 #ifndef GLUCAT_HAVE_STRING_H
00059 #define GLUCAT_HAVE_STRING_H 1
00060 #endif
00061
00062 /* Define to 1 if you have the <sys/stat.h> header file. */
00063 #ifndef GLUCAT_HAVE_SYS_STAT_H
00064 #define GLUCAT_HAVE_SYS_STAT_H 1
00065 #endif
00066
00067 /* Define to 1 if you have the <sys/types.h> header file. */
00068 #ifndef GLUCAT_HAVE_SYS_TYPES_H
00069 #define GLUCAT_HAVE_SYS_TYPES_H 1
00070 #endif
00071
00072 /* Define to 1 if you have the <unistd.h> header file. */
00073 #ifndef GLUCAT_HAVE_UNISTD_H
00074 #define GLUCAT_HAVE_UNISTD_H 1
00075 #endif
00076
00077 /* Name of package */
00078 #ifndef GLUCAT_PACKAGE
00079 #define GLUCAT_PACKAGE "glucat"
00080 #endif
00081
00082 /* Define to the address where bug reports for this package should be sent. */

```

```

00083 #ifndef GLUCAT_PACKAGE_BUGREPORT
00084 #define GLUCAT_PACKAGE_BUGREPORT ""
00085 #endif
00086
00087 /* Define to the full name of this package. */
00088 #ifndef GLUCAT_PACKAGE_NAME
00089 #define GLUCAT_PACKAGE_NAME "glucat"
00090 #endif
00091
00092 /* Define to the full name and version of this package. */
00093 #ifndef GLUCAT_PACKAGE_STRING
00094 #define GLUCAT_PACKAGE_STRING "glucat 0.12.0"
00095 #endif
00096
00097 /* Define to the one symbol short name of this package. */
00098 #ifndef GLUCAT_PACKAGE_TARNAME
00099 #define GLUCAT_PACKAGE_TARNAME "glucat"
00100 #endif
00101
00102 /* Define to the home page for this package. */
00103 #ifndef GLUCAT_PACKAGE_URL
00104 #define GLUCAT_PACKAGE_URL ""
00105 #endif
00106
00107 /* Define to the version of this package. */
00108 #ifndef GLUCAT_PACKAGE_VERSION
00109 #define GLUCAT_PACKAGE_VERSION "0.12.0"
00110 #endif
00111
00112 /* Define to 1 if all of the C90 standard headers exist (not just the ones
00113 required in a freestanding environment). This macro is provided for
00114 backward compatibility; new code need not use it. */
00115 #ifndef GLUCAT_STDC_HEADERS
00116 #define GLUCAT_STDC_HEADERS 1
00117 #endif
00118
00119 /* Version number of package */
00120 #ifndef GLUCAT_VERSION
00121 #define GLUCAT_VERSION "0.12.0"
00122 #endif
00123
00124 /* once: _GLUCAT_GLUCAT_CONFIG_H */
00125 #endif

```

## 7.23 glucat/glucat\_imp.h File Reference

```

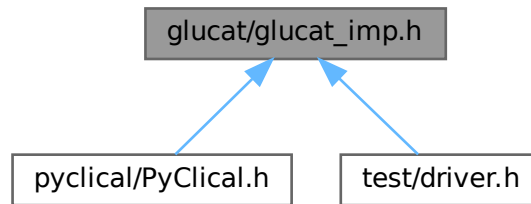
#include "glucat/errors_imp.h"
#include "glucat/index_set_imp.h"
#include "glucat/scalar_imp.h"
#include "glucat/clifford_algebra_imp.h"
#include "glucat/random.h"
#include "glucat/framed_multi_imp.h"
#include "glucat/matrix_imp.h"
#include "glucat/generation_imp.h"
#include "glucat/matrix_multi_imp.h"

```

Include dependency graph for glucat\_imp.h:



This graph shows which files directly or indirectly include this file:



## 7.24 glucat\_imp.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_GLUCAT_IMP_H
00002 #define _GLUCAT_GLUCAT_IMP_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 glucat_imp.h : Organize GluCat template definitions which cannot be compiled separately
00006 -----
00007 begin : Sun 2001-12-25
00008 copyright : (C) 2001-2012 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 For Arvind Raja's original header comments, see glucat.h
00032 *****/
00033
00034 // Template definitions which cannot be compiled separately
00035
00036 #include "glucat/errors_imp.h"
00037
00038 #include "glucat/index_set_imp.h"
00039
00040 #include "glucat/scalar_imp.h"
00041
00042 #include "glucat/clifford_algebra_imp.h"
00043
00044 #include "glucat/random.h"
00045
00046 #include "glucat/framed_multi_imp.h"
00047
00048 #include "glucat/matrix_imp.h"
00049
00050 #include "glucat/generation_imp.h"
00051
00052 #include "glucat/matrix_multi_imp.h"
00053
00054 #endif // _GLUCAT_GLUCAT_IMP_H

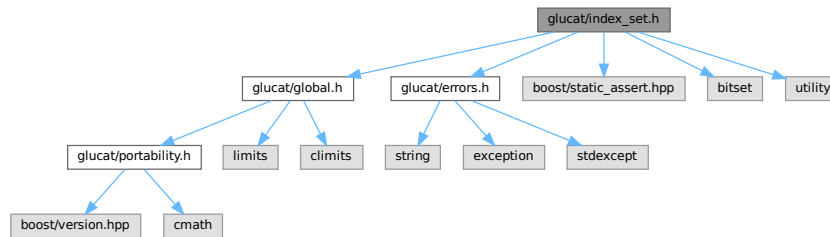
```



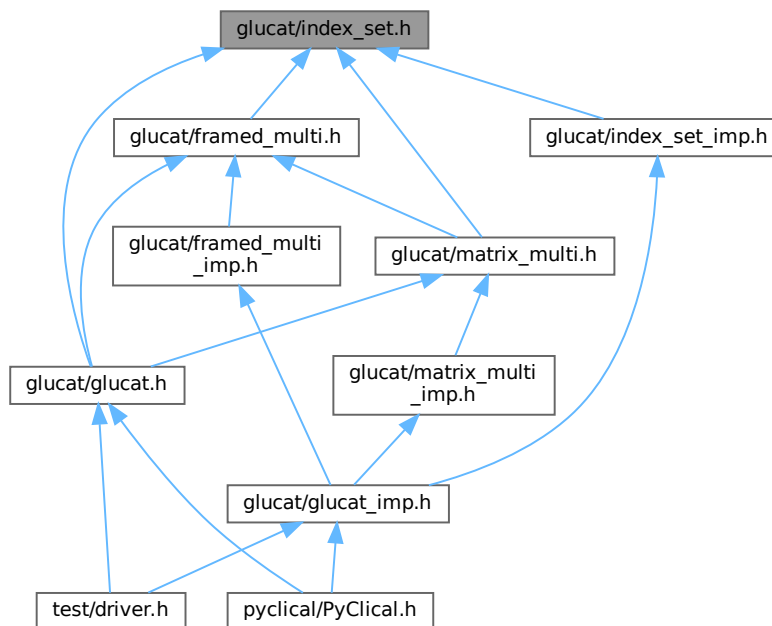
## 7.25 glucat/index\_set.h File Reference

```
#include "glucat/global.h"
#include "glucat/errors.h"
#include <boost/static_assert.hpp>
#include <bitset>
#include <utility>
```

Include dependency graph for index\_set.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [glucat::index\\_set< LO, HI >](#)  
*Index set class based on std::bitset<> in Gnu standard C++ library.*
- class [glucat::index\\_set< LO, HI >::reference](#)  
*Index set member reference.*

## Namespaces

- namespace [glucat](#)

## Functions

- `template<const index\_t LO, const index\_t HI>`  
`auto glucat::operator^ (const index\_set< LO, HI > &lhs, const index\_set< LO, HI > &rhs) -> const index\_set< LO, HI >`  
*Symmetric set difference: exclusive or.*
- `template<const index\_t LO, const index\_t HI>`  
`auto glucat::operator& (const index\_set< LO, HI > &lhs, const index\_set< LO, HI > &rhs) -> const index\_set< LO, HI >`  
*Set intersection: and.*
- `template<const index\_t LO, const index\_t HI>`  
`auto glucat::operator| (const index\_set< LO, HI > &lhs, const index\_set< LO, HI > &rhs) -> const index\_set< LO, HI >`  
*Set union: or.*
- `template<const index\_t LO, const index\_t HI>`  
`auto glucat::compare (const index\_set< LO, HI > &a, const index\_set< LO, HI > &b) -> int`  
*"lexicographic compare" eg. {3,4,5} is less than {3,7,8}*
- `glucat::GLUCAT\_CTAssert (sizeof(set\_value\_t) >=sizeof(std::bitset< DEFAULT\_HI-DEFAULT\_LO >), Default_index_set_too_big_for_value) template< const index\_t LO`  
*Size of [set\\_value\\_t](#) should be enough to contain [bitset](#)<[DEFAULT\\_HI](#)-[DEFAULT\\_LO](#)>*
- `const index\_t HI auto glucat::operator<< (std::ostream &os, const index\_set< LO, HI > &ist) -> std::ostream &`
- `template<const index\_t LO, const index\_t HI>`  
`auto glucat::operator>> (std::istream &s, index\_set< LO, HI > &ist) -> std::istream &`  
*Read in index set.*
- `auto glucat::sign\_of\_square (index\_t j) -> int`  
*Square of generator {j}.*
- `template<const index\_t LO, const index\_t HI>`  
`auto glucat::min\_neg (const index\_set< LO, HI > &ist) -> index\_t`  
*Minimum negative index, or 0 if none.*
- `template<const index\_t LO, const index\_t HI>`  
`auto glucat::max\_pos (const index\_set< LO, HI > &ist) -> index\_t`  
*Maximum positive index, or 0 if none.*

## 7.26 [index\\_set.h](#)

[Go to the documentation of this file.](#)

```
00001 #ifndef _GLUCAT_INDEX_SET_H
00002 #define _GLUCAT_INDEX_SET_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 index_set.h : Declare a class for a set of non-zero integer indices
00006 -----
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2012 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
```

```

00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 #include "glucat/global.h"
00035 #include "glucat/errors.h"
00036
00037 #include <boost/static_assert.hpp>
00038
00039 #include <bitset>
00040 #include <utility>
00041
00042 namespace glucat
00043 {
00044 template<const index_t LO, const index_t HI>
00045 class index_set; // forward
00046
00047 template<const index_t LO, const index_t HI>
00048 auto
00049 operator^ (const index_set<LO,HI>& lhs,
00050 const index_set<LO,HI>& rhs) -> const index_set<LO,HI>;
00051
00052 template<const index_t LO, const index_t HI>
00053 auto
00054 operator& (const index_set<LO,HI>& lhs,
00055 const index_set<LO,HI>& rhs) -> const index_set<LO,HI>;
00056
00057 template<const index_t LO, const index_t HI>
00058 auto
00059 operator| (const index_set<LO,HI>& lhs,
00060 const index_set<LO,HI>& rhs) -> const index_set<LO,HI>;
00061
00062 // -1 if a<b, +1 if a>b, 0 if a==b
00063 template<const index_t LO, const index_t HI>
00064 auto
00065 compare(const index_set<LO,HI>& a, const index_set<LO,HI>& b) -> int;
00066
00067 template<const index_t LO, const index_t HI>
00068 class index_set :
00069 private std::bitset<HI-LO>
00070 {
00071 private:
00072 BOOST_STATIC_ASSERT((LO <= 0) && (0 <= HI) && (LO < HI) && \
00073 (~LO < _GLUCAT_BITS_PER_ULONG) && \
00074 (HI < _GLUCAT_BITS_PER_ULONG) && \
00075 (HI-LO <= _GLUCAT_BITS_PER_ULONG));
00076 public:
00077 using bitset_t = std::bitset<HI - LO>;
00078 using error_t = error<index_set>;
00079
00080 using index_set_t = index_set;
00081 using index_pair_t = std::pair<index_t, index_t>;
00082
00083 static const index_t v_lo = LO;
00084 static const index_t v_hi = HI;
00085
00086 static auto classname() -> const std::string;
00087 index_set() = default;
00088 index_set(const bitset_t bst);
00089 index_set(const index_t idx);
00090 index_set(const set_value_t folded_val, const index_set_t frm, const bool prechecked = false);
00091 index_set(const index_pair_t& range, const bool prechecked = false);
00092 index_set(const std::string& str);
00093
00094 auto operator== (const index_set_t rhs) const -> bool;
00095 auto operator!= (const index_set_t rhs) const -> bool;
00096 auto operator~ () const -> index_set_t;
00097 auto operator^= (const index_set_t rhs) -> index_set_t&;
00098 auto operator&= (const index_set_t rhs) -> index_set_t&;
00099 auto operator|= (const index_set_t rhs) -> index_set_t&;
00100 auto operator[] (const index_t idx) const -> bool;
00101 auto test(const index_t idx) const -> bool;
00102 auto set() -> index_set_t&;
00103 auto set(const index_t idx) -> index_set_t&;
00104 auto set(const index_t idx, const int val) -> index_set_t&;

```

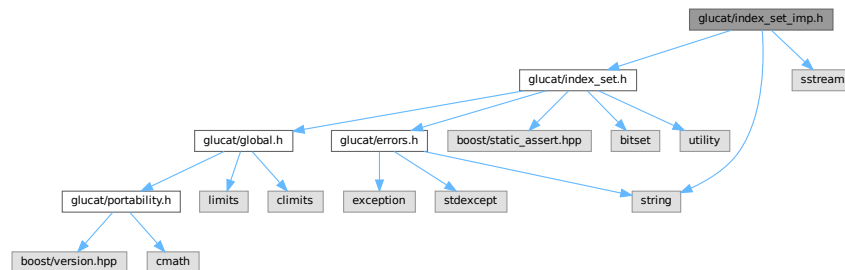
```

00127 auto reset() -> index_set_t&;
00129 auto reset(const index_t idx) -> index_set_t&;
00131 auto flip() -> index_set_t&;
00133 auto flip(const index_t idx) -> index_set_t&;
00135 auto count() const -> index_t;
00137 auto count_neg() const -> index_t;
00139 auto count_pos() const -> index_t;
00141 auto min() const -> index_t;
00143 auto max() const -> index_t;
00144
00145 // Functions which support Clifford algebra operations
00147 auto operator< (const index_set_t rhs) const -> bool;
00149 auto is_contiguous () const -> bool;
00151 auto fold () const -> const index_set_t;
00153 auto fold (const index_set_t frm, const bool prechecked = false) const -> const
index_set_t;
00155 auto unfold (const index_set_t frm, const bool prechecked = false) const -> const
index_set_t;
00157 auto value_of_fold (const index_set_t frm) const -> set_value_t;
00159 auto sign_of_mult (const index_set_t ist) const -> int;
00161 auto sign_of_square () const -> int;
00162
00164 auto hash_fn () const -> size_t;
00165
00166 // Friends
00167 friend auto operator^<> (const index_set_t& lhs, const index_set_t& rhs) -> const index_set_t;
00168 friend auto operator&<> (const index_set_t& lhs, const index_set_t& rhs) -> const index_set_t;
00169 friend auto operator|<> (const index_set_t& lhs, const index_set_t& rhs) -> const index_set_t;
00170 friend auto compare<> (const index_set_t& lhs, const index_set_t& rhs) -> int;
00171
00172 // Member reference:
00173 class reference;
00174 friend class reference;
00175
00177 class reference {
00178 friend class index_set;
00179
00180 public:
00182 reference() = delete;
00183 reference (index_set_t& ist, index_t idx);
00184 ~reference () = default;
00186 auto operator== (const reference& c_j) const -> bool;
00188 auto operator= (const bool x) -> reference&;
00190 auto operator= (const reference& c_j) -> reference&;
00192 auto operator~ () const -> bool;
00194 operator bool () const;
00196 auto flip() -> reference&;
00197
00198 private:
00199 index_set_t* m_pst;
00200 index_t m_idx;
00201 };
00203 auto operator[](index_t idx) -> reference;
00204 private:
00206 auto lex_less_than (const index_set_t rhs) const -> bool;
00207 };
00208
00210 _GLUCAT_CTAssert(sizeof(set_value_t) >= sizeof(std::bitset<DEFAULT_HI-DEFAULT_LO>),
00211 Default_index_set_too_big_for_value)
00212
00213 // non-members
00214
00216 template<const index_t LO, const index_t HI>
00217 auto
00218 operator<< (std::ostream& os, const index_set<LO,HI>& ist) -> std::ostream&;
00219
00221 template<const index_t LO, const index_t HI>
00222 auto
00223 operator>> (std::istream& s, index_set<LO,HI>& ist) -> std::istream&;
00224
00225 // Functions which support Clifford algebra operations
00227 auto sign_of_square(index_t j) -> int;
00228
00230 template<const index_t LO, const index_t HI>
00231 auto
00232 min_neg(const index_set<LO,HI>& ist) -> index_t;
00233
00235 template<const index_t LO, const index_t HI>
00236 auto
00237 max_pos(const index_set<LO,HI>& ist) -> index_t;
00238 }
00239 #endif // _GLUCAT_INDEX_SET_H

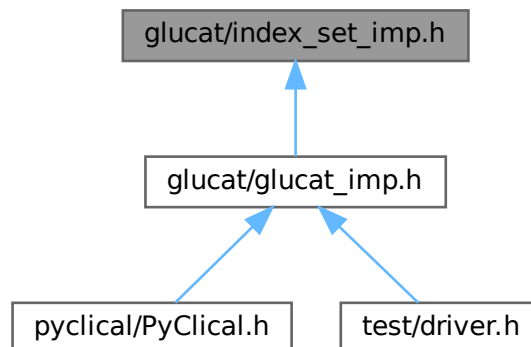
```

## 7.27 glucat/index\_set\_imp.h File Reference

```
#include "glucat/index_set.h"
#include <string>
#include <sstream>
Include dependency graph for index_set_imp.h:
```



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [glucat](#)

### Functions

- `template<const index\_t LO, const index\_t HI>`  
`auto glucat::operator^ (const index\_set< LO, HI > &lhs, const index\_set< LO, HI > &rhs) -> const index\_set< LO, HI >`  
*Symmetric set difference: exclusive or.*
- `template<const index\_t LO, const index\_t HI>`  
`auto glucat::operator& (const index\_set< LO, HI > &lhs, const index\_set< LO, HI > &rhs) -> const index\_set< LO, HI >`

*Set intersection: and.*

- `template<const index_t LO, const index_t HI>`  
`auto glucat::operator| (const index_set< LO, HI > &lhs, const index_set< LO, HI > &rhs) -> const index_set< LO, HI >`

*Set union: or.*

- `template<const index_t LO, const index_t HI>`  
`auto glucat::compare (const index_set< LO, HI > &a, const index_set< LO, HI > &b) -> int`  
*"lexicographic compare" eg. {3,4,5} is less than {3,7,8}*
- `template<const index_t LO, const index_t HI>`  
`auto glucat::operator<< (std::ostream &os, const index_set< LO, HI > &ist) -> std::ostream &`

*Write out index set.*

- `template<const index_t LO, const index_t HI>`  
`auto glucat::operator>> (std::istream &s, index_set< LO, HI > &ist) -> std::istream &`

*Read in index set.*

- `static auto glucat::inverse_reversed_gray (unsigned long x) -> unsigned long`

*Inverse reversed Gray code.*

- `static auto glucat::inverse_gray (unsigned long x) -> unsigned long`

*Inverse Gray code.*

- `auto glucat::sign_of_square (index_t j) -> int`

*Square of generator {j}.*

- `template<const index_t LO, const index_t HI>`  
`auto glucat::min_neg (const index_set< LO, HI > &ist) -> index_t`

*Minimum negative index, or 0 if none.*

- `template<const index_t LO, const index_t HI>`  
`auto glucat::max_pos (const index_set< LO, HI > &ist) -> index_t`

*Maximum positive index, or 0 if none.*

## 7.28 index\_set\_imp.h

[Go to the documentation of this file.](#)

```
00001 #ifndef _GLUCAT_INDEX_SET_IMP_H
00002 #define _GLUCAT_INDEX_SET_IMP_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 index_set_imp.h : Implement a class for a set of non-zero integer indices
00006 -----
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2016 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 #include "glucat/index_set.h"
```

```

00035
00036 #include <string>
00037 #include <sstream>
00038
00039 namespace glucat
00040 {
00041 // References for algorithms:
00042 // [JA]: Joerg Arndt, "Algorithms for programmers", http://www.jjj.de/fxt/fxtbook.pdf
00043 // Chapter 1, Bit wizardry, http://www.jjj.de/bitwizardry/bitwizardrypage.html
00044 // [L]: Pertti Lounesto, "Clifford algebras and spinors", Cambridge UP, 1997.
00045
00046 template<const index_t LO, const index_t HI>
00047 inline
00048 auto
00049 index_set<LO,HI>::
00050 classname() -> const std::string
00051 { return "index_set"; }
00052
00053 template<const index_t LO, const index_t HI>
00054 index_set<LO,HI>::
00055 index_set(const index_t idx)
00056 { this->set(idx); }
00057
00058 template<const index_t LO, const index_t HI>
00059 index_set<LO,HI>::
00060 index_set(const bitset_t bst):
00061 bitset_t(bst)
00062 { }
00063
00064 template<const index_t LO, const index_t HI>
00065 index_set<LO,HI>::
00066 index_set(const set_value_t folded_val, const index_set_t frm, const bool prechecked)
00067 {
00068 if (!prechecked && folded_val >= (set_value_t(1) << frm.count()))
00069 throw error_t("index_set(val,frm): cannot create: value gives an index set outside of frame");
00070 const index_set_t folded_frame = frm.fold();
00071 const index_t min_index = folded_frame.min();
00072 const index_t skip = min_index > 0 ? 1 : 0;
00073 const index_set_t folded_set = index_set_t(bitset_t(folded_val) << (min_index - skip - LO));
00074 *this = folded_set.unfold(frm);
00075 }
00076
00077 template<const index_t LO, const index_t HI>
00078 index_set<LO,HI>::
00079 index_set(const index_pair_t& range, const bool prechecked)
00080 {
00081 if (!prechecked && (range.first < LO || range.second > HI))
00082 throw error_t("index_set(range): cannot create: range is too large");
00083 const index_t begin_bit = (range.first < 0)
00084 ? range.first-LO
00085 : range.first-LO-1;
00086 const index_t end_bit = (range.second < 0)
00087 ? range.second-LO+1
00088 : range.second-LO;
00089 unsigned long mask = ((end_bit == _GLUCAT_BITS_PER_ULONGLONG)
00090 ? -1UL
00091 : (1UL << end_bit)-1UL)
00092 & ~((1UL << begin_bit)-1UL);
00093 *this = bitset_t(mask);
00094 }
00095
00096 template<const index_t LO, const index_t HI>
00097 index_set<LO,HI>::
00098 index_set(const std::string& str)
00099 {
00100 std::istringstream ss(str);
00101 ss >> *this;
00102 if (!ss)
00103 throw error_t("index_set_t(str): could not parse string");
00104 // Peek to see if the end of the string has been reached.
00105 ss.peek();
00106 if (!ss.eof())
00107 throw error_t("index_set_t(str): could not parse entire string");
00108 }
00109
00110 template<const index_t LO, const index_t HI>
00111 inline
00112 auto
00113 index_set<LO,HI>::
00114 operator== (const index_set_t rhs) const -> bool
00115 {
00116 const auto* pthis = static_cast<const bitset_t*>(this);
00117 return *pthis == static_cast<bitset_t>(rhs);
00118 }
00119
00120 template<const index_t LO, const index_t HI>
00121 inline

```

```

00129 auto
00130 index_set<LO,HI>::
00131 operator!= (const index_set_t rhs) const -> bool
00132 {
00133 const auto* pthis = static_cast<const bitset_t*>(this);
00134 return *pthis != static_cast<bitset_t>(rhs);
00135 }
00136
00137 template<const index_t LO, const index_t HI>
00138 inline
00139 auto
00140 index_set<LO,HI>::
00141 operator~ () const -> index_set_t
00142 { return bitset_t::operator~(); }
00143
00144 template<const index_t LO, const index_t HI>
00145 inline
00146 auto
00147 index_set<LO,HI>::
00148 operator^= (const index_set_t rhs) -> index_set_t&
00149 {
00150 bitset_t* pthis = this;
00151 *pthis ^= static_cast<bitset_t>(rhs);
00152 return *this;
00153 }
00154
00155 template<const index_t LO, const index_t HI>
00156 inline
00157 auto
00158 operator^ (const index_set<LO,HI>& lhs,
00159 const index_set<LO,HI>& rhs) -> const
00160 index_set<LO,HI>
00161 {
00162 using index_set_t = index_set<LO, HI>;
00163 using bitset_t = typename index_set_t::bitset_t;
00164 return static_cast<bitset_t>(lhs) ^ static_cast<bitset_t>(rhs);
00165 }
00166
00167 template<const index_t LO, const index_t HI>
00168 inline
00169 auto
00170 index_set<LO,HI>::
00171 operator&= (const index_set_t rhs) -> index_set_t&
00172 {
00173 bitset_t* pthis = this;
00174 *pthis &= static_cast<bitset_t>(rhs);
00175 return *this;
00176 }
00177
00178 template<const index_t LO, const index_t HI>
00179 inline
00180 auto
00181 operator& (const index_set<LO,HI>& lhs,
00182 const index_set<LO,HI>& rhs) -> const
00183 index_set<LO,HI>
00184 {
00185 using index_set_t = index_set<LO, HI>;
00186 using bitset_t = typename index_set_t::bitset_t;
00187 return static_cast<bitset_t>(lhs) & static_cast<bitset_t>(rhs);
00188 }
00189
00190 template<const index_t LO, const index_t HI>
00191 inline
00192 auto
00193 index_set<LO,HI>::
00194 operator|= (const index_set_t rhs) -> index_set_t&
00195 {
00196 bitset_t* pthis = this;
00197 *pthis |= static_cast<bitset_t>(rhs);
00198 return *this;
00199 }
00200
00201 template<const index_t LO, const index_t HI>
00202 inline
00203 auto
00204 operator| (const index_set<LO,HI>& lhs,
00205 const index_set<LO,HI>& rhs) -> const
00206 index_set<LO,HI>
00207 {
00208 using index_set_t = index_set<LO, HI>;
00209 using bitset_t = typename index_set_t::bitset_t;
00210 return static_cast<bitset_t>(lhs) | static_cast<bitset_t>(rhs);
00211 }
00212
00213 template<const index_t LO, const index_t HI>
00214 inline
00215 auto

```



```

00224 index_set<LO,HI>::
00225 operator[] (const index_t idx) -> reference
00226 { return reference(*this, idx); }
00227
00229 template<const index_t LO, const index_t HI>
00230 inline
00231 auto
00232 index_set<LO,HI>::
00233 operator[] (const index_t idx) const -> bool
00234 { return this->test(idx); }
00235
00237 template<const index_t LO, const index_t HI>
00238 inline
00239 auto
00240 index_set<LO,HI>::
00241 test(const index_t idx) const -> bool
00242 {
00243 // Reference: [JA], 1.2.1
00244 return (idx < 0)
00245 ? bool(bitset_t::to_ulong() & (1UL << (idx - LO)))
00246 : (idx > 0)
00247 ? bool(bitset_t::to_ulong() & (1UL << (idx - LO - 1)))
00248 : false;
00249 }
00250
00252 template<const index_t LO, const index_t HI>
00253 inline
00254 auto
00255 index_set<LO,HI>::
00256 set() -> index_set_t&
00257 {
00258 bitset_t::set();
00259 return *this;
00260 }
00261
00263 template<const index_t LO, const index_t HI>
00264 inline
00265 auto
00266 index_set<LO,HI>::
00267 set(index_t idx) -> index_set_t&
00268 {
00269 if (idx > 0)
00270 bitset_t::set(idx-LO-1);
00271 else if (idx < 0)
00272 bitset_t::set(idx-LO);
00273 return *this;
00274 }
00275
00277 template<const index_t LO, const index_t HI>
00278 inline
00279 auto
00280 index_set<LO,HI>::
00281 set(const index_t idx, const int val) -> index_set_t&
00282 {
00283 if (idx > 0)
00284 bitset_t::set(idx-LO-1, val);
00285 else if (idx < 0)
00286 bitset_t::set(idx-LO, val);
00287 return *this;
00288 }
00289
00291 template<const index_t LO, const index_t HI>
00292 inline
00293 auto
00294 index_set<LO,HI>::
00295 reset() -> index_set_t&
00296 {
00297 bitset_t::reset();
00298 return *this;
00299 }
00300
00302 template<const index_t LO, const index_t HI>
00303 inline
00304 auto
00305 index_set<LO,HI>::
00306 reset(const index_t idx) -> index_set_t&
00307 {
00308 if (idx > 0)
00309 bitset_t::reset(idx-LO-1);
00310 else if (idx < 0)
00311 bitset_t::reset(idx-LO);
00312 return *this;
00313 }
00314
00316 template<const index_t LO, const index_t HI>
00317 inline
00318 auto

```

```

00319 index_set<LO,HI>::
00320 flip() -> index_set<LO,HI>&
00321 {
00322 bitset_t::flip();
00323 return *this;
00324 }
00325
00326 template<const index_t LO, const index_t HI>
00327 inline
00328 auto
00329 index_set<LO,HI>::
00330 flip(const index_t idx) -> index_set_t&
00331 {
00332 {
00333 if (idx > 0)
00334 bitset_t::flip(idx-LO-1);
00335 else if (idx < 0)
00336 bitset_t::flip(idx-LO);
00337 return *this;
00338 }
00339
00340 template<const index_t LO, const index_t HI>
00341 inline
00342 auto
00343 index_set<LO,HI>::
00344 count() const -> index_t
00345 {
00346 {
00347 unsigned long val = bitset_t::to_ulong();
00348 // Reference: [JA], 1.3
00349 if (val == 0)
00350 return 0;
00351 else
00352 {
00353 index_t result = 1;
00354 while (val &= val-1)
00355 ++result;
00356 return result;
00357 }
00358 }
00359
00360 template<const index_t LO, const index_t HI>
00361 inline
00362 auto
00363 index_set<LO,HI>::
00364 count_neg() const -> index_t
00365 {
00366 {
00367 static const index_set_t lo_mask = bitset_t((1UL << -LO) - 1UL);
00368 const index_set_t neg_part = *this & lo_mask;
00369 return neg_part.count();
00370 }
00371
00372 template<const index_t LO, const index_t HI>
00373 inline
00374 auto
00375 index_set<LO,HI>::
00376 count_pos() const -> index_t
00377 {
00378 {
00379 const auto* pthis = static_cast<const bitset_t*>(this);
00380 const index_set_t pos_part = *pthis >> -LO;
00381 return pos_part.count();
00382 }
00383
00384 #if (_GLUCAT_BITS_PER_ULONG == 64)
00385 template<const index_t LO, const index_t HI>
00386 inline
00387 auto
00388 index_set<LO,HI>::
00389 min() const -> index_t
00390 {
00391 {
00392 // Reference: [JA], 1.3
00393 unsigned long val = bitset_t::to_ulong();
00394 if (val == 0)
00395 return 0;
00396 else
00397 {
00398 val -= val & (val-1); // isolate lowest bit
00399
00400 index_t idx = 0;
00401 const index_t nbits = HI - LO;
00402
00403 if (nbits > 8)
00404 {
00405 if (val & 0xffffffff00000000ul)
00406 idx += 32;
00407 if (val & 0xffff0000ffff0000ul)
00408 idx += 16;
00409 if (val & 0xff00ff00ff00ff00ul)
00410 idx += 8;

```

```

00411 }
00412 if (val & 0xf0f0f0f0f0f0f0ul)
00413 idx += 4;
00414 if (val & 0xccccccccccccccul)
00415 idx += 2;
00416 if (val & 0aaaaaaaaaaaaaaul)
00417 idx += 1;
00418
00419 return idx + ((idx < -LO) ? LO : LO+1);
00420 }
00421 }
00422 #elif (_GLUCAT_BITS_PER_ULONG == 32)
00423 template<const index_t LO, const index_t HI>
00424 inline
00425 index_t
00426 index_set<LO,HI>::
00427 min() const
00428 {
00429 // Reference: [JA], 1.3
00430 unsigned long val = bitset_t::to_ulong();
00431 if (val == 0)
00432 return 0;
00433 else
00434 {
00435 val -= val & (val-1); // isolate lowest bit
00436
00437 index_t idx = 0;
00438 const index_t nbits = HI - LO;
00439 if (nbits > 8)
00440 {
00441 if (val & 0xffff0000ul)
00442 idx += 16;
00443 if (val & 0xff00ff00ul)
00444 idx += 8;
00445 }
00446 if (val & 0xf0f0f0f0ul)
00447 idx += 4;
00448 if (val & 0xccccccccul)
00449 idx += 2;
00450 if (val & 0aaaaaaaaul)
00451 idx += 1;
00452
00453 return idx + ((idx < -LO) ? LO : LO+1);
00454 }
00455 }
00456 }
00457 #else
00458 template<const index_t LO, const index_t HI>
00459 auto
00460 index_set<LO,HI>::
00461 min() const -> index_t
00462 {
00463 for (auto
00464 idx = LO;
00465 idx != 0;
00466 ++idx)
00467 if (this->test(idx))
00468 return idx;
00469 for (auto
00470 idx = index_t(1);
00471 idx <= HI;
00472 ++idx)
00473 if (this->test(idx))
00474 return idx;
00475 return 0;
00476 }
00477 }
00478 #endif
00479
00480 #if (_GLUCAT_BITS_PER_ULONG == 64)
00481 template<const index_t LO, const index_t HI>
00482 inline
00483 auto
00484 index_set<LO,HI>::
00485 max() const -> index_t
00486 {
00487 // Reference: [JA], 1.6
00488 auto val = bitset_t::to_ulong();
00489 if (val == 0)
00490 return 0;
00491 else
00492 {
00493 auto idx = index_t(0);
00494 const auto nbits = HI - LO;
00495 if (nbits > 8)
00496 {
00497 if (val & 0xffffffff00000000ul)
00498 { val >>= 32; idx += 32; }
00499 if (val & 0x00000000ffff0000ul)

```

```

00501 { val >>= 16; idx += 16; }
00502 if (val & 0x000000000000ff00ul)
00503 { val >>= 8; idx += 8; }
00504 }
00505 if (val & 0x0000000000000f0ul)
00506 { val >>= 4; idx += 4; }
00507 if (val & 0x00000000000000cul)
00508 { val >>= 2; idx += 2; }
00509 if (val & 0x000000000000002ul)
00510 { idx += 1; }
00511 return idx + ((idx < -LO) ? LO : LO+1);
00512 }
00513 }
00514 #elif (_GLUCAT_BITS_PER_ULONG == 32)
00515 template<const index_t LO, const index_t HI>
00516 inline
00517 auto
00518 index_set<LO,HI>::
00519 max() const -> index_t
00520 {
00521 // Reference: [JA], 1.6
00522 auto val = bitset_t::to_ulong();
00523 if (val == 0)
00524 return 0;
00525 else
00526 {
00527 auto idx = index_t(0);
00528 const auto nbits = HI - LO;
00529 if (nbits > 8)
00530 {
00531 if (val & 0xffff0000ul)
00532 { val >>= 16; idx += 16; }
00533 if (val & 0x0000ff00ul)
00534 { val >>= 8; idx += 8; }
00535 }
00536 if (val & 0x000000f0ul)
00537 { val >>= 4; idx += 4; }
00538 if (val & 0x0000000cul)
00539 { val >>= 2; idx += 2; }
00540 if (val & 0x00000002ul)
00541 { idx += 1; }
00542 return idx + ((idx < -LO) ? LO : LO+1);
00543 }
00544 }
00545 }
00546 #else
00547 template<const index_t LO, const index_t HI>
00548 auto
00549 index_set<LO,HI>::
00550 max() const -> index_t
00551 {
00552 for (auto
00553 idx = HI;
00554 idx != 0;
00555 --idx)
00556 if (this->test(idx))
00557 return idx;
00558 for (auto
00559 idx = index_t(-1);
00560 idx >= LO;
00561 --idx)
00562 if (this->test(idx))
00563 return idx;
00564 return 0;
00565 }
00566 #endif
00567 // eg. {3,4,5} is less than {3,7,8}
00568 template<const index_t LO, const index_t HI>
00569 inline
00570 auto
00571 compare(const index_set<LO,HI>& a, const index_set<LO,HI>& b) -> int
00572 {
00573 return (a == b)
00574 ? 0
00575 : a.lex_less_than(b)
00576 ? -1
00577 : 1;
00578 }
00579 // eg. {3,4,5} is less than {3,7,8}
00580 template<const index_t LO, const index_t HI>
00581 inline
00582 auto
00583 index_set<LO,HI>::
00584 lex_less_than(const index_set_t rhs) const -> bool
00585 { return bitset_t::to_ulong() < rhs.bitset_t::to_ulong(); }
00586
00587
00588
00589
00590
00591

```

```

00593 // Order by count, then order lexicographically within the equivalence class of count.
00594 template<const index_t LO, const index_t HI>
00595 inline
00596 auto
00597 index_set<LO,HI>::
00598 operator< (const index_set_t rhs) const -> bool
00599 {
00600 const auto this_grade = this->count();
00601 const auto rhs_grade = rhs.count();
00602 return (this_grade < rhs_grade)
00603 ? true
00604 : (this_grade > rhs_grade)
00605 ? false
00606 : this->lex_less_than(rhs);
00607 }
00608
00609 template<const index_t LO, const index_t HI>
00610 auto
00611 operator<< (std::ostream& os, const index_set<LO,HI>& ist) -> std::ostream&
00612 {
00613 {
00614 index_t i;
00615 os << '{';
00616 for (i = LO;
00617 (i <= HI) && !(ist[i]);
00618 ++i)
00619 { }
00620 if (i <= HI)
00621 os << i;
00622 for (++i;
00623 i <= HI;
00624 ++i)
00625 if (ist[i])
00626 os << ',' << i;
00627 os << '}';
00628 return os;
00629 }
00630
00631 template<const index_t LO, const index_t HI>
00632 auto
00633 operator>> (std::istream& s, index_set<LO,HI>& ist) -> std::istream&
00634 {
00635 // Parsing variables.
00636 auto i = index_t(0);
00637 using index_set_t = index_set<LO,HI>;
00638 auto local_ist = index_set_t();
00639 // Parsing control variables.
00640 auto parse_index_list = true;
00641 auto expect_closing_brace = false;
00642 auto expect_index = false;
00643 // Parse an optional opening brace.
00644 auto c = s.peek();
00645 // If there is a failure or end of file, this ends parsing.
00646 if (!s.good())
00647 parse_index_list = false;
00648 else
00649 { // Check for an opening brace.
00650 expect_closing_brace = (c == int('{'));
00651 if (expect_closing_brace)
00652 { // Consume the opening brace.
00653 s.get();
00654 // The next character may be a closing brace,
00655 // indicating the empty index set.
00656 c = s.peek();
00657 if (s.good() && (c == int('}')))
00658 { // A closing brace has been parsed and is no longer expected.
00659 expect_closing_brace = false;
00660 // Consume the closing brace.
00661 s.get();
00662 // This ends parsing.
00663 parse_index_list = false;
00664 }
00665 }
00666 }
00667 if (s.good() && parse_index_list)
00668 { // Parse an optional index list.
00669 // The index list starts with a first index.
00670 for (s >> i;
00671 !s.fail();
00672 s >> i)
00673 { // An index has been parsed. Check to see if it is in range.
00674 if ((i < LO) || (i > HI))
00675 { // An index out of range is a failure.
00676 s.clear(std::istream::failbit);
00677 break;
00678 }
00679 // Add the index to the index set local_ist.
00680 local_ist.set(i);
00681 }
00682 }

```

```

00682 // Immediately after parsing an index, an index is no longer expected.
00683 expect_index = false;
00684 // Reading the index may have resulted in an end of file condition.
00685 // If so, this ends the index list.
00686 if (s.eof())
00687 break;
00688 // The index list continues with a comma, and
00689 // may be ended by a closing brace, if it was begun with an opening brace.
00690 // Parse a possible comma or closing brace.
00691 c = s.peek();
00692 if (!s.good())
00693 break;
00694 // First, test for a closing brace, if expected.
00695 if (expect_closing_brace && (c == int('}')))
00696 { // Consume the closing brace.
00697 s.get();
00698 // Immediately after parsing the closing brace, it is no longer expected.
00699 expect_closing_brace = false;
00700 // A closing brace ends the index list.
00701 break;
00702 }
00703 // Now test for a comma.
00704 if (c == int(','))
00705 { // Consume the comma.
00706 s.get();
00707 // A index is expected after the comma.
00708 expect_index = true;
00709 }
00710 else
00711 { // Any other character here is a failure.
00712 s.clear(std::istream::failbit);
00713 break;
00714 }
00715 }
00716 }
00717 // If an index or a closing brace is still expected, this is a failure.
00718 if (expect_index || expect_closing_brace)
00719 s.clear(std::istream::failbit);
00720 // End of file is not a failure.
00721 if (s)
00722 { // The index set has been successfully parsed.
00723 ist = local_ist;
00724 }
00725 return s;
00726 }
00727
00728 template<const index_t LO, const index_t HI>
00729 inline
00730 auto
00731 index_set<LO,HI>::
00732 is_contiguous () const -> bool
00733 {
00734 {
00735 const auto min_index = this->min();
00736 const auto max_index = this->max();
00737 return (min_index < 0 && max_index > 0)
00738 ? max_index - min_index == this->count()
00739 : (min_index == 1 || max_index == -1) &&
00740 (max_index - min_index == this->count() - 1);
00741 }
00742 }
00743
00744 template<const index_t LO, const index_t HI>
00745 inline
00746 auto
00747 index_set<LO,HI>::
00748 fold() const -> const
00749 index_set<LO,HI>
00750 { return this->fold(*this, true); }
00751
00752 template<const index_t LO, const index_t HI>
00753 auto
00754 index_set<LO,HI>::
00755 fold(const index_set_t frm, const bool prechecked) const -> const
00756 index_set<LO,HI>
00757 {
00758 {
00759 if (!prechecked && ((*this | frm) != frm))
00760 throw error_t("fold(frm): cannot fold from outside of frame");
00761 const auto frm_min = frm.min();
00762 const auto frm_max = frm.max();
00763 auto result = index_set_t();
00764 auto fold_idx = index_t(-1);
00765 for (auto
00766 unfold_idx = fold_idx;
00767 unfold_idx >= frm_min;
00768 --unfold_idx)
00769 if (frm.test(unfold_idx))
00770 // result.set(fold_idx--, this->test(unfold_idx));
00771 {

```

```

00772 if (this->test(unfold_idx))
00773 result.set(fold_idx);
00774 --fold_idx;
00775 }
00776 fold_idx = index_t(1);
00777 for (auto
00778 unfold_idx = fold_idx;
00779 unfold_idx <= frm_max;
00780 ++unfold_idx)
00781 if (frm.test(unfold_idx))
00782 // result.set(fold_idx++, this->test(unfold_idx));
00783 {
00784 if (this->test(unfold_idx))
00785 result.set(fold_idx);
00786 ++fold_idx;
00787 }
00788 return result;
00789 }
00790
00791 template<const index_t LO, const index_t HI>
00792 auto
00793 index_set<LO,HI>::
00794 unfold(const index_set_t frm, const bool prechecked) const -> const index_set_t
00795 {
00796 {
00797 const char* msg =
00798 "unfold(frm): cannot unfold into a smaller frame";
00799 const auto frm_min = frm.min();
00800 const auto frm_max = frm.max();
00801 auto result = index_set_t();
00802 auto fold_idx = index_t(-1);
00803 for (auto
00804 unfold_idx = fold_idx;
00805 unfold_idx >= frm_min;
00806 --unfold_idx)
00807 if (frm.test(unfold_idx))
00808 if (this->test(fold_idx--))
00809 result.set(unfold_idx);
00810 if (!prechecked && ((fold_idx+1) > this->min()))
00811 throw error_t(msg);
00812 fold_idx = index_t(1);
00813 for (auto
00814 unfold_idx = fold_idx;
00815 unfold_idx <= frm_max;
00816 ++unfold_idx)
00817 if (frm.test(unfold_idx))
00818 if (this->test(fold_idx++))
00819 result.set(unfold_idx);
00820 if (!prechecked && ((fold_idx-1) < this->max()))
00821 throw error_t(msg);
00822 return result;
00823 }
00824
00825 template<const index_t LO, const index_t HI>
00826 inline
00827 auto
00828 index_set<LO,HI>::
00829 value_of_fold(const index_set_t frm) const -> set_value_t
00830 {
00831 {
00832 const auto min_index = frm.fold().min();
00833 if (min_index == 0)
00834 return 0;
00835 else
00836 {
00837 const auto folded_set = this->fold(frm);
00838 const auto skip = min_index > 0 ? index_t(1) : index_t(0);
00839 return folded_set.bitset_t::to_ulong() >> (min_index-LO-skip);
00840 }
00841 }
00842
00843 inline
00844 static
00845 auto inverse_reversed_gray(unsigned long x) -> unsigned long
00846 {
00847 {
00848 // Reference: [JA]
00849 #if (_GLUCAT_BITS_PER_ULONG >= 64)
00850 x ^= x << 32; // for 64-bit words
00851 #endif
00852 x ^= x << 16; // reversed_gray ** 16
00853 x ^= x << 8; // reversed_gray ** 8
00854 x ^= x << 4; // reversed_gray ** 4
00855 x ^= x << 2; // reversed_gray ** 2
00856 x ^= x << 1; // reversed_gray ** 1
00857 return x;
00858 }
00859
00860 inline
00861 static

```

```

00863 auto inverse_gray(unsigned long x) -> unsigned long
00864 {
00865 // Reference: [JA]
00866 #if (_GLUCAT_BITS_PER_ULONG >= 64)
00867 x ^= x » 32; // for 64-bit words
00868 #endif
00869 x ^= x » 16; // gray ** 16
00870 x ^= x » 8; // gray ** 8
00871 x ^= x » 4; // gray ** 4
00872 x ^= x » 2; // gray ** 2
00873 x ^= x » 1; // gray ** 1
00874 return x;
00875 }
00876
00877 template<const index_t LO, const index_t HI>
00878 auto
00880 index_set<LO,HI>::
00881 sign_of_mult(const index_set_t rhs) const -> int
00882 {
00883 // Implemented using Walsh functions and Gray codes.
00884 // Reference: [L] Chapter 21, 21.3
00885 // Reference: [JA]
00886 const auto uthis = this->bitset_t::to_ulong();
00887 const auto urhs = rhs.bitset_t::to_ulong();
00888 const auto nbits = HI - LO;
00889 auto negative = 0UL;
00890 if (nbits > 8)
00891 {
00892 // Set h to be the inverse reversed Gray code of rhs.
00893 // This sets each bit of h to be the cumulative ^ of
00894 // the same and lower bits of rhs.
00895 const auto h = inverse_reversed_gray(urhs);
00896 // Set k to be the inverse Gray code of *this & h.
00897 // This sets the low bit of k to be parity(*this & h).
00898 const auto k = inverse_gray(uthis & h);
00899 // Set q to be the inverse Gray code of the positive part of *this & rhs.
00900 const auto q = inverse_gray((uthis & urhs) » -LO);
00901 negative = k ^ q;
00902 }
00903 else
00904 {
00905 auto h = 0UL;
00906 for (auto
00907 j = index_t(0);
00908 j < -LO;
00909 ++j)
00910 {
00911 h ^= urhs » j;
00912 negative ^= h & (uthis » j);
00913 }
00914 for (auto
00915 j = index_t(-LO);
00916 j < nbits;
00917 ++j)
00918 {
00919 negative ^= h & (uthis » j);
00920 h ^= urhs » j;
00921 }
00922 }
00923 return 1 - int((negative & 1) « 1);
00924 }
00925
00927 template<const index_t LO, const index_t HI>
00928 inline
00929 auto
00930 index_set<LO,HI>::
00931 sign_of_square() const -> int
00932 {
00933 auto result = 1 - int((this->count_neg() % 2) « 1);
00934 switch (this->count() % 4)
00935 {
00936 case 2:
00937 case 3:
00938 result *= -1;
00939 break;
00940 default:
00941 break;
00942 }
00943 return result;
00944 }
00945
00947 template<const index_t LO, const index_t HI>
00948 inline
00949 auto
00950 index_set<LO,HI>::
00951 hash_fn() const -> size_t
00952 {

```



```

00953 static const auto lo_mask = (1UL « -LO) - 1UL;
00954 const auto uthis = bitset_t::to_ulong();
00955 const auto neg_part = uthis & lo_mask;
00956 const auto pos_part = uthis » -LO;
00957 return size_t(neg_part ^ pos_part);
00958 }
00959
00961 inline
00962 auto
00963 sign_of_square(index_t j) -> int
00964 { return (j < 0) ? -1 : 1; }
00965
00967 template<const index_t LO, const index_t HI>
00968 inline
00969 auto
00970 min_neg(const index_set<LO,HI>& ist) -> index_t
00971 { return std::min(ist.min(), 0); }
00972
00974 template<const index_t LO, const index_t HI>
00975 inline
00976 auto
00977 max_pos(const index_set<LO,HI>& ist) -> index_t
00978 { return std::max(ist.max(), 0); }
00979
00980 // index_set reference
00981
00983 template<const index_t LO, const index_t HI>
00984 inline
00985 index_set<LO,HI>::reference::
00986 reference(index_set_t& ist, index_t idx) :
00987 m_pst(&ist),
00988 m_idx(idx)
00989 { }
00990
00992 template<const index_t LO, const index_t HI>
00993 inline
00994 auto
00995 index_set<LO,HI>::reference::
00996 operator== (const reference& c_j) const -> bool
00997 { return m_pst == c_j.m_pst && m_idx == c_j.m_idx; }
00998
01000 template<const index_t LO, const index_t HI>
01001 inline
01002 auto
01003 index_set<LO,HI>::reference::
01004 operator= (bool x) -> reference&
01005 {
01006 if (x)
01007 m_pst->set(m_idx);
01008 else
01009 m_pst->reset(m_idx);
01010 return *this;
01011 }
01012
01014 template<const index_t LO, const index_t HI>
01015 inline
01016 auto
01017 index_set<LO,HI>::reference::
01018 operator= (const reference& c_j) -> reference&
01019 {
01020 if (&c_j != this && c_j != *this)
01021 {
01022 if ((c_j.m_pst)[c_j.m_idx])
01023 m_pst->set(m_idx);
01024 else
01025 m_pst->reset(m_idx);
01026 }
01027 return *this;
01028 }
01029
01031 template<const index_t LO, const index_t HI>
01032 inline
01033 auto
01034 index_set<LO,HI>::reference::
01035 operator~ () const -> bool
01036 { return !(m_pst->test(m_idx)); }
01037
01039 template<const index_t LO, const index_t HI>
01040 inline
01041 index_set<LO,HI>::reference::
01042 operator bool () const
01043 { return m_pst->test(m_idx); }
01044
01046 template<const index_t LO, const index_t HI>
01047 inline
01048 auto
01049 index_set<LO,HI>::reference::

```

```

01050 flip() -> reference&
01051 {
01052 m_pst->flip(m_idx);
01053 return *this;
01054 }
01055 }
01056 #endif // _GLUCAT_INDEX_SET_IMP_H

```

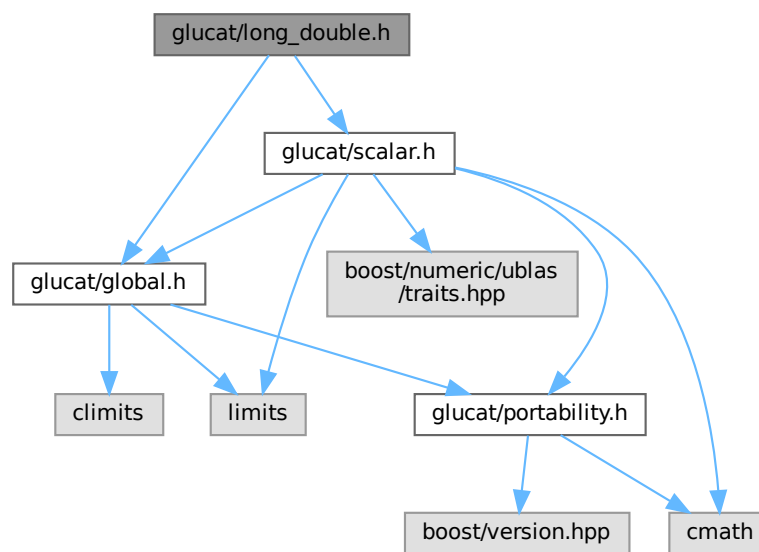
## 7.29 glucat/long\_double.h File Reference

```

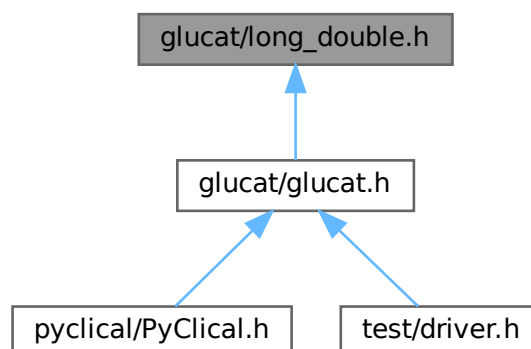
#include "glucat/global.h"
#include "glucat/scalar.h"

```

Include dependency graph for long\_double.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace `glucat`

## Variables

- static const long double `glucat::l_pi` = 3.1415926535897932384626433832795029L
- static const long double `glucat::l_ln2` = 0.6931471805599453094172321214581766L

## 7.30 long\_double.h

[Go to the documentation of this file.](#)

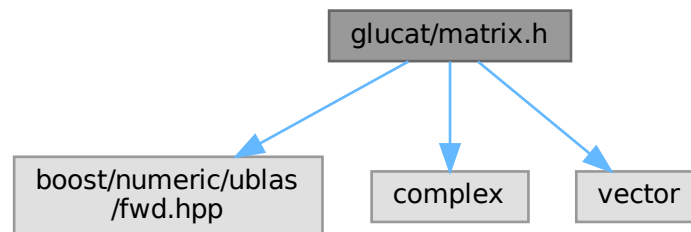
```

00001 #ifndef _GLUCAT_LONG_DOUBLE_H
00002 #define _GLUCAT_LONG_DOUBLE_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 long_double.h : Define std functions for long double
00006 -----
00007 begin : 2001-12-18
00008 copyright : (C) 2001-2016 by Paul C. Leopardi
00009 *****/
00010 This library is free software: you can redistribute it and/or modify
00011 it under the terms of the GNU Lesser General Public License as published
00012 by the Free Software Foundation, either version 3 of the License, or
00013 (at your option) any later version.
00014
00015 This library is distributed in the hope that it will be useful,
00016 but WITHOUT ANY WARRANTY; without even the implied warranty of
00017 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00018 GNU Lesser General Public License for more details.
00019
00020 You should have received a copy of the GNU Lesser General Public License
00021 along with this library. If not, see <http://www.gnu.org/licenses/>.
00022
00023 *****/
00024 This library is based on a prototype written by Arvind Raja and was
00025 licensed under the LGPL with permission of the author. See Arvind Raja,
00026 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00027 in Ablamowicz, Lounesto and Parra (eds.)
00028 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
00029 *****/
00030 See also Arvind Raja's original header comments and references in glucat.h
00031 *****/
00032
00033 #include "glucat/global.h"
00034 #include "glucat/scalar.h"
00035
00036 namespace glucat
00037 {
00038 #if defined(__USE_GNU)
00039 static const long double l_pi = M_PI;
00040 static const long double l_ln2 = M_LN2;
00041 #else
00042 static const long double l_pi = 3.1415926535897932384626433832795029L;
00043 static const long double l_ln2 = 0.6931471805599453094172321214581766L;
00044 #endif
00045
00046 template<>
00047 inline
00048 auto
00049 numeric_traits<long double>::
00050 pi() -> long double
00051 { return l_pi; }
00052
00053 template<>
00054 inline
00055 auto
00056 numeric_traits<long double>::
00057 ln_2() -> long double
00058 { return l_ln2; }
00059 }
00060 #endif // _GLUCAT_LONG_DOUBLE_H

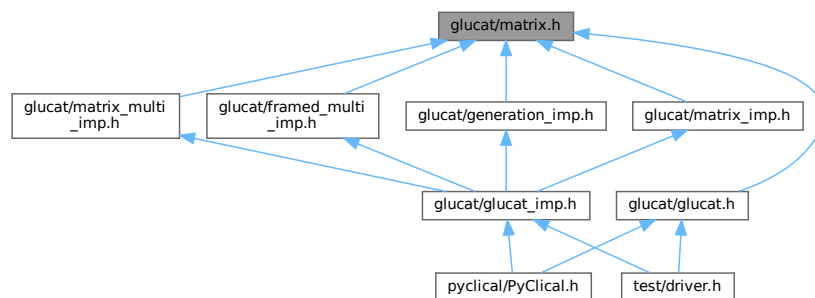
```

## 7.31 glucat/matrix.h File Reference

```
#include <boost/numeric/ublas/fwd.hpp>
#include <complex>
#include <vector>
Include dependency graph for matrix.h:
```



This graph shows which files directly or indirectly include this file:



### Classes

- struct [glucat::matrix::eig\\_genus< Matrix\\_T >](#)  
*Structure containing classification of eigenvalues.*

### Namespaces

- namespace [glucat](#)
- namespace [glucat::matrix](#)

### Typedefs

- using [glucat::matrix::eig\\_case\\_t](#)  
*Classification of eigenvalues of a matrix.*

## Functions

- template<typename LHS\_T , typename RHS\_T >  
 auto [glucat::matrix::kron](#) (const LHS\_T &lhs, const RHS\_T &rhs) -> const RHS\_T  
*Kronecker tensor product of matrices - as per Matlab kron.*
- template<typename LHS\_T , typename RHS\_T >  
 auto [glucat::matrix::mono\\_kron](#) (const LHS\_T &lhs, const RHS\_T &rhs) -> const RHS\_T  
*Sparse Kronecker tensor product of monomial matrices.*
- template<typename LHS\_T , typename RHS\_T >  
 auto [glucat::matrix::nork](#) (const LHS\_T &lhs, const RHS\_T &rhs, const bool mono=true) -> const RHS\_T  
*Left inverse of Kronecker product.*
- template<typename LHS\_T , typename RHS\_T >  
 auto [glucat::matrix::signed\\_perm\\_nork](#) (const LHS\_T &lhs, const RHS\_T &rhs) -> const RHS\_T  
*Left inverse of Kronecker product where lhs is a signed permutation matrix.*
- template<typename Matrix\_T >  
 auto [glucat::matrix::nnz](#) (const Matrix\_T &m) -> typename Matrix\_T::size\_type  
*Number of non-zeros.*
- template<typename Matrix\_T >  
 auto [glucat::matrix::isinf](#) (const Matrix\_T &m) -> bool  
*Infinite.*
- template<typename Matrix\_T >  
 auto [glucat::matrix::isnan](#) (const Matrix\_T &m) -> bool  
*Not a Number.*
- template<typename Matrix\_T >  
 auto [glucat::matrix::unit](#) (const typename Matrix\_T::size\_type n) -> const Matrix\_T  
*Unit matrix - as per Matlab eye.*
- template<typename LHS\_T , typename RHS\_T >  
 auto [glucat::matrix::mono\\_prod](#) (const ublas::matrix\_expression< LHS\_T > &lhs, const ublas::matrix\_↵  
 expression< RHS\_T > &rhs) -> const typename RHS\_T::expression\_type  
*Product of monomial matrices.*
- template<typename LHS\_T , typename RHS\_T >  
 auto [glucat::matrix::sparse\\_prod](#) (const ublas::matrix\_expression< LHS\_T > &lhs, const ublas::matrix\_↵  
 expression< RHS\_T > &rhs) -> const typename RHS\_T::expression\_type  
*Product of sparse matrices.*
- template<typename LHS\_T , typename RHS\_T >  
 auto [glucat::matrix::prod](#) (const ublas::matrix\_expression< LHS\_T > &lhs, const ublas::matrix\_expression<  
 RHS\_T > &rhs) -> const typename RHS\_T::expression\_type  
*Product of matrices.*
- template<typename Scalar\_T , typename LHS\_T , typename RHS\_T >  
 auto [glucat::matrix::inner](#) (const LHS\_T &lhs, const RHS\_T &rhs) -> Scalar\_T  
*Inner product:  $\sum(x(i,j)*y(i,j))/x.nrows()$*
- template<typename Matrix\_T >  
 auto [glucat::matrix::norm\\_frob2](#) (const Matrix\_T &val) -> typename Matrix\_T::value\_type  
*Square of Frobenius norm.*
- template<typename Matrix\_T >  
 auto [glucat::matrix::trace](#) (const Matrix\_T &val) -> typename Matrix\_T::value\_type  
*Matrix trace.*
- template<typename Matrix\_T >  
 auto [glucat::matrix::eigenvalues](#) (const Matrix\_T &val) -> std::vector< std::complex< double > >  
*Eigenvalues of a matrix.*
- template<typename Matrix\_T >  
 auto [glucat::matrix::classify\\_eigenvalues](#) (const Matrix\_T &val) -> [eig\\_genus](#)< Matrix\_T >  
*Classify the eigenvalues of a matrix.*

## 7.32 matrix.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_MATRIX_H
00002 #define _GLUCAT_MATRIX_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 matrix.h : Declare common matrix functions
00006
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2012 by Paul C. Leopardi
00009 : uBLAS interface contributed by Joerg Walter
00010 *****/
00011
00012 This library is free software: you can redistribute it and/or modify
00013 it under the terms of the GNU Lesser General Public License as published
00014 by the Free Software Foundation, either version 3 of the License, or
00015 (at your option) any later version.
00016
00017 This library is distributed in the hope that it will be useful,
00018 but WITHOUT ANY WARRANTY; without even the implied warranty of
00019 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00020 GNU Lesser General Public License for more details.
00021
00022 You should have received a copy of the GNU Lesser General Public License
00023 along with this library. If not, see <http://www.gnu.org/licenses/>.
00024
00025 *****/
00026 This library is based on a prototype written by Arvind Raja and was
00027 licensed under the LGPL with permission of the author. See Arvind Raja,
00028 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00029 in Ablamowicz, Lounesto and Parra (eds.)
00030 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00031 *****/
00032 See also Arvind Raja's original header comments in glucat.h
00033 *****/
00034
00035 #include <boost/numeric/ublas/fwd.hpp>
00036
00037 #include <complex>
00038 #include <vector>
00039
00040 namespace glucat
00041 {
00042 namespace ublas = boost::numeric::ublas;
00043
00044 namespace matrix
00045 {
00046 template< typename LHS_T, typename RHS_T >
00047 auto
00048 kron(const LHS_T& lhs, const RHS_T& rhs) -> const
00049 RHS_T;
00050
00051 template< typename LHS_T, typename RHS_T >
00052 auto
00053 mono_kron(const LHS_T& lhs, const RHS_T& rhs) -> const
00054 RHS_T;
00055
00056 template< typename LHS_T, typename RHS_T >
00057 auto
00058 nork(const LHS_T& lhs, const RHS_T& rhs, const bool mono = true) -> const
00059 RHS_T;
00060
00061 template< typename LHS_T, typename RHS_T >
00062 auto
00063 signed_perm_nork(const LHS_T& lhs, const RHS_T& rhs) -> const
00064 RHS_T;
00065
00066 template< typename Matrix_T >
00067 auto
00068 nnz(const Matrix_T& m) -> typename Matrix_T::size_type;
00069
00070 template< typename Matrix_T >
00071 auto
00072 isinf(const Matrix_T& m) -> bool;
00073
00074 template< typename Matrix_T >
00075 auto
00076 isnan(const Matrix_T& m) -> bool;
00077
00078 template< typename Matrix_T >
00079 auto
00080 unit(const typename Matrix_T::size_type n) -> const
00081 Matrix_T;
00082
00083 }
00084
00085 }

```

```

00092 template< typename LHS_T, typename RHS_T >
00093 auto
00094 mono_prod(const ublas::matrix_expression<LHS_T>& lhs,
00095 const ublas::matrix_expression<RHS_T>& rhs) -> const
00096 typename RHS_T::expression_type;
00097
00099 template< typename LHS_T, typename RHS_T >
00100 auto
00101 sparse_prod(const ublas::matrix_expression<LHS_T>& lhs,
00102 const ublas::matrix_expression<RHS_T>& rhs) -> const
00103 typename RHS_T::expression_type;
00104
00106 template< typename LHS_T, typename RHS_T >
00107 auto
00108 prod(const ublas::matrix_expression<LHS_T>& lhs,
00109 const ublas::matrix_expression<RHS_T>& rhs) -> const
00110 typename RHS_T::expression_type;
00111
00113 template< typename Scalar_T, typename LHS_T, typename RHS_T >
00114 auto
00115 inner(const LHS_T& lhs, const RHS_T& rhs) -> Scalar_T;
00116
00118 template< typename Matrix_T >
00119 auto
00120 norm_frob2(const Matrix_T& val) -> typename Matrix_T::value_type;
00121
00123 template< typename Matrix_T >
00124 auto
00125 trace(const Matrix_T& val) -> typename Matrix_T::value_type;
00126
00128 template< typename Matrix_T >
00129 auto
00130 eigenvalues(const Matrix_T& val) -> std::vector< std::complex<double> >;
00131
00133 using eig_case_t = enum {
00134 safe_eigs,
00135 neg_real_eigs,
00136 both_eigs};
00137
00139 template< typename Matrix_T >
00140 struct eig_genus
00141 {
00142 using Scalar_T = typename Matrix_T::value_type;
00144 bool m_is_singular = false;
00146 eig_case_t m_eig_case = safe_eigs;
00148 Scalar_T m_safe_arg = Scalar_T(0);
00149 };
00150
00152 template< typename Matrix_T >
00153 auto
00154 classify_eigenvalues(const Matrix_T& val) -> eig_genus<Matrix_T>;
00155 }
00156 }
00157
00158 #endif // _GLUCAT_MATRIX_H

```

## 7.33 glucat/matrix\_imp.h File Reference

```

#include "glucat/errors.h"
#include "glucat/scalar.h"
#include "glucat/matrix.h"
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/vector_proxy.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/matrix_expression.hpp>
#include <boost/numeric/ublas/matrix_proxy.hpp>
#include <boost/numeric/ublas/matrix_sparse.hpp>
#include <boost/numeric/ublas/operation.hpp>
#include <boost/numeric/ublas/operation_sparse.hpp>
#include <boost/numeric/bindings/lapack/driver/gees.hpp>
#include <boost/numeric/bindings/ublas.hpp>
#include <set>

```





- `template<typename Matrix_T >`  
`auto glucat::matrix::nnz (const Matrix_T &m) -> typename Matrix_T::size_type`  
*Number of non-zeros.*
- `template<typename Matrix_T >`  
`auto glucat::matrix::isinf (const Matrix_T &m) -> bool`  
*Infinite.*
- `template<typename Matrix_T >`  
`auto glucat::matrix::isnan (const Matrix_T &m) -> bool`  
*Not a Number.*
- `template<typename Matrix_T >`  
`auto glucat::matrix::unit (const typename Matrix_T::size_type n) -> const Matrix_T`  
*Unit matrix - as per Matlab eye.*
- `template<typename LHS_T , typename RHS_T >`  
`auto glucat::matrix::mono_prod (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_↵`  
`expression< RHS_T > &rhs) -> const typename RHS_T::expression_type`  
*Product of monomial matrices.*
- `template<typename LHS_T , typename RHS_T >`  
`auto glucat::matrix::sparse_prod (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_↵`  
`expression< RHS_T > &rhs) -> const typename RHS_T::expression_type`  
*Product of sparse matrices.*
- `template<typename LHS_T , typename RHS_T >`  
`auto glucat::matrix::prod (const ublas::matrix_expression< LHS_T > &lhs, const ublas::matrix_expression<`  
`RHS_T > &rhs) -> const typename RHS_T::expression_type`  
*Product of matrices.*
- `template<typename Scalar_T , typename LHS_T , typename RHS_T >`  
`auto glucat::matrix::inner (const LHS_T &lhs, const RHS_T &rhs) -> Scalar_T`  
*Inner product:  $\sum(x(i,j)*y(i,j))/x.nrows()$*
- `template<typename Matrix_T >`  
`auto glucat::matrix::norm_frob2 (const Matrix_T &val) -> typename Matrix_T::value_type`  
*Square of Frobenius norm.*
- `template<typename Matrix_T >`  
`auto glucat::matrix::trace (const Matrix_T &val) -> typename Matrix_T::value_type`  
*Matrix trace.*
- `template<typename Matrix_T >`  
`static auto glucat::matrix::to_lapack (const Matrix_T &val) -> ublas::matrix< double, ublas::column_major >`  
*Convert matrix to LAPACK format.*
- `template<typename Matrix_T >`  
`auto glucat::matrix::eigenvalues (const Matrix_T &val) -> std::vector< std::complex< double > >`  
*Eigenvalues of a matrix.*
- `template<typename Matrix_T >`  
`auto glucat::matrix::classify_eigenvalues (const Matrix_T &val) -> eig_genus< Matrix_T >`  
*Classify the eigenvalues of a matrix.*

## 7.34 matrix\_imp.h

[Go to the documentation of this file.](#)

```
00001 #ifndef _GLUCAT_MATRIX_IMP_H
00002 #define _GLUCAT_MATRIX_IMP_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 matrix_imp.h : Implement common matrix functions
00006 -----
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2012 by Paul C. Leopardi
00009 : uBLAS interface contributed by Joerg Walter
```

```

00010 *****
00011
00012 This library is free software: you can redistribute it and/or modify
00013 it under the terms of the GNU Lesser General Public License as published
00014 by the Free Software Foundation, either version 3 of the License, or
00015 (at your option) any later version.
00016
00017 This library is distributed in the hope that it will be useful,
00018 but WITHOUT ANY WARRANTY; without even the implied warranty of
00019 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00020 GNU Lesser General Public License for more details.
00021
00022 You should have received a copy of the GNU Lesser General Public License
00023 along with this library. If not, see <http://www.gnu.org/licenses/>.
00024
00025 *****
00026 This library is based on a prototype written by Arvind Raja and was
00027 licensed under the LGPL with permission of the author. See Arvind Raja,
00028 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00029 in Ablamowicz, Lounesto and Parra (eds.)
00030 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00031 *****
00032 See also Arvind Raja's original header comments in glucat.h
00033 *****/
00034
00035 #include "glucat/errors.h"
00036 #include "glucat/scalar.h"
00037 #include "glucat/matrix.h"
00038
00039 # if defined(_GLUCAT_GCC_IGNORE_UNUSED_LOCAL_TYPEDEFS)
00040 # pragma GCC diagnostic push
00041 # pragma GCC diagnostic ignored "-Wunused-local-typedefs"
00042 # endif
00043 # if defined(_GLUCAT_HAVE_BOOST_SERIALIZATION_ARRAY_WRAPPER_H)
00044 # include <boost/serialization/array_wrapper.hpp>
00045 # endif
00046 #include <boost/numeric/ublas/vector.hpp>
00047 #include <boost/numeric/ublas/vector_proxy.hpp>
00048 #include <boost/numeric/ublas/matrix.hpp>
00049 #include <boost/numeric/ublas/matrix_expression.hpp>
00050 #include <boost/numeric/ublas/matrix_proxy.hpp>
00051 #include <boost/numeric/ublas/matrix_sparse.hpp>
00052 #include <boost/numeric/ublas/operation.hpp>
00053 #include <boost/numeric/ublas/operation_sparse.hpp>
00054
00055 #if defined(_GLUCAT_USE_BINDINGS)
00056 # include <boost/numeric/bindings/lapack/driver/gees.hpp>
00057 # include <boost/numeric/bindings/ublas.hpp>
00058 #endif
00059
00060 #if defined(_GLUCAT_USE_BLAZE)
00061 #include <blaze/Math.h>
00062 #include <blaze/math/DynamicMatrix.h>
00063 #include <blaze/math/DynamicVector.h>
00064 #endif
00065
00066 # if defined(_GLUCAT_GCC_IGNORE_UNUSED_LOCAL_TYPEDEFS)
00067 # pragma GCC diagnostic pop
00068 # endif
00069
00070 #include <set>
00071 #include <vector>
00072
00073 namespace glucat { namespace matrix
00074 {
00075 // References for algorithms:
00076 // [v]: C. F. van Loan and N. Pitsianis, "Approximation with Kronecker products",
00077 // in Linear Algebra for Large Scale and Real-Time Applications, Marc S. Moonen,
00078 // Gene H. Golub, and Bart L. R. Moor (eds.), 1993, pp. 293--314.
00079
00080 template< typename LHS_T, typename RHS_T >
00081 auto
00082 kron(const LHS_T& lhs, const RHS_T& rhs) -> const
00083 RHS_T
00084 {
00085 {
00086 const auto rhs_s1 = rhs.size1();
00087 const auto rhs_s2 = rhs.size2();
00088 auto result = RHS_T(lhs.size1()*rhs_s1, lhs.size2()*rhs_s2);
00089 result.clear();
00090
00091 for (auto
00092 lhs_it1 = lhs.begin1();
00093 lhs_it1 != lhs.end1();
00094 ++lhs_it1)
00095 for (auto
00096 lhs_it2 = lhs_it1.begin();
00097 lhs_it2 != lhs_it1.end();

```

```

00098 ++lhs_it2)
00099 {
00100 const auto start1 = rhs_s1 * lhs_it2.index1();
00101 const auto start2 = rhs_s2 * lhs_it2.index2();
00102 const auto& lhs_val = *lhs_it2;
00103 for (auto
00104 rhs_it1 = rhs.begin1();
00105 rhs_it1 != rhs.end1();
00106 ++rhs_it1)
00107 for (auto
00108 rhs_it2 = rhs_it1.begin();
00109 rhs_it2 != rhs_it1.end();
00110 ++rhs_it2)
00111 result(start1 + rhs_it2.index1(), start2 + rhs_it2.index2()) = lhs_val * *rhs_it2;
00112 }
00113 return result;
00114 }
00115
00117 template< typename LHS_T, typename RHS_T >
00118 auto
00119 mono_kron(const LHS_T& lhs, const RHS_T& rhs) -> const
00120 RHS_T
00121 {
00122 const auto rhs_s1 = rhs.size1();
00123 const auto rhs_s2 = rhs.size2();
00124 const auto dim = lhs.size1()*rhs_s1;
00125 auto result = RHS_T(dim, dim, dim);
00126 result.clear();
00127
00128 for (auto
00129 lhs_it1 = lhs.begin1();
00130 lhs_it1 != lhs.end1();
00131 ++lhs_it1)
00132 {
00133 const auto lhs_it2 = lhs_it1.begin();
00134 const auto start1 = rhs_s1 * lhs_it2.index1();
00135 const auto start2 = rhs_s2 * lhs_it2.index2();
00136 const auto& lhs_val = *lhs_it2;
00137 for (auto
00138 rhs_it1 = rhs.begin1();
00139 rhs_it1 != rhs.end1();
00140 ++rhs_it1)
00141 {
00142 const auto rhs_it2 = rhs_it1.begin();
00143 result(start1 + rhs_it2.index1(), start2 + rhs_it2.index2()) = lhs_val * *rhs_it2;
00144 }
00145 }
00146 return result;
00147 }
00148
00150 template< typename LHS_T, typename RHS_T >
00151 void
00152 nork_range(RHS_T& result,
00153 const typename LHS_T::const_iterator2 lhs_it2,
00154 const RHS_T& rhs,
00155 const typename RHS_T::size_type res_s1,
00156 const typename RHS_T::size_type res_s2)
00157 {
00158 // Definition matches [v] Section 4, Theorem 4.1.
00159 const auto start1 = res_s1 * lhs_it2.index1();
00160 const auto start2 = res_s2 * lhs_it2.index2();
00161 using ublas::range;
00162 const auto& rang1 = range(start1, start1 + res_s1);
00163 const auto& rang2 = range(start2, start2 + res_s2);
00164 using matrix_range_t = ublas::matrix_range<const RHS_T>;
00165 const auto& rhs_range = matrix_range_t(rhs, rang1, rang2);
00166 using Scalar_T = typename RHS_T::value_type;
00167 const auto lhs_val = numeric_traits<Scalar_T>::to_scalar_t(*lhs_it2);
00168 for (auto
00169 rhs_it1 = rhs_range.begin1();
00170 rhs_it1 != rhs_range.end1();
00171 ++rhs_it1)
00172 for (auto
00173 rhs_it2 = rhs_it1.begin();
00174 rhs_it2 != rhs_it1.end();
00175 ++rhs_it2)
00176 result(rhs_it2.index1(), rhs_it2.index2()) += lhs_val * *rhs_it2;
00177 }
00178
00180 template< typename LHS_T, typename RHS_T >
00181 auto
00182 nork(const LHS_T& lhs, const RHS_T& rhs, const bool mono) -> const
00183 RHS_T
00184 {
00185 // nork(A, kron(A, B)) is close to B
00186 // Definition matches [v] Section 4, Theorem 4.1.
00187 const auto lhs_s1 = lhs.size1();

```

```

00188 const auto lhs_s2 = lhs.size2();
00189 const auto rhs_s1 = rhs.size1();
00190 const auto rhs_s2 = rhs.size2();
00191 const auto res_s1 = rhs_s1 / lhs_s1;
00192 const auto res_s2 = rhs_s2 / lhs_s2;
00193 using Scalar_T = typename RHS_T::value_type;
00194 const auto norm_frob2_lhs = norm_frob2(lhs);
00195 if (!mono)
00196 {
00197 using error_t = error<RHS_T>;
00198 if (rhs_s1 == 0)
00199 throw error_t("matrix", "nork: number of rows must not be 0");
00200 if (rhs_s2 == 0)
00201 throw error_t("matrix", "nork: number of cols must not be 0");
00202 if (res_s1 * lhs_s1 != rhs_s1)
00203 throw error_t("matrix", "nork: incompatible numbers of rows");
00204 if (res_s2 * lhs_s2 != rhs_s2)
00205 throw error_t("matrix", "nork: incompatible numbers of cols");
00206 if (norm_frob2_lhs == Scalar_T(0))
00207 throw error_t("matrix", "nork: LHS must not be 0");
00208 }
00209 auto result = RHS_T(res_s1, res_s2);
00210 result.clear();
00211 for (auto
00212 lhs_it1 = lhs.begin1();
00213 lhs_it1 != lhs.end1();
00214 ++lhs_it1)
00215 for (auto
00216 lhs_it2 = lhs_it1.begin();
00217 lhs_it2 != lhs_it1.end();
00218 ++lhs_it2)
00219 if (*lhs_it2 != Scalar_T(0))
00220 nork_range<LHS_T, RHS_T>(result, lhs_it2, rhs, res_s1, res_s2);
00221 result /= norm_frob2_lhs;
00222 return result;
00223 }
00224
00226 template< typename LHS_T, typename RHS_T >
00227 auto
00228 signed_perm_nork(const LHS_T& lhs, const RHS_T& rhs) -> const
00229 RHS_T
00230 {
00231 // signed_perm_nork(A, kron(A, B)) is close to B
00232 // Definition matches [v] Section 4, Theorem 4.1.
00233 const auto lhs_s1 = lhs.size1();
00234 const auto lhs_s2 = lhs.size2();
00235 const auto rhs_s1 = rhs.size1();
00236 const auto rhs_s2 = rhs.size2();
00237 const auto res_s1 = rhs_s1 / lhs_s1;
00238 const auto res_s2 = rhs_s2 / lhs_s2;
00239 using Scalar_T = typename RHS_T::value_type;
00240 const auto norm_frob2_lhs = Scalar_T(double(lhs_s1));
00241 auto result = RHS_T(res_s1, res_s2);
00242 result.clear();
00243 for (auto
00244 lhs_it1 = lhs.begin1();
00245 lhs_it1 != lhs.end1();
00246 ++lhs_it1)
00247 {
00248 const auto lhs_it2 = lhs_it1.begin();
00249 nork_range<LHS_T, RHS_T>(result, lhs_it2, rhs, res_s1, res_s2);
00250 }
00251 result /= norm_frob2_lhs;
00252 return result;
00253 }
00254
00256 template< typename Matrix_T >
00257 auto
00258 nnz(const Matrix_T& m) -> typename Matrix_T::size_type
00259 {
00260 using size_t = typename Matrix_T::size_type;
00261 auto result = size_t(0);
00262 for (auto
00263 it1 = m.begin1();
00264 it1 != m.end1();
00265 ++it1)
00266 for (auto& entry : it1)
00267 if (entry != 0)
00268 ++result;
00269 return result;
00270 }
00271
00273 template< typename Matrix_T >
00274 auto
00275 isinf(const Matrix_T& m) -> bool
00276 {
00277 using Scalar_T = typename Matrix_T::value_type;

```

```

00278 for (auto
00279 it1 = m.begin1();
00280 it1 != m.end1();
00281 ++it1)
00282 for (auto& entry : it1)
00283 if (numeric_traits<Scalar_T>::isInf(entry))
00284 return true;
00285 return false;
00286 }
00287
00288
00290 template< typename Matrix_T >
00291 auto
00292 isnan(const Matrix_T& m) -> bool
00293 {
00294 using Scalar_T = typename Matrix_T::value_type;
00295 for (auto
00296 it1 = m.begin1();
00297 it1 != m.end1();
00298 ++it1)
00299 for (auto& entry : it1)
00300 if (numeric_traits<Scalar_T>::isNaN(entry))
00301 return true;
00302 return false;
00303 }
00304
00305
00307 template< typename Matrix_T >
00308 inline
00309 auto
00310 unit(const typename Matrix_T::size_type dim) -> const
00311 Matrix_T
00312 {
00313 using Scalar_T = typename Matrix_T::value_type;
00314 return ublas::identity_matrix<Scalar_T>(dim);
00315 }
00316
00318 template< typename LHS_T, typename RHS_T >
00319 auto
00320 mono_prod(const ublas::matrix_expression<LHS_T>& lhs,
00321 const ublas::matrix_expression<RHS_T>& rhs) -> const typename RHS_T::expression_type
00322 {
00323 using rhs_expression_t = const RHS_T;
00324 using matrix_row_t = typename ublas::matrix_row<rhs_expression_t>;
00325
00326 const auto dim = lhs().size1();
00327 // The following assumes that RHS_T is a sparse matrix type.
00328 auto result = RHS_T(dim, dim, dim);
00329 for (auto
00330 lhs_row = lhs().begin1();
00331 lhs_row != lhs().end1();
00332 ++lhs_row)
00333 {
00334 const auto& lhs_it = lhs_row.begin();
00335 if (lhs_it != lhs_row.end())
00336 {
00337 const auto& rhs_row = matrix_row_t(rhs(), lhs_it.index2());
00338 const auto& rhs_it = rhs_row.begin();
00339 if (rhs_it != rhs_row.end())
00340 result(lhs_it.index1(), rhs_it.index()) = (*lhs_it) * (*rhs_it);
00341 }
00342 }
00343 return result;
00344 }
00345
00347 template< typename LHS_T, typename RHS_T >
00348 inline
00349 auto
00350 sparse_prod(const ublas::matrix_expression<LHS_T>& lhs,
00351 const ublas::matrix_expression<RHS_T>& rhs) -> const typename RHS_T::expression_type
00352 {
00353 using expression_t = typename RHS_T::expression_type;
00354 return ublas::sparse_prod<expression_t>(lhs(), rhs());
00355 }
00356
00358 template< typename LHS_T, typename RHS_T >
00359 inline
00360 auto
00361 prod(const ublas::matrix_expression<LHS_T>& lhs,
00362 const ublas::matrix_expression<RHS_T>& rhs) -> const typename RHS_T::expression_type
00363 {
00364 const auto dim = lhs().size1();
00365 RHS_T result(dim, dim);
00366 ublas::axpy_prod(lhs, rhs, result, true);
00367 return result;
00368 }
00369

```

```

00371 template< typename Scalar_T, typename LHS_T, typename RHS_T >
00372 auto
00373 inner(const LHS_T& lhs, const RHS_T& rhs) -> Scalar_T
00374 {
00375 auto result = Scalar_T(0);
00376 for (auto
00377 lhs_it1 = lhs.begin1();
00378 lhs_it1 != lhs.end1();
00379 ++lhs_it1)
00380 for (auto
00381 lhs_it2 = lhs_it1.begin();
00382 lhs_it2 != lhs_it1.end();
00383 ++lhs_it2)
00384 {
00385 const auto& rhs_val = rhs(lhs_it2.index1(), lhs_it2.index2());
00386 if (rhs_val != Scalar_T(0))
00387 result += (*lhs_it2) * rhs_val;
00388 }
00389 return result / lhs.size1();
00390 }
00391
00393 template< typename Matrix_T >
00394 auto
00395 norm_frob2(const Matrix_T& val) -> typename Matrix_T::value_type
00396 {
00397 using Scalar_T = typename Matrix_T::value_type;
00398
00399 auto result = Scalar_T(0);
00400 for (auto
00401 val_it1 = val.begin1();
00402 val_it1 != val.end1();
00403 ++val_it1)
00404 for (auto& val_entry : val_it1)
00405 {
00406 if (numeric_traits<Scalar_T>::isNaN(val_entry))
00407 return numeric_traits<Scalar_T>::NaN();
00408 result += val_entry * val_entry;
00409 }
00410 return result;
00411 }
00412
00414 template< typename Matrix_T >
00415 auto
00416 trace(const Matrix_T& val) -> typename Matrix_T::value_type
00417 {
00418 using Scalar_T = typename Matrix_T::value_type;
00419
00420 auto result = Scalar_T(0);
00421 auto dim = val.size1();
00422 for (auto
00423 ndx = decltype(dim)(0);
00424 ndx != dim;
00425 ++ndx)
00426 {
00427 const Scalar_T crd = val(ndx, ndx);
00428 if (numeric_traits<Scalar_T>::isNaN(crd))
00429 return numeric_traits<Scalar_T>::NaN();
00430 result += crd;
00431 }
00432 return result;
00433 }
00434
00435 #if defined(_GLUCAT_USE_BINDINGS)
00437 template< typename Matrix_T >
00438 static
00439 auto
00440 to_lapack(const Matrix_T& val) -> ublas::matrix<double, ublas::column_major>
00441 {
00442 const auto s1 = val.size1();
00443 const auto s2 = val.size2();
00444
00445 using lapack_matrix_t = typename ublas::matrix<double, ublas::column_major>;
00446 auto result = lapack_matrix_t(s1, s2);
00447 result.clear();
00448
00449 using Scalar_T = typename Matrix_T::value_type;
00450 using traits_t = numeric_traits<Scalar_T>;
00451
00452 for (auto
00453 val_it1 = val.begin1();
00454 val_it1 != val.end1();
00455 ++val_it1)
00456 for (auto
00457 val_it2 = val_it1.begin();
00458 val_it2 != val_it1.end();
00459 ++val_it2)
00460 result(val_it2.index1(), val_it2.index2()) = traits_t::to_double(*val_it2);

```

```

00461
00462 return result;
00463 }
00464 #endif
00465
00466 #if defined(_GLUCAT_USE_BLAZE)
00467 template< typename Matrix_T >
00468 static
00469 auto
00470 to_blaze(const Matrix_T& val) -> blaze::DynamicMatrix<double, blaze::rowMajor>
00471 {
00472 {
00473 const auto s1 = val.size1();
00474 const auto s2 = val.size2();
00475
00476 using blaze_matrix_t = typename blaze::DynamicMatrix<double, blaze::rowMajor>;
00477 auto result = blaze_matrix_t(s1, s2);
00478
00479 using Scalar_T = typename Matrix_T::value_type;
00480 using traits_t = numeric_traits<Scalar_T>;
00481
00482 for (auto
00483 val_it1 = val.begin1();
00484 val_it1 != val.end1();
00485 ++val_it1)
00486 for (auto
00487 val_it2 = val_it1.begin();
00488 val_it2 != val_it1.end();
00489 ++val_it2)
00490 result(val_it2.index1(), val_it2.index2()) = traits_t::to_double(*val_it2);
00491
00492 return result;
00493 }
00494 #endif
00495
00496 template< typename Matrix_T >
00497 auto
00500 eigenvalues(const Matrix_T& val) -> std::vector< std::complex<double> >
00501 {
00502 using complex_t = std::complex<double>;
00503 using complex_vector_t = typename std::vector<complex_t>;
00504
00505 const auto dim = val.size1();
00506 auto lambda = complex_vector_t(dim);
00507
00508 #if defined(_GLUCAT_USE_BINDINGS)
00509 namespace lapack = boost::numeric::bindings::lapack;
00510 using lapack_matrix_t = typename ublas::matrix<double, ublas::column_major>;
00511
00512 auto T = to_lapack(val);
00513 auto V = T;
00514 using vector_t = typename ublas::vector<double>;
00515 auto real_lambda = vector_t(dim);
00516 auto imag_lambda = vector_t(dim);
00517 fortran_int_t sdim = 0;
00518
00519 lapack::gees('N', 'N', nullptr, T, sdim, real_lambda, imag_lambda, V);
00520
00521 for (auto
00522 k = decltype(dim)(0);
00523 k != dim;
00524 ++k)
00525 lambda[k] = complex_t(real_lambda[k], imag_lambda[k]);
00526 #endif
00527 #if defined(_GLUCAT_USE_BLAZE)
00528 using blaze_matrix_t = typename blaze::DynamicMatrix<double, blaze::rowMajor>;
00529 using complex_t = std::complex<double>;
00530 using blaze_complex_vector_t = blaze::DynamicVector<complex_t, blaze::columnVector>;
00531
00532 auto blaze_val = to_blaze(val);
00533 auto blaze_lambda = blaze_complex_vector_t(dim);
00534 blaze::geev(blaze_val, blaze_lambda);
00535
00536 for (auto
00537 k = decltype(dim)(0);
00538 k != dim;
00539 ++k)
00540 lambda[k] = blaze_lambda[k];
00541 #endif
00542 return lambda;
00543 }
00544
00545 template< typename Matrix_T >
00546 auto
00548 classify_eigenvalues(const Matrix_T& val) -> eig_genus<Matrix_T>
00549 {
00550 using Scalar_T = typename Matrix_T::value_type;

```

```

00551 eig_genus<Matrix_T> result;
00552
00553 using complex_t = std::complex<double>;
00554 using complex_vector_t = typename std::vector<complex_t>;
00555 auto lambda = eigenvalues(val);
00556
00557 std::set<double> arg_set;
00558
00559 using vector_index_t = typename complex_vector_t::size_type;
00560 const auto dim = lambda.size();
00561 static const auto epsilon =
00562 std::max(std::numeric_limits<double>::epsilon(),
00563 numeric_traits<Scalar_T>::to_double(std::numeric_limits<Scalar_T>::epsilon()));
00564 static const auto zero_eig_tol = 4096.0*epsilon;
00565
00566 bool neg_real_eig_found = false;
00567 bool imag_eig_found = false;
00568 bool zero_eig_found = false;
00569
00570 for (auto
00571 k = decltype(dim)(0);
00572 k != dim;
00573 ++k)
00574 {
00575 const auto lambda_k = lambda[k];
00576 arg_set.insert(std::arg(lambda_k));
00577
00578 const auto real_lambda_k = std::real(lambda_k);
00579 const auto imag_lambda_k = std::imag(lambda_k);
00580 const auto norm_tol = 4096.0*epsilon*std::norm(lambda_k);
00581
00582 if (!neg_real_eig_found &&
00583 real_lambda_k < -epsilon &&
00584 (imag_lambda_k == 0.0 ||
00585 imag_lambda_k * imag_lambda_k < norm_tol))
00586 neg_real_eig_found = true;
00587 if (!imag_eig_found &&
00588 imag_lambda_k > epsilon &&
00589 (real_lambda_k == 0.0 ||
00590 real_lambda_k * real_lambda_k < norm_tol))
00591 imag_eig_found = true;
00592 if (!zero_eig_found &&
00593 std::norm(lambda_k) < zero_eig_tol)
00594 zero_eig_found = true;
00595 }
00596
00597 if (zero_eig_found)
00598 result.m_is_singular = true;
00599
00600 static const auto pi = numeric_traits<double>::pi();
00601 if (neg_real_eig_found)
00602 {
00603 if (imag_eig_found)
00604 result.m_eig_case = both_eigs;
00605 else
00606 {
00607 result.m_eig_case = neg_real_eigs;
00608 result.m_safe_arg = Scalar_T(-pi / 2.0);
00609 }
00610 }
00611
00612 if (result.m_eig_case == both_eigs)
00613 {
00614 auto arg_it = arg_set.begin();
00615 auto first_arg = *arg_it;
00616 auto best_arg = first_arg;
00617 auto best_diff = 0.0;
00618 auto previous_arg = first_arg;
00619 for (++arg_it;
00620 arg_it != arg_set.end();
00621 ++arg_it)
00622 {
00623 const auto arg_diff = *arg_it - previous_arg;
00624 if (arg_diff > best_diff)
00625 {
00626 best_diff = arg_diff;
00627 best_arg = previous_arg;
00628 }
00629 previous_arg = *arg_it;
00630 }
00631 const auto arg_diff = first_arg + 2.0*pi - previous_arg;
00632 if (arg_diff > best_diff)
00633 {
00634 best_diff = arg_diff;
00635 best_arg = previous_arg;
00636 }
00637 result.m_safe_arg = Scalar_T(pi - (best_arg + best_diff / 2.0));

```



```

00638 }
00639 return result;
00640 }
00641 } }
00642
00643 #endif // _GLUCAT_MATRIX_IMP_H

```

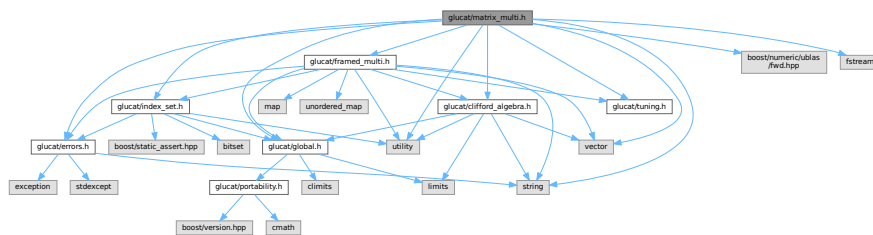
## 7.35 glucat/matrix\_multi.h File Reference

```

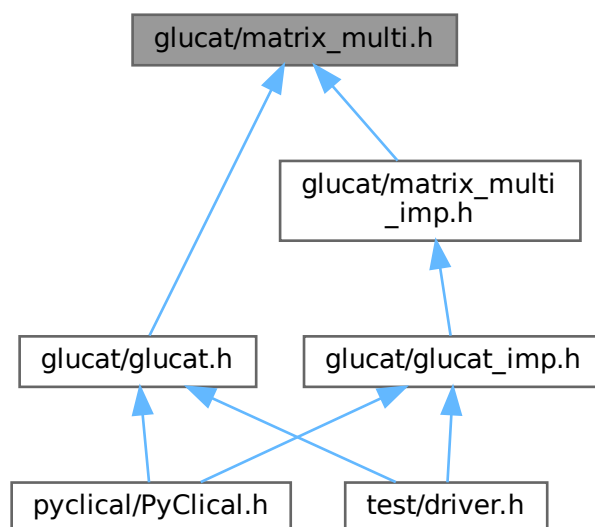
#include "glucat/global.h"
#include "glucat/errors.h"
#include "glucat/index_set.h"
#include "glucat/clifford_algebra.h"
#include "glucat/tuning.h"
#include "glucat/framed_multi.h"
#include <boost/numeric/ublas/fwd.hpp>
#include <fstream>
#include <string>
#include <utility>
#include <vector>

```

Include dependency graph for matrix\_multi.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class `glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >`  
*A `matrix_multi<Scalar_T,LO,HI,Tune_P>` is a matrix approximation to a multivector.*
- struct `std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > >`  
*Numeric limits for `matrix_multi` inherit limits for the corresponding scalar type.*

## Namespaces

- namespace `glucat`
- namespace `std`

## Functions

- template<typename Scalar\_T , const `index_t` LO, const `index_t` HI, typename Tune\_P >  
auto `glucat::operator*` (const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &lhs, const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &rhs) -> const `matrix_multi< Scalar_T, LO, HI, Tune_P >`  
*Geometric product.*
- template<typename Scalar\_T , const `index_t` LO, const `index_t` HI, typename Tune\_P >  
auto `glucat::operator^` (const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &lhs, const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &rhs) -> const `matrix_multi< Scalar_T, LO, HI, Tune_P >`  
*Outer product.*
- template<typename Scalar\_T , const `index_t` LO, const `index_t` HI, typename Tune\_P >  
auto `glucat::operator&` (const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &lhs, const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &rhs) -> const `matrix_multi< Scalar_T, LO, HI, Tune_P >`  
*Inner product.*
- template<typename Scalar\_T , const `index_t` LO, const `index_t` HI, typename Tune\_P >  
auto `glucat::operator%` (const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &lhs, const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &rhs) -> const `matrix_multi< Scalar_T, LO, HI, Tune_P >`  
*Left contraction.*
- template<typename Scalar\_T , const `index_t` LO, const `index_t` HI, typename Tune\_P >  
auto `glucat::star` (const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &lhs, const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &rhs) -> `Scalar_T`  
*Hestenes scalar product.*
- template<typename Scalar\_T , const `index_t` LO, const `index_t` HI, typename Tune\_P >  
auto `glucat::operator/` (const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &lhs, const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &rhs) -> const `matrix_multi< Scalar_T, LO, HI, Tune_P >`  
*Geometric quotient.*
- template<typename Scalar\_T , const `index_t` LO, const `index_t` HI, typename Tune\_P >  
auto `glucat::operator|` (const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &lhs, const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &rhs) -> const `matrix_multi< Scalar_T, LO, HI, Tune_P >`  
*Transformation via twisted adjoint action.*
- template<typename Scalar\_T , const `index_t` LO, const `index_t` HI, typename Tune\_P >  
auto `glucat::operator>>` (std::istream &s, `matrix_multi< Scalar_T, LO, HI, Tune_P >` &val) -> std::istream &  
*Read multivector from input.*
- template<typename Scalar\_T , const `index_t` LO, const `index_t` HI, typename Tune\_P >  
auto `glucat::operator<<` (std::ostream &os, const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &val) -> std::ostream &  
*Write multivector to output.*
- template<typename Scalar\_T , const `index_t` LO, const `index_t` HI, typename Tune\_P >  
auto `glucat::reframe` (const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &lhs, const `matrix_multi< Scalar_T, LO, HI, Tune_P >` &rhs, `matrix_multi< Scalar_T, LO, HI, Tune_P >` &lhs\_reframed, `matrix_multi< Scalar_T, LO, HI, Tune_P >` &rhs\_reframed) -> const `index_set< LO, HI >`

*Find a common frame for operands of a binary operator.*

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO,`  
`HI, Tune_P > &i, bool prechecked) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >`

*Square root of multivector with specified complexifier.*

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::matrix_sqrt (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_↵`  
`_T, LO, HI, Tune_P > &i, const index_t level) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >`

*Square root of multivector with specified complexifier.*

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_T, LO,`  
`HI, Tune_P > &i, bool prechecked) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >`

*Natural logarithm of multivector with specified complexifier.*

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::matrix_log (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val, const matrix_multi< Scalar_↵`  
`_T, LO, HI, Tune_P > &i, const index_t level) -> const matrix_multi< Scalar_T, LO, HI, Tune_P >`

*Natural logarithm of multivector with specified complexifier.*

- `template<typename Scalar_T , const index_t LO, const index_t HI, typename Tune_P >`  
`auto glucat::exp (const matrix_multi< Scalar_T, LO, HI, Tune_P > &val) -> const matrix_multi< Scalar_T,`  
`LO, HI, Tune_P >`

*Exponential of multivector.*

## 7.36 matrix\_multi.h

[Go to the documentation of this file.](#)

```
00001 #ifndef _GLUCAT_MATRIX_MULTI_H
00002 #define _GLUCAT_MATRIX_MULTI_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 matrix_multi.h : Declare a class for the matrix representation of a multivector
00006 -----
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2021 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 #include "glucat/global.h"
00035 #include "glucat/errors.h"
00036 #include "glucat/index_set.h"
00037 #include "glucat/clifford_algebra.h"
00038 #include "glucat/tuning.h"
00039 #include "glucat/framed_multi.h"
00040
00041 #include <boost/numeric/ublas/fwd.hpp>
00042
00043 #include <fstream>
00044 #include <string>
```

```

00045 #include <utility>
00046 #include <vector>
00047
00048 namespace glucat
00049 {
00050 namespace ublas = boost::numeric::ublas;
00051
00052 // Forward declarations for friends
00053
00054 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00055 class framed_multi; // forward
00056
00057 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00058 class matrix_multi; // forward
00059
00060 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00061 auto
00062 operator* (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;
00064
00066 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00067 auto
00068 operator^ (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;
00069
00071 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00072 auto
00073 operator& (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;
00074
00076 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00077 auto
00078 operator% (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;
00079
00081 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00082 auto
00083 star(const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs)
-> Scalar_T;
00084
00086 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00087 auto
00088 operator/ (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;
00089
00091 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00092 auto
00093 operator| (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;
00094
00096 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00097 auto
00098 operator» (std::istream& s, matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::istream&;
00099
00101 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00102 auto
00103 operator« (std::ostream& os, const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::ostream&;
00104
00106 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00107 auto
00108 reframe (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs,
matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs_reframed,
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs_reframed) -> const index_set<LO,HI>;
00109
00112 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00113 auto
00114 sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val, const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
bool prechecked) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;
00115
00117 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00118 auto
00119 matrix_sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;
00122
00124 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00125 auto
00126 log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val, const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
bool prechecked) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;
00127
00129 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00130 auto
00131 matrix_log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;
00133
00134

```

```

00136 template< typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI,
typename Tune_P = tuning<> >
00137 class matrix_multi :
00138 public clifford_algebra< Scalar_T, index_set<LO,HI>, matrix_multi<Scalar_T,LO,HI,Tune_P> >
00139 {
00140 public:
00141 using multivector_t = matrix_multi;
00142 using matrix_multi_t = multivector_t;
00143 using scalar_t = Scalar_T;
00144 using tune_p = Tune_P;
00145 using index_set_t = index_set<LO, HI>;
00146 using term_t = std::pair<const index_set_t, Scalar_T>;
00147 using vector_t = std::vector<Scalar_T>;
00148 using error_t = error<multivector_t>;
00149 using framed_multi_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00150 template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
00151 friend class framed_multi;
00152 template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
00153 friend class matrix_multi;
00154
00155 private:
00156 using orientation_t = ublas::row_major;
00157 using basis_matrix_t = ublas::compressed_matrix<int, orientation_t>;
00158 using matrix_t = ublas::matrix<Scalar_T, orientation_t>;
00159 using matrix_index_t = typename matrix_t::size_type;
00160
00161 public:
00162 static auto classname() -> const std::string;
00163 ~matrix_multi() override = default;
00164 matrix_multi();
00165 template< typename Other_Scalar_T >
00166 matrix_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val);
00167 template< typename Other_Scalar_T >
00168 matrix_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val,
00169 const index_set_t frm, const bool prechecked = false);
00170 matrix_multi(const multivector_t& val,
00171 const index_set_t frm, const bool prechecked = false);
00172 matrix_multi(const index_set_t ist, const Scalar_T& crd = Scalar_T(1));
00173 matrix_multi(const index_set_t ist, const Scalar_T& crd,
00174 const index_set_t frm, const bool prechecked = false);
00175 matrix_multi(const Scalar_T& scr, const index_set_t frm = index_set_t());
00176 matrix_multi(const int scr, const index_set_t frm = index_set_t());
00177 matrix_multi(const vector_t& vec,
00178 const index_set_t frm, const bool prechecked = false);
00179 matrix_multi(const std::string& str);
00180 matrix_multi(const std::string& str,
00181 const index_set_t frm, const bool prechecked = false);
00182 matrix_multi(const char* str)
00183 { *this = matrix_multi(std::string(str)); };
00184 matrix_multi(const char* str,
00185 const index_set_t frm, const bool prechecked = false)
00186 { *this = matrix_multi(std::string(str), frm, prechecked); };
00187 template< typename Other_Scalar_T >
00188 matrix_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val);
00189 template< typename Other_Scalar_T >
00190 matrix_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val,
00191 const index_set_t frm, const bool prechecked = false);
00192 auto fast_matrix_multi(const index_set_t frm) const -> const matrix_multi_t;
00193 template< typename Other_Scalar_T >
00194 auto fast_framed_multi() const -> const framed_multi<Other_Scalar_T,LO,HI,Tune_P>;
00195
00196 private:
00197 template< typename Matrix_T >
00198 matrix_multi(const Matrix_T& mtx, const index_set_t frm);
00199 matrix_multi(const matrix_t& mtx, const index_set_t frm);
00200 auto basis_element(const index_set<LO,HI>& ist) const -> const basis_matrix_t;
00201
00202 public:
00203 _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS
00204
00205 auto operator= (const multivector_t& rhs) -> multivector_t&;
00206
00207 static auto random(const index_set_t frm, Scalar_T fill = Scalar_T(1)) -> const matrix_multi_t;
00208
00209 // Friend declarations
00210
00211 friend auto
00212 operator* <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
00213 friend auto
00214 operator^ <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
00215 friend auto
00216 operator& <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
00217 friend auto
00218 operator% <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
00219 friend auto

```

```

00244 star <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> Scalar_T;
00245 friend auto
00246 operator/ <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
00247 friend auto
00248 operator| <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
00249
00250 friend auto
00251 operator« <>(std::istream& s, multivector_t& val) -> std::istream&;
00252 friend auto
00253 operator« <>(std::ostream& os, const multivector_t& val) -> std::ostream&;
00254 template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
00255 friend auto
00256 reframe (const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& lhs, const
matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& rhs,
00257 matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& lhs_reframed,
matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& rhs_reframed) -> const
index_set<Other_LO,Other_HI>;
00258 template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
00259 friend auto
00260 matrix_sqrt (const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& val,
00261 const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& i,
00262 const index_t level)
00263 -> const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>;
00264 template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
00265 friend auto
00266 matrix_log (const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& val,
00267 const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& i,
00268 const index_t level)
00269 -> const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>;
00270
00272 auto operator+= (const term_t& rhs) -> multivector_t&;
00273
00274 private:
00275 // Data members
00276
00278 index_set_t m_frame;
00280 matrix_t m_matrix;
00281 };
00282
00283 // Non-members
00284
00286 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00287 auto
00288 exp(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>;
00289
00290 }
00291
00292 namespace std
00293 {
00295 template < typename Scalar_T, const glucat::index_t LO, const glucat::index_t HI, typename Tune_P >
00296 struct numeric_limits< glucat::matrix_multi<Scalar_T,LO,HI,Tune_P> > :
00297 public numeric_limits<Scalar_T>
00298 { };
00299 }
00300 #endif // _GLUCAT_MATRIX_MULTI_H

```

## 7.37 glucat/matrix\_multi\_imp.h File Reference

```

#include "glucat/matrix_multi.h"
#include "glucat/scalar.h"
#include "glucat/generation.h"
#include "glucat/matrix.h"
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/matrix_expression.hpp>
#include <boost/numeric/ublas/matrix_proxy.hpp>
#include <boost/numeric/ublas/matrix_sparse.hpp>
#include <boost/numeric/ublas/operation.hpp>
#include <boost/numeric/ublas/operation_sparse.hpp>
#include <boost/numeric/ublas/triangular.hpp>
#include <boost/numeric/ublas/lu.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <fstream>

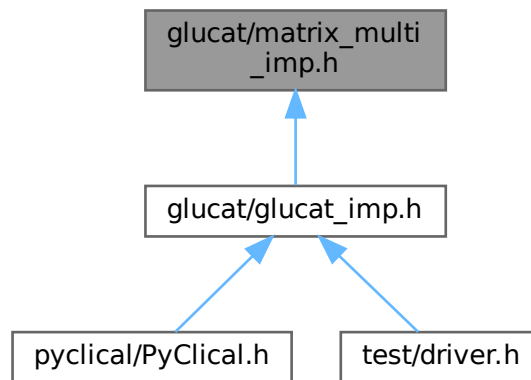
```

```
#include <iomanip>
#include <array>
#include <iostream>
```

Include dependency graph for matrix\_multi\_imp.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [glucat::basis\\_table< Scalar\\_T, LO, HI, Matrix\\_T >](#)  
*Table of basis elements used as a cache by basis\_element()*
- struct [pade::pade\\_sqrt\\_numer< Scalar\\_T >](#)  
*Coefficients of numerator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)*
- struct [pade::pade\\_sqrt\\_denom< Scalar\\_T >](#)  
*Coefficients of denominator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)*
- struct [pade::pade\\_sqrt\\_numer< float >](#)
- struct [pade::pade\\_sqrt\\_denom< float >](#)
- struct [pade::pade\\_sqrt\\_numer< long double >](#)
- struct [pade::pade\\_sqrt\\_denom< long double >](#)
- struct [pade::pade\\_sqrt\\_numer< dd\\_real >](#)
- struct [pade::pade\\_sqrt\\_denom< dd\\_real >](#)
- struct [pade::pade\\_sqrt\\_numer< qd\\_real >](#)
- struct [pade::pade\\_sqrt\\_denom< qd\\_real >](#)
- struct [pade::pade\\_log\\_numer< Scalar\\_T >](#)  
*Coefficients of numerator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)*
- struct [pade::pade\\_log\\_denom< Scalar\\_T >](#)  
*Coefficients of denominator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)*

- struct [pade::pade\\_log\\_numer](#)< float >
- struct [pade::pade\\_log\\_denom](#)< float >
- struct [pade::pade\\_log\\_numer](#)< long double >
- struct [pade::pade\\_log\\_denom](#)< long double >
- struct [pade::pade\\_log\\_numer](#)< dd\_real >
- struct [pade::pade\\_log\\_denom](#)< dd\_real >
- struct [pade::pade\\_log\\_numer](#)< qd\_real >
- struct [pade::pade\\_log\\_denom](#)< qd\_real >

## Namespaces

- namespace [glucat](#)
- namespace [pade](#)

## Functions

- auto [glucat::offset\\_level](#) (const [index\\_t](#) p, const [index\\_t](#) q) -> [index\\_t](#)  
*Determine the log2 dim corresponding to signature p, q.*
- template<typename Matrix\_Index\_T, const [index\\_t](#) LO, const [index\\_t](#) HI>  
static auto [glucat::folded\\_dim](#) (const [index\\_set](#)< LO, HI > &sub) -> Matrix\_Index\_T  
*Determine the matrix dimension of the fold of a subalgebra.*
- template<typename Scalar\_T, const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::reframe](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs, [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs\_reframed, [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs\_reframed) -> const [index\\_set](#)< LO, HI >  
*Find a common frame for operands of a binary operator.*
- template<typename Scalar\_T, const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator\\*](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >  
*Geometric product.*
- template<typename Scalar\_T, const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator^](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >  
*Outer product.*
- template<typename Scalar\_T, const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator&](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >  
*Inner product.*
- template<typename Scalar\_T, const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator%](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >  
*Left contraction.*
- template<typename Scalar\_T, const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::star](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> Scalar\_T  
*Hestenes scalar product.*
- template<typename Scalar\_T, const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator/](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >  
*Geometric quotient.*
- template<typename Scalar\_T, const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator|](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &lhs, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &rhs) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >



*Transformation via twisted adjoint action.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator<<](#) (std::ostream &os, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val) -> std::ostream &

*Write multivector to output.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::operator>>](#) (std::istream &s, [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val) -> std::istream &

*Read multivector from input.*

- template<typename Multivector\_T , typename Matrix\_T , typename Basis\_Matrix\_T >  
static auto [glucat::fast](#) (const Matrix\_T &X, [index\\_t](#) level) -> Multivector\_T

*Inverse generalized Fast Fourier Transform.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P , const size\_t Size>  
static auto [glucat::pade\\_approx](#) (const std::array< Scalar\_T, Size > &numer, const std::array< Scalar\_T, Size > &denom, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &X) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >

*Pade' approximation.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
static void [glucat::db\\_step](#) ([matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &M, [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &Y)

*Single step of product form of Denman-Beavers square root iteration.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
static auto [glucat::db\\_sqrt](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val, Scalar\_T norm\_tol=std::pow(std::numeric\_limits< Scalar\_T >::epsilon(), 4)) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >

*Product form of Denman-Beavers square root iteration.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
static auto [glucat::cr\\_sqrt](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val, Scalar\_T norm\_Y\_tol=std::pow(std::numeric\_limits< Scalar\_T >::epsilon(), 1)) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >

*Cyclic reduction square root iteration.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::matrix\\_sqrt](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &i, const [index\\_t](#) level) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >

*Square root of multivector with specified complexifier.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::sqrt](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &i, bool prechecked) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >

*Square root of multivector with specified complexifier.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
static auto [glucat::pade\\_log](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >

*Pade' approximation of log.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
static auto [glucat::cascade\\_log](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >

*Incomplete square root cascade and Pade' approximation of log.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::matrix\\_log](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &i, const [index\\_t](#) level) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >

*Natural logarithm of multivector with specified complexifier.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::log](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val, const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &i, bool prechecked) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >

*Natural logarithm of multivector with specified complexifier.*

- template<typename Scalar\_T , const [index\\_t](#) LO, const [index\\_t](#) HI, typename Tune\_P >  
auto [glucat::exp](#) (const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P > &val) -> const [matrix\\_multi](#)< Scalar\_T, LO, HI, Tune\_P >

*Exponential of multivector.*

## 7.38 matrix\_multi\_imp.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_MATRIX_MULTI_IMP_H
00002 #define _GLUCAT_MATRIX_MULTI_IMP_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 matrix_multi_imp.h : Implement the matrix representation of a multivector
00006 -----
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2021 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 #include "glucat/matrix_multi.h"
00035
00036 #include "glucat/scalar.h"
00037 #include "glucat/generation.h"
00038 #include "glucat/matrix.h"
00039
00040 # if defined(_GLUCAT_GCC_IGNORE_UNUSED_LOCAL_TYPEDEFS)
00041 # pragma GCC diagnostic push
00042 # pragma GCC diagnostic ignored "-Wunused-local-typedefs"
00043 # endif
00044 # if defined(_GLUCAT_HAVE_BOOST_SERIALIZATION_ARRAY_WRAPPER_H)
00045 # include <boost/serialization/array_wrapper.hpp>
00046 # endif
00047 #include <boost/numeric/ublas/matrix.hpp>
00048 #include <boost/numeric/ublas/matrix_expression.hpp>
00049 #include <boost/numeric/ublas/matrix_proxy.hpp>
00050 #include <boost/numeric/ublas/matrix_sparse.hpp>
00051 #include <boost/numeric/ublas/operation.hpp>
00052 #include <boost/numeric/ublas/operation_sparse.hpp>
00053 #include <boost/numeric/ublas/triangular.hpp>
00054 #include <boost/numeric/ublas/lu.hpp>
00055 #include <boost/numeric/ublas/io.hpp>
00056 # if defined(_GLUCAT_GCC_IGNORE_UNUSED_LOCAL_TYPEDEFS)
00057 # pragma GCC diagnostic pop
00058 # endif
00059
00060 #include <fstream>
00061 #include <iomanip>
00062 #include <array>
00063 #include <iostream>
00064
00065 namespace glucat
00066 {
00067 // References for algorithms:
00068 // [CHKL]:
00069 // [L]: Pertti Lounesto, "Clifford algebras and spinors", Cambridge UP, 1997.
00070 // [MB]: Beatrice Meini, "The Matrix Square Root From a New Functional Perspective:
00071 // Theoretical Results and Computational Issues", SIAM Journal on
00072 // Matrix Analysis and Applications 26(2):362-376, 2004.
00073 // [P]: Ian R. Porteous, "Clifford algebras and the classical groups", Cambridge UP, 1995.
00074
00075 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00076 auto
00077 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00078 classname() -> const std::string
00079 { return "matrix_multi"; }
00080
00081
00082 // Reference: [P] Table 15.27, p 133
00083 inline

```

```

00085 auto
00086 offset_level(const index_t p, const index_t q) -> index_t
00087 {
00088 // Offsets between the log2 of the matrix dimension for the current signature
00089 // and that of the real superalgebra
00090 static const std::array<int, 8> offset_log2_dim = {0, 1, 0, 1, 1, 2, 1, 1};
00091 const index_t bott = pos_mod(p-q, 8);
00092 return (p+q)/2 + offset_log2_dim[bott];
00093 }
00094
00095 // Reference: [P] Table 15.27, p 133
00096 template< typename Matrix_Index_T, const index_t LO, const index_t HI >
00097 inline
00098 static
00099 auto
00100 folded_dim(const index_set<LO,HI>& sub) -> Matrix_Index_T
00101 { return 1 « offset_level(sub.count_pos(), sub.count_neg()); }
00102
00103 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00104 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00105 matrix_multi()
00106 : m_frame(index_set_t()),
00107 m_matrix(matrix_t(1, 1))
00108 { this->m_matrix.clear(); }
00109
00110 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00111 template< typename Other_Scalar_T >
00112 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00113 matrix_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val)
00114 : m_frame(val.m_frame), m_matrix(val.m_matrix.size(), val.m_matrix.size())
00115 {
00116 this->m_matrix.clear();
00117 for (auto
00118 val_it1 = val.m_matrix.begin();
00119 val_it1 != val.m_matrix.end();
00120 ++val_it1)
00121 for (auto
00122 val_it2 = val_it1.begin();
00123 val_it2 != val_it1.end();
00124 ++val_it2)
00125 this->m_matrix(val_it2.index1(), val_it2.index2()) =
00126 numeric_traits<Scalar_T>::to_scalar_t(*val_it2);
00127 }
00128
00129 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00130 template< typename Other_Scalar_T >
00131 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00132 matrix_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val, const index_set_t frm, const bool
00133 prechecked)
00134 : m_frame(frm)
00135 {
00136 if (frm != val.m_frame)
00137 *this = multivector_t(framed_multi_t(val), frm);
00138 else
00139 {
00140 const matrix_index_t dim = folded_dim<matrix_index_t>(frm);
00141 this->m_matrix.resize(dim, dim, false);
00142 this->m_matrix.clear();
00143 for (auto
00144 val_it1 = val.m_matrix.begin();
00145 val_it1 != val.m_matrix.end();
00146 ++val_it1)
00147 for (auto
00148 val_it2 = val_it1.begin();
00149 val_it2 != val_it1.end();
00150 ++val_it2)
00151 this->m_matrix(val_it2.index1(), val_it2.index2()) =
00152 numeric_traits<Scalar_T>::to_scalar_t(*val_it2);
00153 }
00154 }
00155
00156 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00157 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00158 matrix_multi(const multivector_t& val, const index_set_t frm, const bool prechecked)
00159 : m_frame(frm)
00160 {
00161 if (frm != val.m_frame)
00162 *this = multivector_t(framed_multi_t(val), frm);
00163 else
00164 this->m_matrix = val.m_matrix;
00165 }
00166
00167 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00168 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00169 matrix_multi(const index_set_t ist, const Scalar_T& crd)
00170 : m_frame(ist)
00171 {

```

```

00175 const auto dim = folded_dim<matrix_index_t>(this->m_frame);
00176 this->m_matrix.resize(dim, dim, false);
00177 this->m_matrix.clear();
00178 *this += term_t(ist, crd);
00179 }
00180
00182 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00183 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00184 matrix_multi(const index_set_t ist, const Scalar_T& crd, const index_set_t frm, const bool
prechecked)
00185 : m_frame(frm)
00186 {
00187 if (!prechecked && (ist | frm) != frm)
00188 throw error_t("multivector_t(ist,crd,frm): cannot initialize with value outside of frame");
00189 const matrix_index_t dim = folded_dim<matrix_index_t>(frm);
00190 this->m_matrix.resize(dim, dim, false);
00191 this->m_matrix.clear();
00192 *this += term_t(ist, crd);
00193 }
00194
00196 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00197 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00198 matrix_multi(const Scalar_T& scr, const index_set_t frm)
00199 : m_frame(frm)
00200 {
00201 const auto dim = folded_dim<matrix_index_t>(frm);
00202 this->m_matrix.resize(dim, dim, false);
00203 this->m_matrix.clear();
00204 *this += term_t(index_set_t(), scr);
00205 }
00206
00208 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00209 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00210 matrix_multi(const int scr, const index_set_t frm)
00211 { *this = multivector_t(Scalar_T(scr), frm); }
00212
00214 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00215 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00216 matrix_multi(const vector_t& vec,
00217 const index_set_t frm, const bool prechecked)
00218 : m_frame(frm)
00219 {
00220 if (!prechecked && index_t(vec.size()) != frm.count())
00221 throw error_t("multivector_t(vec,frm): cannot initialize with vector not matching frame");
00222 const auto dim = folded_dim<matrix_index_t>(frm);
00223 this->m_matrix.resize(dim, dim, false);
00224 this->m_matrix.clear();
00225 auto idx = frm.min();
00226 const auto frm_end = frm.max()+1;
00227 for (auto& crd : vec)
00228 {
00229 *this += term_t(index_set_t(idx), crd);
00230 for (
00231 ++idx;
00232 idx != frm_end && !frm[idx];
00233 ++idx)
00234 ;
00235 }
00236 }
00237
00239 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00240 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00241 matrix_multi(const std::string& str)
00242 { *this = framed_multi_t(str); }
00243
00245 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00246 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00247 matrix_multi(const std::string& str, const index_set_t frm, const bool prechecked)
00248 { *this = multivector_t(framed_multi_t(str), frm, prechecked); }
00249
00251 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00252 template< typename Other_Scalar_T >
00253 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00254 matrix_multi(const framed_multi_t<Other_Scalar_T,LO,HI,Tune_P>& val)
00255 : m_frame(val.frame())
00256 {
00257 if (val.size() >= Tune_P::fast_size_threshold)
00258 try
00259 {
00260 *this = val.template fast_matrix_multi<Scalar_T>(this->m_frame);
00261 return;
00262 }
00263 catch (const glucat_error& e)
00264 { }
00265 const auto dim = folded_dim<matrix_index_t>(this->m_frame);
00266 this->m_matrix.resize(dim, dim, false);
00267 this->m_matrix.clear();

```

```

00268
00269 using framed_multi_t = framed_multi<Other_Scalar_T,LO,HI,Tune_P>;
00270 for (auto& val_term : val)
00271 *this += val_term;
00272 }
00273
00275 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00276 template< typename Other_Scalar_T >
00277 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00278 matrix_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& framed_val, const index_set_t frm,
const bool prechecked)
00279 {
00280 const auto val = framed_val.truncated();
00281 const auto our_frame = val.frame() | frm;
00282 if (val.size() >= Tune_P::fast_size_threshold)
00283 {
00284 try
00285 {
00286 *this = val.template fast_matrix_multi<Scalar_T>(our_frame);
00287 return;
00288 }
00289 catch (const glucat_error& e)
00290 { }
00291 this->m_frame = our_frame;
00292 const auto dim = folded_dim<matrix_index_t>(our_frame);
00293 this->m_matrix.resize(dim, dim, false);
00294 this->m_matrix.clear();
00295
00296 using framed_multi_t = framed_multi<Other_Scalar_T,LO,HI,Tune_P>;
00297 for (auto& val_term : val)
00298 *this += val_term;
00299 }
00300
00301 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00302 template< typename Matrix_T >
00303 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00304 matrix_multi(const Matrix_T& mtx, const index_set_t frm)
00305 : m_frame(frm), m_matrix(mtx.size1(), mtx.size2())
00306 {
00307 this->m_matrix.clear();
00308
00309 for (auto
00310 mtx_it1 = mtx.begin1();
00311 mtx_it1 != mtx.end1();
00312 ++mtx_it1)
00313 for (auto
00314 mtx_it2 = mtx_it1.begin();
00315 mtx_it2 != mtx_it1.end();
00316 ++mtx_it2)
00317 this->m_matrix(mtx_it2.index1(), mtx_it2.index2()) =
numeric_traits<Scalar_T>::to_scalar_t(*mtx_it2);
00318 }
00319
00321 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00322 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00323 matrix_multi(const matrix_t& mtx, const index_set_t frm)
00324 : m_frame(frm), m_matrix(mtx)
00325 { }
00326
00328 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00329 auto
00330 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00331 operator= (const multivector_t& rhs) -> multivector_t&
00332 {
00333 // Check for assignment to self
00334 if (this == &rhs)
00335 return *this;
00336 this->m_frame = rhs.m_frame;
00337 this->m_matrix = rhs.m_matrix;
00338 return *this;
00339 }
00340
00342 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00343 inline
00344 auto
00345 reframe (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs,
00346 matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs_reframed,
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs_reframed) -> const index_set<LO,HI>
00347 {
00348 using index_set_t = index_set<LO, HI>;
00349 using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
00350 using framed_multi_t = typename multivector_t::framed_multi_t;
00351 // Determine the initial common frame
00352 index_set_t our_frame = lhs.m_frame | rhs.m_frame;
00353 framed_multi_t framed_lhs;
00354 framed_multi_t framed_rhs;
00355 if ((lhs.m_frame != our_frame) || (rhs.m_frame != our_frame))

```

```

00356 {
00357 // The common frame may expand as a result of the transform to framed_multi_t
00358 framed_lhs = framed_multi_t(lhs);
00359 framed_rhs = framed_multi_t(rhs);
00360 our_frame |= framed_lhs.frame() | framed_rhs.frame();
00361 }
00362 // Do the reframing only where necessary
00363 if (lhs.m_frame != our_frame)
00364 lhs_reframed = multivector_t(framed_lhs, our_frame, true);
00365 if (rhs.m_frame != our_frame)
00366 rhs_reframed = multivector_t(framed_rhs, our_frame, true);
00367 return our_frame;
00368 }
00369
00370 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00371 auto
00372 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00373 operator== (const multivector_t& rhs) const -> bool
00374 {
00375 // Ensure that there is no aliasing
00376 if (this == &rhs)
00377 return true;
00378
00379 // Operate only within a common frame
00380 multivector_t lhs_reframed;
00381 multivector_t rhs_reframed;
00382 const index_set_t our_frame = reframe(*this, rhs, lhs_reframed, rhs_reframed);
00383 const multivector_t& lhs_ref = (this->m_frame == our_frame)
00384 ? *this
00385 : lhs_reframed;
00386 const multivector_t& rhs_ref = (rhs.m_frame == our_frame)
00387 ? rhs
00388 : rhs_reframed;
00389
00390 return ublas::norm_inf(lhs_ref.m_matrix - rhs_ref.m_matrix) == 0;
00391 }
00392
00393 // Test for equality of multivector and scalar
00394 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00395 inline
00396 auto
00397 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00398 operator== (const Scalar_T& scr) const -> bool
00399 {
00400 if (scr != Scalar_T(0))
00401 return *this == multivector_t(framed_multi_t(scr), this->m_frame, true);
00402 else if (ublas::norm_inf(this->m_matrix) != 0)
00403 return false;
00404 else
00405 {
00406 const matrix_index_t dim = this->m_matrix.size1();
00407 return !(dim == 1 && this->isnan());
00408 }
00409 }
00410
00411 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00412 inline
00413 auto
00414 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00415 operator+= (const Scalar_T& scr) -> multivector_t&
00416 { return *this += term_t(index_set_t(), scr); }
00417
00418 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00419 inline
00420 auto
00421 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00422 operator+= (const multivector_t& rhs) -> multivector_t&
00423 {
00424 // Ensure that there is no aliasing
00425 if (this == &rhs)
00426 return *this *= Scalar_T(2);
00427
00428 // Operate only within a common frame
00429 multivector_t rhs_reframed;
00430 const index_set_t our_frame = reframe(*this, rhs, *this, rhs_reframed);
00431 const multivector_t& rhs_ref = (rhs.m_frame == our_frame)
00432 ? rhs
00433 : rhs_reframed;
00434
00435 noalias(this->m_matrix) += rhs_ref.m_matrix;
00436 return *this;
00437 }
00438
00439 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00440 inline
00441 auto
00442 matrix_multi<Scalar_T,LO,HI,Tune_P>::

```

```

00447 operator-= (const Scalar_T& scr) -> multivector_t&
00448 { return *this += term_t(index_set_t(), -scr); }
00449
00451 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00452 inline
00453 auto
00454 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00455 operator-= (const multivector_t& rhs) -> multivector_t&
00456 {
00457 // Ensure that there is no aliasing
00458 if (this == &rhs)
00459 return *this = Scalar_T(0);
00460
00461 // Operate only within a common frame
00462 multivector_t rhs_reframed;
00463 const index_set_t our_frame = reframe(*this, rhs, *this, rhs_reframed);
00464 const multivector_t& rhs_ref = (rhs.m_frame == our_frame)
00465 ? rhs
00466 : rhs_reframed;
00467
00468 noalias(this->m_matrix) -= rhs_ref.m_matrix;
00469 return *this;
00470 }
00471
00473 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00474 inline
00475 auto
00476 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00477 operator- () const -> const multivector_t
00478 { return multivector_t(-(this->m_matrix), this->m_frame); }
00479
00481 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00482 inline
00483 auto
00484 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00485 operator*= (const Scalar_T& scr) -> multivector_t&
00486 { // multiply coordinates of all terms by scalar
00487
00488 using traits_t = numeric_traits<Scalar_T>;
00489 if (traits_t::isNan_or_isInf(scr) || this->isnan())
00490 return *this = traits_t::NaN();
00491 if (scr == Scalar_T(0))
00492 *this = Scalar_T(0);
00493 else
00494 this->m_matrix *= scr;
00495 return *this;
00496 }
00497
00499 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00500 inline
00501 auto
00502 operator* (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
00503 {
00504 using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
00505 using index_set_t = typename multivector_t::index_set_t;
00506
00507 if (lhs.isnan() || rhs.isnan())
00508 return numeric_traits<Scalar_T>::NaN();
00509
00510 // Operate only within a common frame
00511 multivector_t lhs_reframed;
00512 multivector_t rhs_reframed;
00513 const index_set_t our_frame = reframe(lhs, rhs, lhs_reframed, rhs_reframed);
00514 const multivector_t& lhs_ref = (lhs.m_frame == our_frame)
00515 ? lhs
00516 : lhs_reframed;
00517 const multivector_t& rhs_ref = (rhs.m_frame == our_frame)
00518 ? rhs
00519 : rhs_reframed;
00520
00521 using matrix_t = typename multivector_t::matrix_t;
00522 using matrix_index_t = typename matrix_t::size_type;
00523
00524 const matrix_index_t dim = lhs_ref.m_matrix.size();
00525 multivector_t result = multivector_t(matrix_t(dim, dim), our_frame);
00526 result.m_matrix.clear();
00527 ublas::axpy_prod(lhs_ref.m_matrix, rhs_ref.m_matrix, result.m_matrix, true);
00528 return result;
00529 }
00530
00532 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00533 inline
00534 auto
00535 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00536 operator*= (const multivector_t& rhs) -> multivector_t&
00537 { return *this = *this * rhs; }

```

```

00538
00540 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00541 inline
00542 auto
00543 operator^ (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
00544 {
00545 using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
00546 using framed_multi_t = typename multivector_t::framed_multi_t;
00547 return framed_multi_t(lhs) ^ framed_multi_t(rhs);
00548 }
00549
00551 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00552 inline
00553 auto
00554 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00555 operator^= (const multivector_t& rhs) -> multivector_t&
00556 { return *this = *this ^ rhs; }
00557
00559 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00560 inline
00561 auto
00562 operator& (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
00563 {
00564 using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
00565 using framed_multi_t = typename multivector_t::framed_multi_t;
00566 return framed_multi_t(lhs) & framed_multi_t(rhs);
00567 }
00568
00570 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00571 inline
00572 auto
00573 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00574 operator&= (const multivector_t& rhs) -> multivector_t&
00575 { return *this = *this & rhs; }
00576
00578 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00579 inline
00580 auto
00581 operator% (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
00582 {
00583 using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
00584 using framed_multi_t = typename multivector_t::framed_multi_t;
00585 return framed_multi_t(lhs) % framed_multi_t(rhs);
00586 }
00587
00589 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00590 inline
00591 auto
00592 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00593 operator%= (const multivector_t& rhs) -> multivector_t&
00594 { return *this = *this % rhs; }
00595
00597 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00598 inline
00599 auto
00600 star(const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs)
-> Scalar_T
00601 { return (lhs * rhs).scalar(); }
00602
00604 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00605 inline
00606 auto
00607 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00608 operator/= (const Scalar_T& scr) -> multivector_t&
00609 { return *this *= Scalar_T(1)/scr; }
00610
00612 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00613 auto
00614 operator/ (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
00615 {
00616 using traits_t = numeric_traits<Scalar_T>;
00617
00618 if (lhs.isnan() || rhs.isnan())
00619 return traits_t::NaN();
00620
00621 if (rhs == Scalar_T(0))
00622 return traits_t::NaN();
00623
00624 using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
00625 using index_set_t = typename multivector_t::index_set_t;
00626
00627 // Operate only within a common frame
00628 multivector_t lhs_reframed;

```



```

00629 multivector_t rhs_reframed;
00630 const auto our_frame = reframe(lhs, rhs, lhs_reframed, rhs_reframed);
00631 const auto& lhs_ref = (lhs.m_frame == our_frame)
00632 ? lhs
00633 : lhs_reframed;
00634 const auto& rhs_ref = (rhs.m_frame == our_frame)
00635 ? rhs
00636 : rhs_reframed;
00637
00638 // Solve result == lhs_ref/rhs_ref <=> result*rhs_ref == lhs_ref
00639 // We now solve X == B/A
00640 // (where X == result, B == lhs_ref.m_matrix and A == rhs_ref.m_matrix)
00641 // X == B/A <=> X*A == B <=> AT*XT == BT
00642 // So, we solve AT*XT == BT
00643
00644 using matrix_t = typename multivector_t::matrix_t;
00645 using matrix_index_t = typename matrix_t::size_type;
00646
00647 const auto& AT = matrix_t(ublas::trans(rhs_ref.m_matrix));
00648 auto LU = AT;
00649
00650 using permutation_t = ublas::permutation_matrix<matrix_index_t>;
00651
00652 auto pvector = permutation_t(AT.size1());
00653 if (! ublas::lu_factorize(LU, pvector))
00654 {
00655 const auto& BT = matrix_t(ublas::trans(lhs_ref.m_matrix));
00656 auto XT = BT;
00657 ublas::lu_substitute(LU, pvector, XT);
00658 if (matrix::isnan(XT))
00659 return traits_t::NaN();
00660
00661 // Iterative refinement.
00662 // Reference: Nicholas J. Higham, "Accuracy and Stability of Numerical Algorithms",
00663 // SIAM, 1996, ISBN 0-89871-355-2, Chapter 11
00664 if (Tune_P::div_max_steps > 0)
00665 {
00666 // matrix_t R = ublas::prod(AT, XT) - BT;
00667 auto R = matrix_t(-BT);
00668 ublas::axpy_prod(AT, XT, R, false);
00669 if (matrix::isnan(R))
00670 return traits_t::NaN();
00671
00672 auto nr = Scalar_T(ublas::norm_inf(R));
00673 if (nr != Scalar_T(0) && !traits_t::isNaN_or_isInf(nr))
00674 {
00675 auto XTnew = XT;
00676 auto nrold = nr + Scalar_T(1);
00677 for (auto
00678 step = 0;
00679 step != Tune_P::div_max_steps &&
00680 nr < nrold &&
00681 nr != Scalar_T(0) &&
00682 nr == nr;
00683 ++step)
00684 {
00685 nrold = nr;
00686 if (step != 0)
00687 XT = XTnew;
00688 auto& D = R;
00689 ublas::lu_substitute(LU, pvector, D);
00690 XTnew -= D;
00691 // noalias(R) = ublas::prod(AT, XTnew) - BT;
00692 R = -BT;
00693 ublas::axpy_prod(AT, XTnew, R, false);
00694 nr = ublas::norm_inf(R);
00695 }
00696 }
00697 }
00698 return multivector_t(ublas::trans(XT), our_frame);
00699 }
00700 else
00701 // AT is singular. Return NaN
00702 return traits_t::NaN();
00703 }
00704
00706 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00707 inline
00708 auto
00709 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00710 operator/=(const multivector_t& rhs) -> multivector_t&
00711 { return *this = *this / rhs; }
00712
00714 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00715 inline
00716 auto
00717 operator| (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const

```

```

matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
00718 { return rhs * lhs / rhs.involute(); }
00719
00721 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00722 inline
00723 auto
00724 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00725 operator|= (const multivector_t& rhs) -> multivector_t&
00726 { return *this = rhs * *this / rhs.involute(); }
00727
00729 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00730 inline
00731 auto
00732 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00733 inv() const -> const multivector_t
00734 { return multivector_t(Scalar_T(1), this->m_frame) / *this; }
00735
00737 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00738 inline
00739 auto
00740 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00741 pow(int m) const -> const multivector_t
00742 { return glucat::pow(*this, m); }
00743
00745 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00746 auto
00747 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00748 outer_pow(int m) const -> const multivector_t
00749 {
00750 if (m < 0)
00751 throw error_t("outer_pow(m): negative exponent");
00752 framed_multi_t a = *this;
00753 return a.outer_pow(m);
00754 }
00755
00757 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00758 inline
00759 auto
00760 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00761 grade() const -> index_t
00762 { return framed_multi_t(*this).grade(); }
00763
00765 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00766 inline
00767 auto
00768 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00769 frame() const -> const index_set_t
00770 { return this->m_frame; }
00771
00773 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00774 inline
00775 auto
00776 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00777 operator[] (const index_set_t ist) const -> Scalar_T
00778 {
00779 // Use matrix inner product only if ist is in frame
00780 if ((ist | this->m_frame) == this->m_frame)
00781 return matrix::inner<Scalar_T>(this->basis_element(ist), this->m_matrix);
00782 else
00783 return Scalar_T(0);
00784 }
00785
00787 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00788 inline
00789 auto
00790 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00791 operator() (index_t grade) const -> const multivector_t
00792 {
00793 if ((grade < 0) || (grade > HI-LO))
00794 return 0;
00795 else
00796 return (framed_multi_t(*this))(grade);
00797 }
00798
00800 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00801 inline
00802 auto
00803 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00804 scalar() const -> Scalar_T
00805 {
00806 const matrix_index_t dim = this->m_matrix.size1();
00807 return matrix::trace(this->m_matrix) / Scalar_T(double(dim));
00808 }
00809
00811 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00812 inline
00813 auto

```

```

00814 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00815 pure() const -> const multivector_t
00816 { return *this - this->scalar(); }
00817
00819 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00820 inline
00821 auto
00822 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00823 even() const -> const multivector_t
00824 { return framed_multi_t(*this).even(); }
00825
00827 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00828 inline
00829 auto
00830 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00831 odd() const -> const multivector_t
00832 { return framed_multi_t(*this).odd(); }
00833
00835 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00836 auto
00837 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00838 vector_part() const -> const vector_t
00839 { return this->vector_part(this->frame(), true); }
00840
00842 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00843 auto
00844 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00845 vector_part(const index_set_t frm, const bool prechecked) const -> const vector_t
00846 {
00847 if (!prechecked && (this->frame() | frm) != frm)
00848 throw error_t("vector_part(frm): value is outside of requested frame");
00849 vector_t result;
00850 // If we need to enlarge the frame we may as well use a framed_multi_t
00851 if (this->frame() != frm)
00852 return framed_multi_t(*this).vector_part(frm, true);
00853
00854 const auto begin_index = frm.min();
00855 const auto end_index = frm.max()+1;
00856 for (auto
00857 idx = begin_index;
00858 idx != end_index;
00859 ++idx)
00860 if (frm[idx])
00861 // Frame may contain indices which do not correspond to a grade 1 term but
00862 // frame cannot omit any index corresponding to a grade 1 term
00863 result.push_back(
00864 matrix::inner<Scalar_T>(this->basis_element(index_set_t(idx)),
00865 this->m_matrix));
00866 return result;
00867 }
00868
00870 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00871 inline
00872 auto
00873 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00874 involute() const -> const multivector_t
00875 { return framed_multi_t(*this).involute(); }
00876
00878 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00879 inline
00880 auto
00881 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00882 reverse() const -> const multivector_t
00883 { return framed_multi_t(*this).reverse(); }
00884
00886 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00887 inline
00888 auto
00889 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00890 conj() const -> const multivector_t
00891 { return framed_multi_t(*this).conj(); }
00892
00894 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00895 inline
00896 auto
00897 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00898 quad() const -> Scalar_T
00899 { // scalar(conj(x)*x) = 2*quad(even(x)) - quad(x)
00900 // Arvind Raja ref: "old clical: quadfunction(p:pterm):pterm in file compmod.pas"
00901 return framed_multi_t(*this).quad();
00902 }
00903
00905 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00906 inline
00907 auto
00908 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00909 norm() const -> Scalar_T

```

```

00910 {
00911 const matrix_index_t dim = this->m_matrix.size1();
00912 return matrix::norm_frob2(this->m_matrix) / Scalar_T(double(dim));
00913 }
00914
00916 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00917 inline
00918 auto
00919 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00920 max_abs() const -> Scalar_T
00921 { return framed_multi_t(*this).max_abs(); }
00922
00924 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00925 auto
00926 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00927 random(const index_set<LO,HI> frm, Scalar_T fill) -> const multivector_t
00928 {
00929 return framed_multi<Scalar_T,LO,HI,Tune_P>::random(frm, fill);
00930 }
00931
00933 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00934 inline
00935 void
00936 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00937 write(const std::string& msg) const
00938 { framed_multi_t(*this).write(msg); }
00939
00941 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00942 inline
00943 void
00944 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00945 write(std::ofstream& ofile, const std::string& msg) const
00946 {
00947 if (!ofile)
00948 throw error_t("write(ofile,msg): cannot write to output file");
00949 framed_multi_t(*this).write(ofile, msg);
00950 }
00951
00953 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00954 inline
00955 auto
00956 operator<< (std::ostream& os, const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::ostream&
00957 {
00958 os << typename matrix_multi<Scalar_T,LO,HI,Tune_P>::framed_multi_t(val);
00959 return os;
00960 }
00961
00963 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00964 inline
00965 auto
00966 operator>> (std::istream& s, matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::istream&
00967 { // Input looks like 1.0-2.0{1,2}+3.2{3,4}
00968 framed_multi<Scalar_T,LO,HI,Tune_P> local;
00969 s >> local;
00970 // If s.bad() then we have a corrupt input
00971 // otherwise we are fine and can copy the resulting matrix_multi
00972 if (!s.bad())
00973 val = local;
00974 return s;
00975 }
00976
00978 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00979 inline
00980 auto
00981 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00982 isinf() const -> bool
00983 {
00984 if (std::numeric_limits<Scalar_T>::has_infinity)
00985 return matrix::isinf(this->m_matrix);
00986 else
00987 return false;
00988 }
00989
00991 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00992 inline
00993 auto
00994 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00995 isnan() const -> bool
00996 {
00997 if (std::numeric_limits<Scalar_T>::has_quiet_NaN)
00998 return matrix::isnan(this->m_matrix);
00999 else
01000 return false;
01001 }
01002
01004 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01005 inline

```

```

01006 auto
01007 matrix_multi<Scalar_T,LO,HI,Tune_P>::
01008 truncated(const Scalar_T& limit) const -> const multivector_t
01009 { return framed_multi_t(*this).truncated(limit); }
01010
01011 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01012 inline
01013 auto
01014 matrix_multi<Scalar_T,LO,HI,Tune_P>::
01015 operator+= (const term_t& term) -> multivector_t&
01016 {
01017 {
01018 if (term.second != Scalar_T(0))
01019 this->m_matrix.plus_assign(matrix_t(this->basis_element(term.first)) * term.second);
01020 return *this;
01021 }
01022
01023 template< typename Multivector_T, typename Matrix_T, typename Basis_Matrix_T >
01024 static
01025 auto
01026 fast(const Matrix_T& X, index_t level) -> Multivector_T
01027 {
01028 {
01029 using framed_multi_t = Multivector_T;
01030
01031 using index_set_t = typename framed_multi_t::index_set_t;
01032 using Scalar_T = typename framed_multi_t::scalar_t;
01033 using matrix_t = Matrix_T;
01034 using basis_matrix_t = Basis_Matrix_T;
01035 using basis_scalar_t = typename basis_matrix_t::value_type;
01036 using traits_t = numeric_traits<Scalar_T>;
01037
01038 if (level == 0)
01039 return framed_multi_t(traits_t::to_scalar_t(X(0,0)));
01040
01041 if (ublas::norm_inf(X) == 0)
01042 return Scalar_T(0);
01043
01044 const basis_matrix_t& I = matrix::unit<basis_matrix_t>(2);
01045 basis_matrix_t J(2,2,2);
01046 J.clear();
01047 J(0,1) = basis_scalar_t(-1);
01048 J(1,0) = basis_scalar_t(1);
01049 basis_matrix_t K = J;
01050 K(0,1) = basis_scalar_t(1);
01051 basis_matrix_t JK = I;
01052 JK(0,0) = basis_scalar_t(-1);
01053
01054 using matrix::signed_perm_nork;
01055 const index_set_t ist_mn = index_set_t(-level);
01056 const index_set_t ist_pn = index_set_t(level);
01057 const index_set_t ist_mnpn = ist_mn | ist_pn;
01058 if (level == 1)
01059 {
01060 using term_t = typename framed_multi_t::term_t;
01061 const Scalar_T i_x = traits_t::to_scalar_t(signed_perm_nork(I, X)(0, 0));
01062 const Scalar_T j_x = traits_t::to_scalar_t(signed_perm_nork(J, X)(0, 0));
01063 const Scalar_T k_x = traits_t::to_scalar_t(signed_perm_nork(K, X)(0, 0));
01064 const Scalar_T jk_x = traits_t::to_scalar_t(signed_perm_nork(JK, X)(0, 0));
01065 framed_multi_t
01066 result = i_x;
01067 result += term_t(ist_mn, j_x); // j_x * mn;
01068 result += term_t(ist_pn, k_x); // k_x * pn;
01069 return result += term_t(ist_mnpn, jk_x); // jk_x * mnpn;
01070 }
01071 else
01072 {
01073 const framed_multi_t& mn = framed_multi_t(ist_mn);
01074 const framed_multi_t& pn = framed_multi_t(ist_pn);
01075 const framed_multi_t& mnpn = framed_multi_t(ist_mnpn);
01076 const framed_multi_t& i_x = fast<framed_multi_t, matrix_t, basis_matrix_t>
01077 (signed_perm_nork(I, X), level-1);
01078 const framed_multi_t& j_x = fast<framed_multi_t, matrix_t, basis_matrix_t>
01079 (signed_perm_nork(J, X), level-1);
01080 const framed_multi_t& k_x = fast<framed_multi_t, matrix_t, basis_matrix_t>
01081 (signed_perm_nork(K, X), level-1);
01082 const framed_multi_t& jk_x = fast<framed_multi_t, matrix_t, basis_matrix_t>
01083 (signed_perm_nork(JK, X), level-1);
01084 framed_multi_t
01085 result = i_x.even() - jk_x.odd();
01086 result += (j_x.even() - k_x.odd()) * mn;
01087 result += (k_x.even() - j_x.odd()) * pn;
01088 return result += (jk_x.even() - i_x.odd()) * mnpn;
01089 }
01090 }
01091
01092 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01093 inline
01094 auto

```

```

01096 matrix_multi<Scalar_T,LO,HI,Tune_P>::
01097 fast_matrix_multi(const index_set_t frm) const -> const multivector_t
01098 {
01099 if (this->m_frame == frm)
01100 return *this;
01101 else
01102 return (this->template fast_framed_multi<Scalar_T>()).template fast_matrix_multi<Scalar_T>(frm);
01103 }
01104
01106 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01107 template <typename Other_Scalar_T>
01108 auto
01109 matrix_multi<Scalar_T,LO,HI,Tune_P>::
01110 fast_framed_multi() const -> const framed_multi<Other_Scalar_T,LO,HI,Tune_P>
01111 {
01112 // Determine the amount of off-centering needed
01113 index_t p = this->m_frame.count_pos();
01114 index_t q = this->m_frame.count_neg();
01115
01116 const index_t bott = pos_mod(p-q, 8);
01117 p += std::max(gen::offset_to_super[bott],index_t(0));
01118 q -= std::min(gen::offset_to_super[bott],index_t(0));
01119
01120 const index_t orig_p = p;
01121 const index_t orig_q = q;
01122 while (p-q > 4)
01123 { p -= 4; q += 4; }
01124 while (p-q < -3)
01125 { p += 4; q -= 4; }
01126 if (p-q > 1)
01127 {
01128 index_t old_p = p;
01129 p = q+1;
01130 q = old_p-1;
01131 }
01132 const index_t level = (p+q)/2;
01133
01134 // Do the inverse fast transform
01135 using framed_multi_t = framed_multi<Other_Scalar_T,LO,HI,Tune_P>;
01136 framed_multi_t val = fast<framed_multi_t, matrix_t, basis_matrix_t>(this->m_matrix, level);
01137
01138 // Off-centre val
01139 switch (pos_mod(orig_p-orig_q, 8))
01140 {
01141 case 2:
01142 case 3:
01143 case 4:
01144 val.centre_qp1_pm1(p, q);
01145 break;
01146 default:
01147 break;
01148 }
01149 if (orig_p-orig_q > 4)
01150 while (p != orig_p)
01151 val.centre_pp4_qm4(p, q);
01152 if (orig_p-orig_q < -3)
01153 while (p != orig_p)
01154 val.centre_pm4_qp4(p, q);
01155
01156 // Return unfolded val
01157 return val.unfold(this->m_frame);
01158 }
01159
01161 template< typename Scalar_T, const index_t LO, const index_t HI, typename Matrix_T >
01162 class basis_table :
01163 public std::map< std::pair< const index_set<LO,HI>, const index_set<LO,HI> >,
01164 Matrix_T* >
01165 {
01166 public:
01167 static auto basis() -> basis_table& { static basis_table b; return b;}
01168 private:
01169 friend class friend_for_private_destructor;
01170 // Enforce singleton
01171 // Reference: A. Alexandrescu, "Modern C++ Design", Chapter 6
01172 basis_table() = default;
01173 ~basis_table() = default;
01174 public:
01175 basis_table(const basis_table&) = delete;
01176 auto operator= (const basis_table&) -> basis_table& = delete;
01177 };
01178
01180 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01181 auto
01182 matrix_multi<Scalar_T,LO,HI,Tune_P>::
01183 basis_element(const index_set_t& ist) const -> const basis_matrix_t
01184 {
01185 using index_set_pair_t = std::pair<const index_set_t, const index_set_t>;

```

```

01190 const auto& unfolded_pair = index_set_pair_t(ist, this->m_frame);
01191
01192 using basis_table_t = basis_table<Scalar_T, LO, HI, basis_matrix_t>;
01193 auto& basis_cache = basis_table_t::basis();
01194
01195 const auto frame_count = this->m_frame.count();
01196 const auto use_cache = frame_count <= index_t(Tune_P::basis_max_count);
01197
01198 if (use_cache)
01199 {
01200 const auto basis_it = basis_cache.find(unfolded_pair);
01201 if (basis_it != basis_cache.end())
01202 return *(basis_it->second);
01203 }
01204 const auto folded_set = ist.fold(this->m_frame);
01205 const auto folded_frame = this->m_frame.fold();
01206 const auto& folded_pair = index_set_pair_t(folded_set, folded_frame);
01207 using basis_pair_t = std::pair<const index_set_pair_t, basis_matrix_t*>;
01208 if (use_cache)
01209 {
01210 const auto basis_it = basis_cache.find(folded_pair);
01211 if (basis_it != basis_cache.end())
01212 {
01213 auto* result_ptr = basis_it->second;
01214 basis_cache.insert(basis_pair_t(unfolded_pair, result_ptr));
01215 return *result_ptr;
01216 }
01217 }
01218 const auto folded_max = folded_frame.max();
01219 const auto folded_min = folded_frame.min();
01220 const auto p = std::max(folded_max, index_t(0));
01221 const auto q = std::max(index_t(-folded_min), index_t(0));
01222 const auto* e = (gen::generator_table<basis_matrix_t>::generator())(p, q);
01223 const auto dim = matrix_index_t(1) << offset_level(p, q);
01224 auto result = matrix::unit<basis_matrix_t>(dim);
01225 for (auto
01226 k = folded_min;
01227 k <= folded_max;
01228 ++k)
01229 if (folded_set[k])
01230 result = matrix::mono_prod(result, e[k]);
01231 if (use_cache)
01232 {
01233 auto* result_ptr = new basis_matrix_t(result);
01234 basis_cache.insert(basis_pair_t(folded_pair, result_ptr));
01235 basis_cache.insert(basis_pair_t(unfolded_pair, result_ptr));
01236 }
01237 return result;
01238 }
01239
01240 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P, const size_t Size
01241 >
01242 inline
01243 static
01244 auto
01245 pade_approx(
01246 const std::array<Scalar_T, Size>& numer,
01247 const std::array<Scalar_T, Size>& denom,
01248 const matrix_multi<Scalar_T, LO, HI, Tune_P>& X) -> const matrix_multi<Scalar_T, LO, HI, Tune_P>
01249 {
01250 // Pade' approximation
01251 // Reference: [GW], Section 4.3, pp318-322
01252 // Reference: [GL], Section 11.3, p572-576.
01253
01254 using multivector_t = matrix_multi<Scalar_T, LO, HI, Tune_P>;
01255 using traits_t = numeric_traits<Scalar_T>;
01256
01257 if (X.isnan())
01258 return traits_t::NaN();
01259
01260 // Array size is assumed to be even
01261 const auto nbr_even_powers = Size/2 - 1;
01262
01263 // Create an array of even powers
01264 auto XX = std::vector<multivector_t>(nbr_even_powers);
01265 XX[0] = X * X;
01266 XX[1] = XX[0] * XX[0];
01267 for (auto
01268 k = size_t(2);
01269 k != nbr_even_powers;
01270 ++k)
01271 XX[k] = XX[k-2] * XX[1];
01272
01273 // Calculate numerator N and denominator D
01274 auto N = multivector_t(numer[1]);
01275 for (auto
01276 k = size_t(0);

```

```

01277 k != nbr_even_powers;
01278 ++k)
01279 N += XX[k] * numer[2*k + 3];
01280 N *= X;
01281 N += numer[0];
01282 for (auto
01283 k = size_t(0);
01284 k != nbr_even_powers;
01285 ++k)
01286 N += XX[k] * numer[2*k + 2];
01287 auto D = multivector_t(denom[1]);
01288 for (auto
01289 k = size_t(0);
01290 k != nbr_even_powers;
01291 ++k)
01292 D += XX[k] * denom[2*k + 3];
01293 D *= X;
01294 D += denom[0];
01295 for (auto
01296 k = size_t(0);
01297 k != nbr_even_powers;
01298 ++k)
01299 D += XX[k] * denom[2*k + 2];
01300 return N / D;
01301 }
01302
01303 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01304 inline
01305 static
01306 void
01307 db_step(matrix_multi<Scalar_T,LO,HI,Tune_P>& M, matrix_multi<Scalar_T,LO,HI,Tune_P>& Y)
01308 {
01309 // Reference: [CHKL]
01310 const auto& invM = inv(M);
01311 M = ((M + invM)/Scalar_T(2) + Scalar_T(1)) / Scalar_T(2);
01312 Y *= (invM + Scalar_T(1)) / Scalar_T(2);
01313 }
01314
01315 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01316 static
01317 auto
01318 db_sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
01319 Scalar_T norm_tol=std::pow(std::numeric_limits<Scalar_T>::epsilon(), 4)) -> const
01320 matrix_multi<Scalar_T,LO,HI,Tune_P>
01321 {
01322 // Reference: [CHKL]
01323 using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
01324 if (val == Scalar_T(0))
01325 return val;
01326 static const auto sqrt_max_steps = Tune_P::db_sqrt_max_steps;
01327 auto M = val;
01328 auto Y = val;
01329 for (auto
01330 step = 0;
01331 step != sqrt_max_steps && norm(M - Scalar_T(1)) > norm_tol;
01332 ++step)
01333 {
01334 if (Y.isnan())
01335 return numeric_traits<Scalar_T>::NaN();
01336 db_step(M, Y);
01337 }
01338 return Y;
01339 }
01340
01341 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01342 static
01343 auto
01344 cr_sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
01345 Scalar_T norm_Y_tol=std::pow(std::numeric_limits<Scalar_T>::epsilon(), 1)) -> const
01346 matrix_multi<Scalar_T,LO,HI,Tune_P>
01347 {
01348 // Reference: [MB]
01349 using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
01350 if (val == Scalar_T(0))
01351 return val;
01352 static const auto sqrt_max_steps = Tune_P::cr_sqrt_max_steps;
01353 auto Z = Scalar_T(2) * (Scalar_T(1) + val);
01354 auto Y = Scalar_T(1) - val;
01355 using traits_t = numeric_traits<Scalar_T>;
01356 auto norm_Y = norm(Y);
01357 for (auto
01358 step = 0;

```



```

01365 step != sqrt_max_steps && norm_Y > norm_Y_tol;
01366 ++step)
01367 {
01368 const auto old_norm_Y = norm_Y;
01369 Y = (-Y / Z) * Y;
01370 norm_Y = norm(Y);
01371 if (Y.isnan() || (norm_Y > old_norm_Y * Scalar_T(2)))
01372 return numeric_traits<Scalar_T>::NaN();
01373
01374 Z += Y * Scalar_T(2);
01375 }
01376 return Z / Scalar_T(4);
01377 }
01378 }
01379
01380 namespace pade {
01381 // Reference: [Z], Padel
01382 template< typename Scalar_T >
01383 struct pade_sqrt_numer
01384 {
01385 {
01386 using array = std::array<Scalar_T, 14>;
01387 static const array number;
01388 };
01389 template< typename Scalar_T >
01390 const typename pade_sqrt_numer<Scalar_T>::array pade_sqrt_numer<Scalar_T>::number =
01391 {
01392 1.0, 27.0/4.0, 81.0/4.0, 2277.0/64.0,
01393 10395.0/256.0, 32319.0/1024.0, 8721.0/512.0, 26163.0/4096.0,
01394 53703.0/32768.0, 36465.0/131072.0, 3861.0/131072.0, 7371.0/4194304.0,
01395 819.0/16777216.0, 27.0/67108864.0
01396 };
01397
01398 // Reference: [Z], Padel
01399 template< typename Scalar_T >
01400 struct pade_sqrt_denom
01401 {
01402 {
01403 using array = std::array<Scalar_T, 14>;
01404 static const array denom;
01405 };
01406 template< typename Scalar_T >
01407 const typename pade_sqrt_denom<Scalar_T>::array pade_sqrt_denom<Scalar_T>::denom =
01408 {
01409 1.0, 25.0/4.0, 69.0/4.0, 1771.0/64.0,
01410 7315.0/256.0, 20349.0/1024.0, 4845.0/512.0, 12597.0/4096.0,
01411 21879.0/32768.0, 12155.0/131072.0, 1001.0/131072.0, 1365.0/4194304.0,
01412 91.0/16777216.0, 1.0/67108864.0
01413 };
01414
01415 template< >
01416 struct pade_sqrt_numer<float>
01417 {
01418 using array = std::array<float, 10>;
01419 static const array number;
01420 };
01421 const typename pade_sqrt_numer<float>::array pade_sqrt_numer<float>::number =
01422 {
01423 1.0, 19.0/4.0, 19.0/2.0, 665.0/64.0,
01424 1729.0/256.0, 2717.0/1024.0, 627.0/1024.0, 627.0/8192.0,
01425 285.0/65536.0, 19.0/262144.0
01426 };
01427 template< >
01428 struct pade_sqrt_denom<float>
01429 {
01430 using array = std::array<float, 10>;
01431 static const array denom;
01432 };
01433 const typename pade_sqrt_denom<float>::array pade_sqrt_denom<float>::denom =
01434 {
01435 1.0, 17.0/4.0, 15.0/2.0, 455.0/64.0,
01436 1001.0/256.0, 1287.0/1024.0, 231.0/1024.0, 165.0/8192.0,
01437 45.0/65536.0, 1.0/262144.0
01438 };
01439
01440 template< >
01441 struct pade_sqrt_numer<long double>
01442 {
01443 using array = std::array<long double, 18>;
01444 static const array number;
01445 };
01446 const typename pade_sqrt_numer<long double>::array pade_sqrt_numer<long double>::number =
01447 {
01448 1.0L, 35.0L/4.0L, 35.0L, 5425.0L/64.0L,
01449 35525.0L/256.0L, 166257.0L/1024.0L, 143325.0L/1024.0L, 740025.0L/8192.0L,
01450 2877875.0L/65536.0L, 4206125.0L/262144.0L, 572033.0L/131072.0L, 1820105.0L/2097152.0L,
01451 1028755.0L/8388608.0L, 395675.0L/33554432.0L, 24225.0L/33554432.0L, 6783.0L/268435456.0L,
01452 1785.0L/4294967296.0L, 35.0L/17179869184.0L
01453 };

```

```

01454 template< >
01455 struct pade_sqrt_denom<long double>
01456 {
01457 using array = std::array<long double, 18>;
01458 static const array denom;
01459 };
01460 const typename pade_sqrt_denom<long double>::array pade_sqrt_denom<long double>::denom =
01461 {
01462 1.0L, 33.0L/4.0L, 31.0L, 4495.0L/64.0L,
01463 27405.0L/256.0L, 118755.0L/1024.0L, 94185.0L/1024.0L, 444015.0L/8192.0L,
01464 1562275.0L/65536.0L, 2042975.0L/262144.0L, 245157.0L/131072.0L, 676039.0L/2097152.0L,
01465 323323.0L/8388608.0L, 101745.0L/33554432.0L, 4845.0L/33554432.0L, 969.0L/268435456.0L,
01466 153.0L/4294967296.0L, 1.0L/17179869184.0L
01467 };
01468
01469 #if defined(_GLUCAT_USE_QD)
01470 template< >
01471 struct pade_sqrt_number<dd_real>
01472 {
01473 using array = std::array<dd_real, 22>;
01474 static const array number;
01475 };
01476 const typename pade_sqrt_number<dd_real>::array pade_sqrt_number<dd_real>::number =
01477 {
01478 dd_real("1"), dd_real("43")/dd_real("4"),
01479 dd_real("215")/dd_real("4"), dd_real("10621")/dd_real("64"),
01480 dd_real("90687")/dd_real("256"), dd_real("567987")/dd_real("1024"),
01481 dd_real("168861")/dd_real("256"), dd_real("1246355")/dd_real("2048"),
01482 dd_real("7228859")/dd_real("16384"), dd_real("16583853")/dd_real("65536"),
01483 dd_real("7538115")/dd_real("65536"), dd_real("173376645")/dd_real("4194304"),
01484 dd_real("195747825")/dd_real("16777216"), dd_real("171655785")/dd_real("67108864"),
01485 dd_real("14375115")/dd_real("33554432"), dd_real("14375115")/dd_real("268435456"),
01486 dd_real("20764055")/dd_real("4294967296"), dd_real("5167525")/dd_real("17179869184"),
01487 dd_real("206701")/dd_real("17179869184"), dd_real("76153")/dd_real("274877906944"),
01488 dd_real("3311")/dd_real("1099511627776"), dd_real("43")/dd_real("4398046511104")
01489 };
01490 template< >
01491 struct pade_sqrt_denom<dd_real>
01492 {
01493 using array = std::array<dd_real, 22>;
01494 static const array denom;
01495 };
01496 const typename pade_sqrt_denom<dd_real>::array pade_sqrt_denom<dd_real>::denom =
01497 {
01498 dd_real("1"), dd_real("41")/dd_real("4"),
01499 dd_real("195")/dd_real("4"), dd_real("9139")/dd_real("64"),
01500 dd_real("73815")/dd_real("256"), dd_real("435897")/dd_real("1024"),
01501 dd_real("121737")/dd_real("256"), dd_real("840565")/dd_real("2048"),
01502 dd_real("4539051")/dd_real("16384"), dd_real("9641775")/dd_real("65536"),
01503 dd_real("4032015")/dd_real("65536"), dd_real("84672315")/dd_real("4194304"),
01504 dd_real("86493225")/dd_real("16777216"), dd_real("67863915")/dd_real("67108864"),
01505 dd_real("5014575")/dd_real("33554432"), dd_real("4345965")/dd_real("268435456"),
01506 dd_real("5311735")/dd_real("4294967296"), dd_real("1081575")/dd_real("17179869184"),
01507 dd_real("33649")/dd_real("17179869184"), dd_real("8855")/dd_real("274877906944"),
01508 dd_real("231")/dd_real("1099511627776"), dd_real("1")/dd_real("4398046511104")
01509 };
01510
01511 template< >
01512 struct pade_sqrt_number<qd_real>
01513 {
01514 using array = std::array<qd_real, 34>;
01515 static const array number;
01516 };
01517 const typename pade_sqrt_number<qd_real>::array pade_sqrt_number<qd_real>::number =
01518 {
01519 qd_real("1"), qd_real("67")/qd_real("4"),
01520 qd_real("134"), qd_real("43617")/qd_real("64"),
01521 qd_real("633485")/qd_real("256"), qd_real("6992857")/qd_real("1024"),
01522 qd_real("15246721")/qd_real("1024"), qd_real("215632197")/qd_real("8192"),
01523 qd_real("2518145487")/qd_real("65536"),
01524 qd_real("12301285425")/qd_real("262144"),
01525 qd_real("6344873535")/qd_real("131072"),
01526 qd_real("89075432355")/qd_real("2097152"),
01527 qd_real("267226297065")/qd_real("8388608"),
01528 qd_real("687479618945")/qd_real("33554432"),
01529 qd_real("379874182975")/qd_real("33554432"),
01530 qd_real("1443521895305")/qd_real("268435456"),
01531 qd_real("9425348845815")/qd_real("4294967296"),
01532 qd_real("13195488384141")/qd_real("17179869184"),
01533 qd_real("987417498133")/qd_real("4294967296"),
01534 qd_real("8055248011085")/qd_real("137438953472"),
01535 qd_real("6958363175533")/qd_real("549755813888"),
01536 qd_real("5056698705201")/qd_real("219902325552"),
01537 qd_real("766166470485")/qd_real("219902325552"),
01538 qd_real("766166470485")/qd_real("17592186044416"),
01539 qd_real("623623871325")/qd_real("140737488355328"),
01540 qd_real("203123203803")/qd_real("562949953421312"),

```

```

01532 qd_real("6478601247")/qd_real("281474976710656"),
qd_real("5038912081")/qd_real("4503599627370496"),
01533 qd_real("719844583")/qd_real("18014398509481984"),
qd_real("71853815")/qd_real("72057594037927936"),
01534 qd_real("1165197")/qd_real("72057594037927936"),
qd_real("87703")/qd_real("576460752303423488"),
01535 qd_real("12529")/qd_real("18446744073709551616"),
qd_real("67")/qd_real("73786976294838206464")
01536 };
01537 template< >
01538 struct pade_sqrt_denom<qd_real>
01539 {
01540 using array = std::array<qd_real, 34>;
01541 static const array denom;
01542 };
01543 const typename pade_sqrt_denom<qd_real>::array pade_sqrt_denom<qd_real>::denom =
01544 {
01545 qd_real("1"),
01546 qd_real("126"),
01547 qd_real("557845")/qd_real("256"),
01548 qd_real("12515965")/qd_real("1024"),
01549 qd_real("1916797311")/qd_real("65536"),
qd_real("8996462475")/qd_real("262144"),
01550 qd_real("4450881435")/qd_real("131072"),
qd_real("59826782925")/qd_real("2097152"),
01551 qd_real("171503444385")/qd_real("8388608"),
qd_real("420696483235")/qd_real("33554432"),
01552 qd_real("221120793075")/qd_real("33554432"),
qd_real("797168807855")/qd_real("268435456"),
01553 qd_real("4923689695575")/qd_real("4294967296"),
qd_real("6499270398159")/qd_real("17179869184"),
01554 qd_real("456864812569")/qd_real("4294967296"),
qd_real("3486599885395")/qd_real("137438953472"),
01555 qd_real("2804116503573")/qd_real("549755813888"),
qd_real("1886827875075")/qd_real("2199023255552"),
01556 qd_real("263012370465")/qd_real("2199023255552"),
qd_real("240141729555")/qd_real("17592186044416"),
01557 qd_real("176848560525")/qd_real("140737488355328"),
qd_real("51538723353")/qd_real("562949953421312"),
01558 qd_real("1450433115")/qd_real("281474976710656"),
qd_real("977699359")/qd_real("4503599627370496"),
01559 qd_real("118183439")/qd_real("18014398509481984"),
qd_real("9652005")/qd_real("72057594037927936"),
01560 qd_real("121737")/qd_real("72057594037927936"),
qd_real("6545")/qd_real("576460752303423488"),
01561 qd_real("561")/qd_real("18446744073709551616"),
qd_real("1")/qd_real("73786976294838206464")
01562 };
01563 #endif
01564 }
01565
01566 namespace glucat
01567 {
01568 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01569 auto
01570 matrix_sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
01571 const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
01572 const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
01573 {
01574 // Reference: [GW], Section 4.3, pp318-322
01575 // Reference: [GL], Section 11.3, p572-576
01576 // Reference: [Z], Padel
01577
01578 using traits_t = numeric_traits<Scalar_T>;
01579
01580 if (val.isnan())
01581 return traits_t::NaN();
01582
01583 const auto scr_val = val.scalar();
01584 if (val == scr_val)
01585 {
01586 if (scr_val < Scalar_T(0))
01587 return i * traits_t::sqrt(-scr_val);
01588 else
01589 return traits_t::sqrt(scr_val);
01590 }
01591 }
01592
01593 // Scale val towards abs(A) == 1 or towards A == 1 as appropriate
01594 const auto scale =
01595 (scr_val != Scalar_T(0) && norm(val/scr_val - Scalar_T(1)) < Scalar_T(1))
01596 ? scr_val
01597 : (scr_val < Scalar_T(0))
01598 ? -abs(val)
01599 : abs(val);
01600 const auto sqrt_scale = traits_t::sqrt(traits_t::abs(scale));
01601 if (traits_t::isNaN_or_isInf(sqrt_scale))
01602 return traits_t::NaN();

```

```

01603
01604 using multivector_t = matrix_multi<Scalar_T, LO, HI, Tune_P>;
01605 auto rescale = multivector_t(sqrt_scale);
01606 if (scale < Scalar_T(0))
01607 rescale = i * sqrt_scale;
01608
01609 const auto& unitval = val / scale;
01610 static const auto max_norm = Scalar_T(1.0/4.0);
01611 auto use_approx_sqrt = true;
01612 auto use_cr_sqrt = false;
01613 auto scaled_result = multivector_t();
01614 #if defined(_GLUCAT_USE_EIGENVALUES)
01615 static const auto sqrt_2 = traits_t::sqrt(Scalar_T(2));
01616 if (level == 0)
01617 {
01618 using matrix_t = typename multivector_t::matrix_t;
01619
01620 // What kind of eigenvalues does the matrix contain?
01621 const auto genus = matrix::classify_eigenvalues(unitval.m_matrix);
01622 const index_t next_level =
01623 (genus.m_is_singular)
01624 ? level
01625 : level + 1;
01626 switch (genus.m_eig_case)
01627 {
01628 case matrix::neg_real_eigs:
01629 scaled_result = matrix_sqrt(-i * unitval, i, next_level) * (i + Scalar_T(1)) / sqrt_2;
01630 use_approx_sqrt = false;
01631 break;
01632 case matrix::both_eigs:
01633 {
01634 const auto safe_arg = genus.m_safe_arg;
01635 scaled_result = matrix_sqrt(exp(i*safe_arg) * unitval, i, next_level) * exp(-i*safe_arg /
Scalar_T(2));
01636 }
01637 use_approx_sqrt = false;
01638 break;
01639 default:
01640 break;
01641 }
01642 use_cr_sqrt = genus.m_is_singular;
01643 }
01644 #endif
01645 if (use_approx_sqrt)
01646 {
01647 scaled_result =
01648 (norm(unitval - Scalar_T(1)) < max_norm)
01649 // Pade' approximation of square root
01650 ? pade_approx(pade::pade_sqrt_numer<Scalar_T>::numer,
01651 pade::pade_sqrt_denom<Scalar_T>::denom,
01652 unitval - Scalar_T(1))
01653 // Product form of Denman-Beavers square root iteration
01654 : (use_cr_sqrt)
01655 ? cr_sqrt(unitval)
01656 : db_sqrt(unitval);
01657 }
01658 return (scaled_result.isnan() ||
01659 !approx_equal(pow(scaled_result, 2), unitval))
01660 ? traits_t::NaN()
01661 : scaled_result * rescale;
01662 }
01663
01664 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01665 auto
01666 sqrt(const matrix_multi<Scalar_T, LO, HI, Tune_P>& val, const matrix_multi<Scalar_T, LO, HI, Tune_P>& i,
bool prechecked) -> const matrix_multi<Scalar_T, LO, HI, Tune_P>
01667 {
01668 {
01669 // Reference: [GW], Section 4.3, pp318-322
01670 // Reference: [GL], Section 11.3, p572-576
01671 // Reference: [Z], Pade1
01672
01673 using traits_t = numeric_traits<Scalar_T>;
01674
01675 if (val.isnan())
01676 return traits_t::NaN();
01677
01678 check_complex(val, i, prechecked);
01679
01680 switch (Tune_P::function_precision)
01681 {
01682 case precision_demoted:
01683 {
01684 using demoted_scalar_t = typename traits_t::demoted::type;
01685 using demoted_multivector_t = matrix_multi<demoted_scalar_t, LO, HI, Tune_P>;
01686
01687 const auto& demoted_val = demoted_multivector_t(val);
01688 const auto& demoted_i = demoted_multivector_t(i);

```

```

01689 return matrix_sqrt(demoted_val, demoted_i, 0);
01690 }
01691 break;
01692 case precision_promoted:
01693 {
01694 using promoted_scalar_t = typename traits_t::promoted::type;
01695 using promoted_multivector_t = matrix_multi<promoted_scalar_t, LO, HI, Tune_P>;
01696
01697 const auto& promoted_val = promoted_multivector_t(val);
01698 const auto& promoted_i = promoted_multivector_t(i);
01699
01700 return matrix_sqrt(promoted_val, promoted_i, 0);
01701 }
01702 break;
01703 default:
01704 return matrix_sqrt(val, i, 0);
01705 }
01706 }
01707 }
01708 }
01709
01710 namespace pade {
01711 // Reference: [Z], Padel
01712 template< typename Scalar_T >
01713 struct pade_log_number
01714 {
01715 using array = std::array<Scalar_T, 14>;
01716 static const array number;
01717 };
01718 template< typename Scalar_T >
01719 const typename pade_log_number<Scalar_T>::array pade_log_number<Scalar_T>::number =
01720 {
01721 {
01722 0.0, 1.0, 6.0, 4741.0/300.0,
01723 1441.0/60.0, 107091.0/4600.0, 8638.0/575.0, 263111.0/40250.0,
01724 153081.0/80500.0, 395243.0/1101240.0, 28549.0/688275.0, 605453.0/228813200.0,
01725 785633.0/10296594000.0, 1145993.0/1873980108000.0
01726 };
01727 };
01728 // Reference: [Z], Padel
01729 template< typename Scalar_T >
01730 struct pade_log_denom
01731 {
01732 using array = std::array<Scalar_T, 14>;
01733 static const array denom;
01734 };
01735 template< typename Scalar_T >
01736 const typename pade_log_denom<Scalar_T>::array pade_log_denom<Scalar_T>::denom =
01737 {
01738 {
01739 1.0, 13.0/2.0, 468.0/25.0, 1573.0/50.0,
01740 1573.0/46.0, 11583.0/460.0, 10296.0/805.0, 2574.0/575.0,
01741 11583.0/10925.0, 143.0/874.0, 572.0/37145.0, 117.0/148580.0,
01742 13.0/742900.0, 1.0/10400600.0
01743 };
01744 };
01745 template< >
01746 struct pade_log_number<float>
01747 {
01748 using array = std::array<float, 10>;
01749 static const array number;
01750 };
01751 const typename pade_log_number<float>::array pade_log_number<float>::number =
01752 {
01753 {
01754 0.0, 1.0, 4.0, 1337.0/204.0,
01755 385.0/68.0, 1879.0/680.0, 193.0/255.0, 197.0/1820.0,
01756 419.0/61880.0, 7129.0/61261200.0
01757 };
01758 };
01759 template< >
01760 struct pade_log_denom<float>
01761 {
01762 using array = std::array<float, 10>;
01763 static const array denom;
01764 };
01765 const typename pade_log_denom<float>::array pade_log_denom<float>::denom =
01766 {
01767 {
01768 1.0, 9.0/2.0, 144.0/17.0, 147.0/17.0,
01769 441.0/85.0, 63.0/34.0, 84.0/221.0, 9.0/221.0,
01770 9.0/4862.0, 1.0/48620.0
01771 };
01772 };
01773 template< >
01774 struct pade_log_number<long double>
01775 {
01776 using array = std::array<long double, 18>;
01777 static const array number;
01778 };
01779 const typename pade_log_number<long double>::array pade_log_number<long double>::number =
01780 {
01781 {

```

```

01778 0.0L, 1.0L, 8.0L,
01779 3835.0L/132.0L, 11363807.0L/122760.0L, 162981.0L/1705.0L,
01780 9036157.0L/125860.0L, 44211925.0L/2718576.0L, 4149566.0L/849555.0L,
01781 16973929.0L/16020180.0L, 116317061.0L/7025382936.0L, 19679783.0L/18441630207.0L,
01782 23763863.0L/614721006900.0L,
01783 50747.0L/79318839600.0L, 42142223.0L/14295951736466400.0L
01783 };
01784 template< >
01785 struct pade_log_denom<long double>
01786 {
01787 using array = std::array<long double, 18>;
01788 static const array denom;
01789 };
01790 const typename pade_log_denom<long double>::array pade_log_denom<long double>::denom =
01791 {
01792 1.0L, 17.0L/2.0L, 1088.0L/33.0L,
01793 850.0L/11.0L, 41650.0L/341.0L, 140777.0L/1023.0L, 1126216.0L/9889.0L,
01794 63206.0L/899.0L, 790075.0L/24273.0L, 60775.0L/5394.0L, 38896.0L/13485.0L,
01795 21658.0L/40455.0L, 21658.0L/310155.0L, 4165.0L/682341.0L, 680.0L/2047023.0L,
01796 34.0L/3411705.0L,
01797 17.0L/129644790.0L, 1.0L/2333606220
01797 };
01798 #if defined(_GLUCAT_USE_QD)
01799 template< >
01800 struct pade_log_number<dd_real>
01801 {
01802 using array = std::array<dd_real, 22>;
01803 static const array number;
01804 };
01805 const typename pade_log_number<dd_real>::array pade_log_number<dd_real>::number =
01806 {
01807 dd_real("0"), dd_real("1"),
01808 dd_real("10"), dd_real("22781")/dd_real("492"),
01809 dd_real("21603")/dd_real("164"), dd_real("5492649")/dd_real("21320"),
01810 dd_real("978724")/dd_real("2665"), dd_real("4191605")/dd_real("10619"),
01811 dd_real("12874933")/dd_real("39442"), dd_real("11473457")/dd_real("54612"),
01812 dd_real("2406734")/dd_real("22755"), dd_real("166770367")/dd_real("4004880"),
01813 dd_real("30653165")/dd_real("2402928"), dd_real("647746389")/dd_real("215195552"),
01814 dd_real("25346331")/dd_real("47074027"), dd_real("278270613")/dd_real("3900419380"),
01815 dd_real("105689791")/dd_real("15601677520"), dd_real("606046475")/dd_real("1379188292768"),
01816 dd_real("969715")/dd_real("53502994116"), dd_real("11098301")/dd_real("26204577562592"),
01817 dd_real("118999")/dd_real("26204577562592"), dd_real("18858053")/dd_real("1392249205900512960")
01818 };
01819 template< >
01820 struct pade_log_denom<dd_real>
01821 {
01822 using array = std::array<dd_real, 22>;
01823 static const array denom;
01824 };
01825 const typename pade_log_denom<dd_real>::array pade_log_denom<dd_real>::denom =
01826 {
01827 dd_real("1"), dd_real("21")/dd_real("2"),
01828 dd_real("2100")/dd_real("41"), dd_real("12635")/dd_real("82"),
01829 dd_real("341145")/dd_real("1066"), dd_real("1037799")/dd_real("2132"),
01830 dd_real("11069856")/dd_real("19721"), dd_real("9883800")/dd_real("19721"),
01831 dd_real("6918660")/dd_real("19721"), dd_real("293930")/dd_real("1517"),
01832 dd_real("1410864")/dd_real("16687"), dd_real("88179")/dd_real("3034"),
01833 dd_real("734825")/dd_real("94054"), dd_real("305235")/dd_real("188108"),
01834 dd_real("348840")/dd_real("1363783"), dd_real("40698")/dd_real("1363783"),
01835 dd_real("6783")/dd_real("2727566"), dd_real("9975")/dd_real("70916716"),
01836 dd_real("266")/dd_real("53187537"), dd_real("7")/dd_real("70916716"),
01837 dd_real("7")/dd_real("8155422340"), dd_real("1")/dd_real("538257874440")
01838 };
01839 template< >
01840 struct pade_log_number<qd_real>
01841 {
01842 using array = std::array<qd_real, 34>;
01843 static const array number;
01844 };
01845 const typename pade_log_number<qd_real>::array pade_log_number<qd_real>::number =
01846 {
01847 qd_real("0"), qd_real("1"),
01848 qd_real("16"),
01849 qd_real("95201")/qd_real("780"),
01850 qd_real("30721")/qd_real("52"),
01851 qd_real("7416257")/qd_real("3640"),
01852 qd_real("1039099")/qd_real("195"),
01853 qd_real("6097772319")/qd_real("555100"),
01854 qd_real("1564058073")/qd_real("85400"),
01855 qd_real("30404640205")/qd_real("1209264"),

```

```

01853 qd_real("725351278")/qd_real("25193"),
01854 qd_real("4092322670789")/qd_real("147429436"),
01855 qd_real("4559713849589")/qd_real("201040140"),
01856 qd_real("5049361751189")/qd_real("320023080"),
01857 qd_real("74979677195")/qd_real("8000577"),
01858 qd_real("16569850691873")/qd_real("3481514244"),
01859 qd_real("1065906022369")/qd_real("515779888"),
01860 qd_real("335956770855841")/qd_real("438412904800"),
01861 qd_real("1462444287585964")/qd_real("6041877844275"),
01862 qd_real("397242326339851")/qd_real("6122436215532"),
01863 qd_real("64211291334131")/qd_real("4373168725380"),
01864 qd_real("142322343550859")/qd_real("51080680851480"),
01865 qd_real("154355972958659")/qd_real("351179680853925"),
01866 qd_real("167483568676259")/qd_real("2937139148960100"),
01867 qd_real("4230788929433")/qd_real("704913395750424"),
01868 qd_real("197968763176019")/qd_real("392923948371995600"),
01869 qd_real("10537522306718")/qd_real("319250708052246425"),
01870 qd_real("236648286272519")/qd_real("144249197475035425500"),
01871 qd_real("260715545088119")/qd_real("4375558990076074573500"),
01872 qd_real("289596255666839")/qd_real("192874640282553367199880"),
01873 qd_real("8802625510547")/qd_real("361639950529787563499775"),
01874 qd_real("373831661521439")/qd_real("1659204093030665341336967700"),
01875 qd_real("446033437968239")/qd_real("464577146048586295574350956000"),
01876 qd_real("53676090078349")/qd_real("47386868896955802148583797512000")
01877 };
01878 template< >
01879 struct pade_log_denom<qd_real>
01880 {
01881 using array = std::array<qd_real, 34>;
01882 static const array denom;
01883 };
01884 const typename pade_log_denom<qd_real>::array pade_log_denom<qd_real>::denom =
01885 {
01886 qd_real("1"),
01887 qd_real("33")/qd_real("2"),
01888 qd_real("8448")/qd_real("65"),
01889 qd_real("42284")/qd_real("65"),
01890 qd_real("211420")/qd_real("91"),
01891 qd_real("573562")/qd_real("91"),
01892 qd_real("32119472")/qd_real("2379"),
01893 qd_real("92917044")/qd_real("3965"),
01894 qd_real("603960786")/qd_real("17995"),
01895 qd_real("144626625")/qd_real("3599"),
01896 qd_real("2776831200")/qd_real("68381"),
01897 qd_real("16692542100")/qd_real("478667"),
01898 qd_real("12241197540")/qd_real("478667"),
01899 qd_real("1098569010")/qd_real("68381"),
01900 qd_real("31387686000")/qd_real("3624193"),
01901 qd_real("9939433900")/qd_real("2479711"),
01902 qd_real("67091178825")/qd_real("42155087"),
01903 qd_real("2683647153")/qd_real("4959422"),
01904 qd_real("19083713088")/qd_real("121505839"),
01905 qd_real("4708152900")/qd_real("121505839"),
01906 qd_real("941630580")/qd_real("116546417"),
01907 qd_real("88704330")/qd_real("62755763"),
01908 qd_real("12902448")/qd_real("62755763"),
01909 qd_real("1542684")/qd_real("62755763"),
01910 qd_real("6427850")/qd_real("2698497809"),
01911 qd_real("3471039")/qd_real("18889484663"),
01912 qd_real("8544096")/qd_real("774468871183"),
01913 qd_real("39556")/qd_real("79027435835"),
01914 qd_real("118668")/qd_real("7191496660985"),
01915 qd_real("10230")/qd_real("27327687311743"),
01916 qd_real("5456")/qd_real("1011124430534491"),
01917 qd_real("44")/qd_real("1011124430534491"),
01918 qd_real("11")/qd_real("70778710137414370"),
01919 qd_real("1")/qd_real("7219428434016265740")
01920 };
01921 #endif
01922 }
01923
01924 namespace glucat{
01925 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01926 static
01927 auto
01928 pade_log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> const
01929 matrix_multi<Scalar_T,LO,HI,Tune_P>
01930 {
01931 // Reference: [GW], Section 4.3, pp318-322
01932 // Reference: [CHKL]
01933 // Reference: [GL], Section 11.3, p572-576
01934 // Reference: [Z], Padel
01935
01936 using traits_t = numeric_traits<Scalar_T>;
01937 if (val == Scalar_T(0) || val.isnan())
01938 return traits_t::NaN();
01939 else

```

```

01911 return pade_approx(pade::pade_log_numer<Scalar_T>::numer,
01912 pade::pade_log_denom<Scalar_T>::denom,
01913 val - Scalar_T(1));
01914 }
01915
01916 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01917 static
01918 auto
01919 cascade_log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> const
01920 matrix_multi<Scalar_T,LO,HI,Tune_P>
01921 {
01922 // Reference: [CHKL]
01923 using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
01924 using traits_t = numeric_traits<Scalar_T>;
01925 if (val == Scalar_T(0) || val.isnan())
01926 return traits_t::NaN();
01927
01928 using limits_t = std::numeric_limits<Scalar_T>;
01929 static const auto epsilon = limits_t::epsilon();
01930 static const auto max_inner_norm = traits_t::pow(epsilon, 2);
01931 static const auto max_outer_norm = Scalar_T(6.0/limits_t::digits);
01932 auto Y = val;
01933 auto E = multivector_t(Scalar_T(0));
01934 Scalar_T norm_Y_1;
01935 auto pow_2_outer_step = Scalar_T(1);
01936 auto pow_4_outer_step = Scalar_T(1);
01937 int outer_step;
01938 for (outer_step = 0, norm_Y_1 = norm(Y - Scalar_T(1));
01939 outer_step != Tune_P::log_max_outer_steps && norm_Y_1 * pow_2_outer_step > max_outer_norm;
01940 ++outer_step, norm_Y_1 = norm(Y - Scalar_T(1)))
01941 {
01942 if (Y == Scalar_T(0) || Y.isnan())
01943 return traits_t::NaN();
01944
01945 // Incomplete product form of Denman-Beavers square root iteration
01946 auto M = Y;
01947 for (auto
01948 inner_step = 0;
01949 inner_step != Tune_P::log_max_inner_steps &&
01950 norm(M - Scalar_T(1)) * pow_4_outer_step > max_inner_norm;
01951 ++inner_step)
01952 db_step(M, Y);
01953
01954 E += (M - Scalar_T(1)) * pow_2_outer_step;
01955 pow_2_outer_step *= Scalar_T(2);
01956 pow_4_outer_step *= Scalar_T(4);
01957 }
01958 if (outer_step == Tune_P::log_max_outer_steps && norm_Y_1 * pow_2_outer_step > max_outer_norm)
01959 return traits_t::NaN();
01960 else
01961 return pade_log(Y) * pow_2_outer_step - E;
01962 }
01963
01964 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01965 auto
01966 matrix_log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
01967 const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
01968 const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
01969 {
01970 // Scaled incomplete square root cascade and scaled Pade' approximation of log
01971 // Reference: [CHKL]
01972
01973 using traits_t = numeric_traits<Scalar_T>;
01974 if (val == Scalar_T(0) || val.isnan())
01975 return traits_t::NaN();
01976
01977 static const auto pi = traits_t::pi();
01978 const auto scr_val = val.scalar();
01979 if (val == scr_val)
01980 {
01981 if (scr_val < Scalar_T(0))
01982 return i * pi + traits_t::log(-scr_val);
01983 else
01984 return traits_t::log(scr_val);
01985 }
01986
01987 // Scale val towards abs(A) == 1 or towards A == 1 as appropriate
01988 const auto max_norm = Scalar_T(1.0/9.0);
01989 const auto scale =
01990 (scr_val != Scalar_T(0) && norm(val/scr_val - Scalar_T(1)) < max_norm)
01991 ? scr_val
01992 : (scr_val < Scalar_T(0))
01993 ? -abs(val)
01994 : abs(val);
01995 if (scale == Scalar_T(0))
01996 return traits_t::NaN();
01997 }
01998

```



```

01999 using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
02000 const auto log_scale = traits_t::log(traits_t::abs(scale));
02001 auto rescale = multivector_t(log_scale);
02002 if (scale < Scalar_T(0))
02003 rescale = i * pi + log_scale;
02004 const auto unitval = val/scale;
02005 if (inv(unitval).isnan())
02006 return traits_t::NaN();
02007
02008 #if defined(_GLUCAT_USE_EIGENVALUES)
02009 auto scaled_result = multivector_t();
02010 if (level == 0)
02011 {
02012 using matrix_t = typename multivector_t::matrix_t;
02013
02014 // What kind of eigenvalues does the matrix contain?
02015 auto genus = matrix::classify_eigenvalues(unitval.m_matrix);
02016 switch (genus.m_eig_case)
02017 {
02018 case matrix::neg_real_eigs:
02019 scaled_result = matrix_log(-i * unitval, i, level + 1) + i * pi/Scalar_T(2);
02020 break;
02021 case matrix::both_eigs:
02022 {
02023 const Scalar_T safe_arg = genus.m_safe_arg;
02024 scaled_result = matrix_log(exp(i*safe_arg) * unitval, i, level + 1) - i * safe_arg;
02025 }
02026 break;
02027 default:
02028 scaled_result = cascade_log(unitval);
02029 break;
02030 }
02031 }
02032 else
02033 scaled_result = cascade_log(unitval);
02034 #else
02035 auto scaled_result = cascade_log(unitval);
02036 #endif
02037 return (scaled_result.isnan())
02038 ? traits_t::NaN()
02039 : scaled_result + rescale;
02040 }
02041
02042 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
02043 auto
02044 log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val, const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
02045 bool prechecked) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
02046 {
02047 using traits_t = numeric_traits<Scalar_T>;
02048
02049 if (val == Scalar_T(0) || val.isnan())
02050 return traits_t::NaN();
02051
02052 check_complex(val, i, prechecked);
02053
02054 switch (Tune_P::function_precision)
02055 {
02056 case precision_demoted:
02057 {
02058 using demoted_scalar_t = typename traits_t::demoted::type;
02059 using demoted_multivector_t = matrix_multi<demoted_scalar_t,LO,HI,Tune_P>;
02060
02061 const auto& demoted_val = demoted_multivector_t(val);
02062 const auto& demoted_i = demoted_multivector_t(i);
02063
02064 return matrix_log(demoted_val, demoted_i, 0);
02065 }
02066 break;
02067 case precision_promoted:
02068 {
02069 using promoted_scalar_t = typename traits_t::promoted::type;
02070 using promoted_multivector_t = matrix_multi<promoted_scalar_t,LO,HI,Tune_P>;
02071
02072 const auto& promoted_val = promoted_multivector_t(val);
02073 const auto& promoted_i = promoted_multivector_t(i);
02074
02075 return matrix_log(promoted_val, promoted_i, 0);
02076 }
02077 break;
02078 default:
02079 return matrix_log(val, i, 0);
02080 }
02081 }
02082
02083 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
02084 auto
02085 exp(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>

```

```

02087 {
02088 using traits_t = numeric_traits<Scalar_T>;
02089 if (val.isnan())
02090 return traits_t::NaN();
02091
02092 const auto scr_val = val.scalar();
02093 if (val == scr_val)
02094 return traits_t::exp(scr_val);
02095
02096 switch (Tune_P::function_precision)
02097 {
02098 case precision_demoted:
02099 {
02100 using demoted_scalar_t = typename traits_t::demoted::type;
02101 using demoted_multivector_t = matrix_multi<demoted_scalar_t, LO, HI, Tune_P>;
02102
02103 const auto& demoted_val = demoted_multivector_t(val);
02104 return clifford_exp(demoted_val);
02105 }
02106 break;
02107 case precision_promoted:
02108 {
02109 using promoted_scalar_t = typename traits_t::promoted::type;
02110 using promoted_multivector_t = matrix_multi<promoted_scalar_t, LO, HI, Tune_P>;
02111
02112 const auto& promoted_val = promoted_multivector_t(val);
02113 return clifford_exp(promoted_val);
02114 }
02115 break;
02116 default:
02117 return clifford_exp(val);
02118 }
02119 }
02120 }
02121 #endif // _GLUCAT_MATRIX_MULTI_IMP_H

```

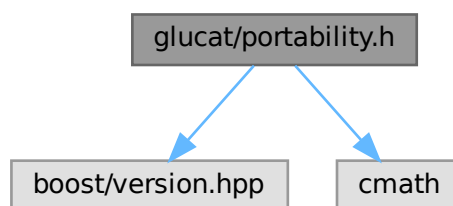
## 7.39 glucat/portability.h File Reference

```

#include <boost/version.hpp>
#include <cmath>

```

Include dependency graph for portability.h:



- `#define _GLUCAT_ISNAN(x)`
- `#define _GLUCAT_ISINF(x)`
- `#define UBLAS_ABS abs`
- `#define UBLAS_SQRT sqrt`

#### 7.39.1.1 \_GLUCAT\_ISINF

**Value:**  
( !\_GLUCAT\_ISNAN (x) && \_GLUCAT\_ISNAN (x-x) )

Referenced by `glucat::numeric_traits< Scalar_T >::isInf()`.

**Value:**  
(x != x)

Referenced by `glucat::numeric_traits< Scalar_T >::isNaN()`.

Definition at line 51 of file portability.h.

### 7.39.1.4 UBLAS\_SQRT

```
#define UBLAS_SQRT sqrt
```

Definition at line 52 of file [portability.h](#).

## 7.40 portability.h

[Go to the documentation of this file.](#)

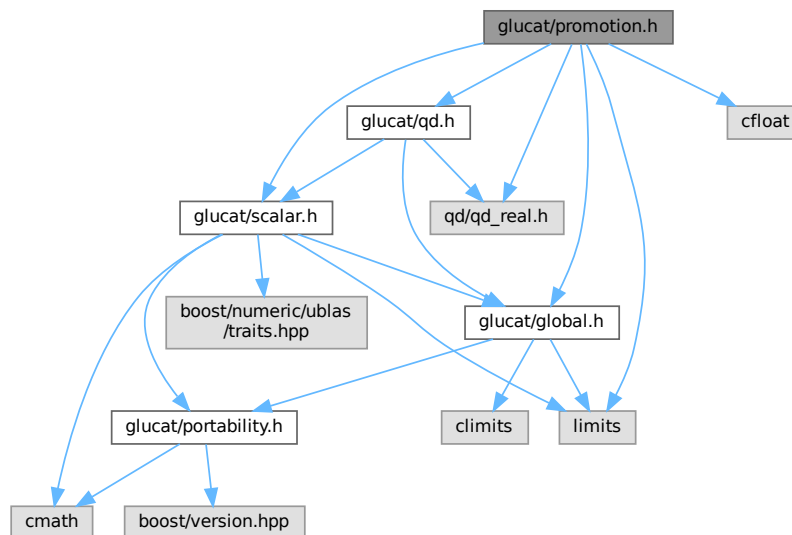
```
00001 #ifndef _GLUCAT_PORTABILITY_H
00002 #define _GLUCAT_PORTABILITY_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 portability.h : Work around non-standard compilers and libraries
00006 -----
00007 begin : Sun 2001-08-18
00008 copyright : (C) 2001-2016 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 #include <boost/version.hpp>
00035 #include <cmath>
00036
00037 // Workaround for isnan and isinf
00038 #if __cplusplus > 199711L
00039 # define _GLUCAT_ISNAN(x) (std::isnan(x))
00040 # define _GLUCAT_ISINF(x) (std::isinf(x))
00041 #else
00042 # define _GLUCAT_ISNAN(x) (x != x)
00043 # define _GLUCAT_ISINF(x) (!_GLUCAT_ISNAN(x) && _GLUCAT_ISNAN(x-x))
00044 #endif
00045
00046 // Workaround for abs and sqrt
00047 #if BOOST_VERSION >= 103400
00048 # define UBLAS_ABS type_abs
00049 # define UBLAS_SQRT type_sqrt
00050 #else
00051 # define UBLAS_ABS abs
00052 # define UBLAS_SQRT sqrt
00053 #endif
00054
00055 // Use with Cygwin gcc to obtain __WORDSIZE
00056 #if defined(HAVE_BITS_WORDSIZE_H)
00057 # include <bits/wordsize.h>
00058 #endif
00059
00060 #endif // _GLUCAT_PORTABILITY_H
```

## 7.41 glucat/promotion.h File Reference

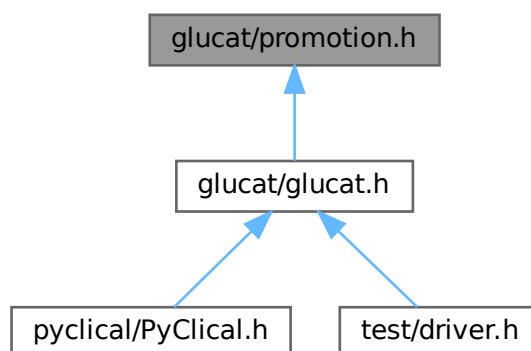
```
#include "glucat/global.h"
#include "glucat/scalar.h"
```

```
#include "glucat/qd.h"
#include <cfloat>
#include <limits>
#include <qd/qd_real.h>
```

Include dependency graph for promotion.h:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [glucat::numeric\\_traits< Scalar\\_T >::promoted](#)  
*Extra traits which extend numeric limits.*
- struct [glucat::numeric\\_traits< Scalar\\_T >::demoted](#)  
*Demoted type for long double.*

## Namespaces

- namespace [glucat](#)

## 7.42 promotion.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_PROMOTION_H
00002 #define _GLUCAT_PROMOTION_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 promotion.h : Define promotion and demotion for specific scalar types
00006 -----
00007 begin : 2021-11-13
00008 copyright : (C) 2021 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
00030 *****/
00031 See also Arvind Raja's original header comments and references in glucat.h
00032 *****/
00033
00034 #include "glucat/global.h"
00035 #include "glucat/scalar.h"
00036 #include "glucat/qd.h"
00037
00038 #include <float>
00039 #include <limits>
00040
00041 #if defined(_GLUCAT_USE_QD)
00042 #include <qd/qd_real.h>
00043 #endif
00044
00045 namespace glucat
00046 {
00047 // Reference: [AA], 2.4, p. 30-31
00048
00049 #if !defined(_GLUCAT_USE_QD) || !defined(QD_API)
00050 #if DBL_MANT_DIG < LDBL_MANT_DIG
00051
00052 template<>
00053 struct
00054 numeric_traits<double>::
00055 promoted {using type = long double;};
00056
00057 template<>
00058 struct
00059 numeric_traits<long double>::
00060 demoted {using type = double;};
00061 #else
00062
00063 template<>
00064 struct
00065 numeric_traits<double>::
00066 promoted {using type = double;};
00067
00068 template<>
00069 struct
00070 numeric_traits<long double>::
00071 demoted {using type = float;};
00072 #endif
00073
00074 }
00075
00076 }
```

```

00079
00080 # endif // DBL_MANT_DIG < LDBL_MANT_DIG
00081
00082 template<>
00083 struct
00084 numeric_traits<long double>::
00085 promoted {using type = long double;};
00086
00087 #else
00088
00089 # if (DBL_MANT_DIG < LDBL_MANT_DIG) && (LDBL_MANT_DIG < DBL_MANT_DIG*2)
00090
00091 template<>
00092 struct
00093 numeric_traits<double>::
00094 promoted {using type = long double;};
00095
00096 template<>
00097 struct
00098 numeric_traits<long double>::
00099 demoted {using type = double;};
00100
00101 template<>
00102 struct
00103 numeric_traits<long double>::
00104 promoted {using type = dd_real;};
00105
00106 template<>
00107 struct
00108 numeric_traits<dd_real>::
00109 demoted {using type = long double;};
00110
00111 template<>
00112 struct
00113 numeric_traits<dd_real>::
00114 promoted {using type = qd_real;};
00115
00116 template<>
00117 struct
00118 numeric_traits<qd_real>::
00119 demoted {using type = dd_real;};
00120
00121 # elif (LDBL_MANT_DIG < DBL_MANT_DIG*2)
00122
00123 template<>
00124 struct
00125 numeric_traits<double>::
00126 promoted {using type = dd_real;};
00127
00128 template<>
00129 struct
00130 numeric_traits<long double>::
00131 demoted {using type = float;};
00132
00133 template<>
00134 struct
00135 numeric_traits<long double>::
00136 promoted {using type = dd_real;};
00137
00138 template<>
00139 struct
00140 numeric_traits<dd_real>::
00141 demoted {using type = double;};
00142
00143 template<>
00144 struct
00145 numeric_traits<dd_real>::
00146 promoted {using type = qd_real;};
00147
00148 template<>
00149 struct
00150 numeric_traits<qd_real>::
00151 demoted {using type = dd_real;};
00152
00153 # else
00154
00155 template<>
00156 struct
00157 numeric_traits<double>::
00158 promoted {using type = dd_real;};
00159
00160 template<>
00161 struct
00162 numeric_traits<dd_real>::
00163 demoted {using type = double;};
00164
00165 template<>
00166 struct
00167 numeric_traits<dd_real>::
00168 demoted {using type = double;};
00169
00170 template<>
00171 struct
00172 numeric_traits<dd_real>::
00173 demoted {using type = double;};
00174
00175 template<>
00176 struct
00177 numeric_traits<dd_real>::
00178 demoted {using type = double;};
00179
00180 template<>

```

```

00182 struct
00183 numeric_traits<dd_real>::
00184 promoted {using type = long double;};
00185
00187 template<>
00188 struct
00189 numeric_traits<long double>::
00190 demoted {using type = dd_real;};
00191
00193 template<>
00194 struct
00195 numeric_traits<long double>::
00196 promoted {using type = qd_real;};
00197
00199 template<>
00200 struct
00201 numeric_traits<qd_real>::
00202 demoted {using type = long double;};
00203
00204 # endif // (DBL_MANT_DIG < LDBL_MANT_DIG) && (LDBL_MANT_DIG < DBL_MANT_DIG*2)
00205
00207 template<>
00208 struct
00209 numeric_traits<qd_real>::
00210 promoted {using type = qd_real;};
00211
00212 #endif // !defined(_GLUCAT_USE_QD) || !defined(QD_API)
00213
00214 } // namespace glucat
00215
00216 #endif // _GLUCAT_PROMOTION_H

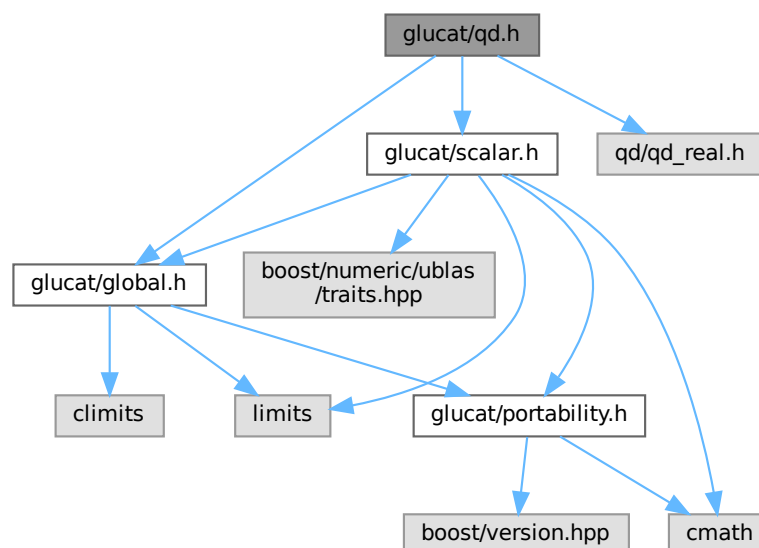
```

## 7.43 glucat/qd.h File Reference

```

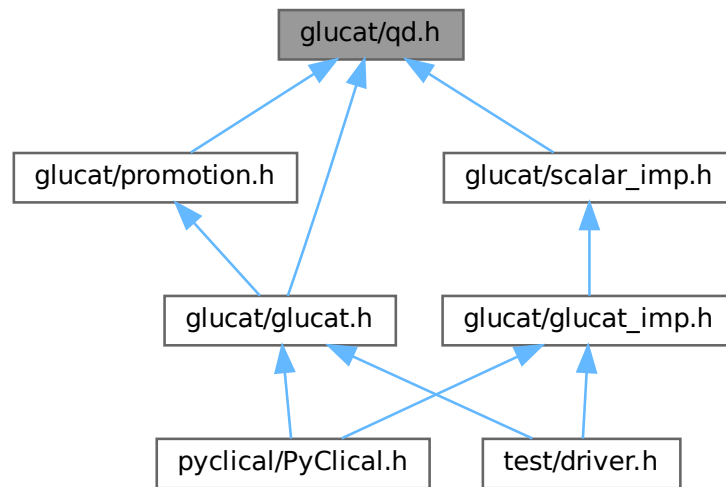
#include "glucat/global.h"
#include "glucat/scalar.h"
#include <qd/qd_real.h>
Include dependency graph for qd.h:

```





This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace `glucat`

## 7.44 qd.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_QD_H
00002 #define _GLUCAT_QD_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 qd.h : Define functions for dd_real and qd_real as scalar_t
00006 -----
00007 begin : 2010-03-23
00008 copyright : (C) 2010-2016 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
00030 *****/
00031 See also Arvind Raja's original header comments and references in glucat.h
00032 *****/
00033

```

```

00034 #include "glucat/global.h"
00035 #include "glucat/scalar.h"
00036
00037 #if defined(_GLUCAT_USE_QD)
00038 # include <qd/qd_real.h>
00039 #endif
00040
00041 namespace glucat
00042 {
00043 // Reference: [AA], 2.4, p. 30-31
00044
00045 #if defined(_GLUCAT_USE_QD) && defined(QD_API)
00046 # define _GLUCAT_QD_F(_T, _F) \
00047 template<> \
00048 inline \
00049 auto \
00050 numeric_traits<_T>:: \
00051 _F(const _T& val) -> _T \
00052 { return ::_F(val); }
00053
00054 template<>
00055 inline
00056 auto
00057 numeric_traits<dd_real>::
00058 isNaN(const dd_real& val) -> bool
00059 { return val.isnan(); }
00060
00061 template<>
00062 inline
00063 auto
00064 numeric_traits<dd_real>::
00065 isInf(const dd_real& val) -> bool
00066 { return val.isinf(); }
00067
00068 template<>
00069 inline
00070 auto
00071 numeric_traits<dd_real>::
00072 isNaN_or_isInf(const dd_real& val) -> bool
00073 { return val.isnan() || val.isinf(); }
00074
00075 template<>
00076 inline
00077 auto
00078 numeric_traits<dd_real>::
00079 to_int(const dd_real& val) -> int
00080 { return ::to_int(val); }
00081
00082 template<>
00083 inline
00084 auto
00085 numeric_traits<dd_real>::
00086 to_double(const dd_real& val) -> double
00087 { return ::to_double(val); }
00088
00089 template<>
00090 inline
00091 auto
00092 numeric_traits<dd_real>::
00093 fmod(const dd_real& lhs, const dd_real& rhs) -> dd_real
00094 { return ::fmod(lhs, rhs); }
00095
00096 template<>
00097 inline
00098 auto
00099 numeric_traits<dd_real>::
00100 pow(const dd_real& val, int n) -> dd_real
00101 {
00102 if (val == dd_real(0))
00103 {
00104 return
00105 (n < 0)
00106 ? NaN()
00107 : (n == 0)
00108 ? dd_real(1)
00109 : dd_real(0);
00110 }
00111 auto result = dd_real(1);
00112 auto power =
00113 (n < 0)
00114 ? dd_real(1)/val
00115 : val;
00116 for (auto
00117 k = std::abs(n);
00118 k != 0;
00119 k /= 2)

```

```

00130 {
00131 if (k % 2)
00132 result *= power;
00133 power *= power;
00134 }
00135 return result;
00136 }
00137
00138 template<>
00139 inline
00140 auto
00141 numeric_traits<dd_real>::
00142 pi() -> dd_real
00143 { return dd_real::_pi; }
00144
00145 template<>
00146 inline
00147 auto
00148 numeric_traits<dd_real>::
00149 ln_2() -> dd_real
00150 { return dd_real::_log2; }
00151
00152 _GLUCAT_QD_F(dd_real, exp)
00153
00154 _GLUCAT_QD_F(dd_real, log)
00155
00156 _GLUCAT_QD_F(dd_real, cos)
00157
00158 _GLUCAT_QD_F(dd_real, acos)
00159
00160 _GLUCAT_QD_F(dd_real, cosh)
00161
00162 _GLUCAT_QD_F(dd_real, sinh)
00163
00164 _GLUCAT_QD_F(dd_real, tan)
00165
00166 _GLUCAT_QD_F(dd_real, atan)
00167
00168 _GLUCAT_QD_F(dd_real, tanh)
00169
00170 template<>
00171 inline
00172 auto
00173 numeric_traits<qd_real>::
00174 isNaN(const qd_real& val) -> bool
00175 { return val.isnan(); }
00176
00177 template<>
00178 inline
00179 auto
00180 numeric_traits<qd_real>::
00181 isInf(const qd_real& val) -> bool
00182 { return val.isinf(); }
00183
00184 template<>
00185 inline
00186 auto
00187 numeric_traits<qd_real>::
00188 isNaN_or_isInf(const qd_real& val) -> bool
00189 { return val.isnan() || val.isinf(); }
00190
00191 template<>
00192 inline
00193 auto
00194 numeric_traits<qd_real>::
00195 to_int(const qd_real& val) -> int
00196 { return ::to_int(val); }
00197
00198 template<>
00199 inline
00200 auto
00201 numeric_traits<qd_real>::

```

```

00224 to_double(const qd_real& val) -> double
00225 { return ::to_double(val); }
00226
00228 template<>
00229 inline
00230 auto
00231 numeric_traits<qd_real>::
00232 fmod(const qd_real& lhs, const qd_real& rhs) -> qd_real
00233 { return ::fmod(lhs, rhs); }
00234
00236 template<>
00237 inline
00238 auto
00239 numeric_traits<qd_real>::
00240 pow(const qd_real& val, int n) -> qd_real
00241 {
00242 if (val == qd_real(0))
00243 {
00244 return
00245 (n < 0)
00246 ? NaN()
00247 : (n == 0)
00248 ? qd_real(1)
00249 : qd_real(0);
00250 }
00251 auto result = qd_real(1);
00252 auto power =
00253 (n < 0)
00254 ? qd_real(1)/val
00255 : val;
00256 for (auto
00257 k = std::abs(n);
00258 k != 0;
00259 k /= 2)
00260 {
00261 if (k % 2)
00262 result *= power;
00263 power *= power;
00264 }
00265 return result;
00266 }
00267
00269 template<>
00270 inline
00271 auto
00272 numeric_traits<qd_real>::
00273 pi() -> qd_real
00274 { return qd_real::_pi; }
00275
00277 template<>
00278 inline
00279 auto
00280 numeric_traits<qd_real>::
00281 ln_2() -> qd_real
00282 { return qd_real::_log2; }
00283
00285 _GLUCAT_QD_F(qd_real, exp)
00286
00287 _GLUCAT_QD_F(qd_real, log)
00288
00289 _GLUCAT_QD_F(qd_real, cos)
00290
00291 _GLUCAT_QD_F(qd_real, acos)
00292
00293 _GLUCAT_QD_F(qd_real, acos)
00294
00295 _GLUCAT_QD_F(qd_real, cosh)
00296
00297 _GLUCAT_QD_F(qd_real, cosh)
00298
00299 _GLUCAT_QD_F(qd_real, sin)
00300
00301 _GLUCAT_QD_F(qd_real, sin)
00302
00303 _GLUCAT_QD_F(qd_real, asin)
00304
00305 _GLUCAT_QD_F(qd_real, asin)
00306
00307 _GLUCAT_QD_F(qd_real, sinh)
00308
00309 _GLUCAT_QD_F(qd_real, tan)
00310
00311 _GLUCAT_QD_F(qd_real, atan)
00312
00313 _GLUCAT_QD_F(qd_real, tanh)
00314
00315 _GLUCAT_QD_F(qd_real, tanh)

```

```

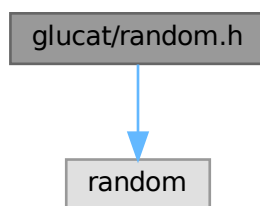
00316
00317 #endif // !defined(_GLUCAT_USE_QD) || !defined(QD_API)
00318
00319 } // namespace glucat
00320
00321 #endif // _GLUCAT_QD_H

```

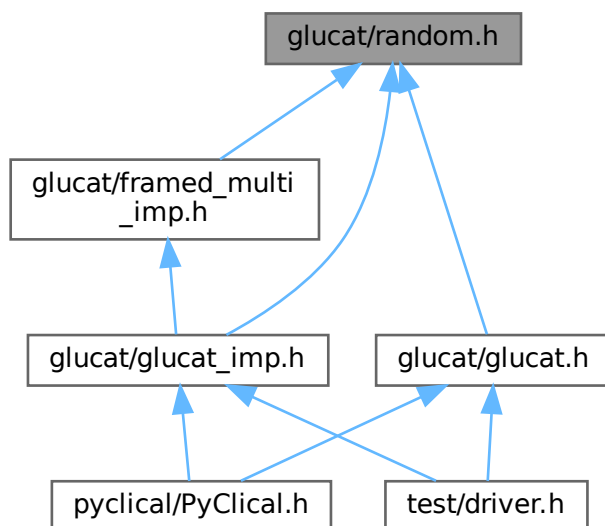
## 7.45 glucat/random.h File Reference

```
#include <random>
```

Include dependency graph for random.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [glucat::random\\_generator< Scalar\\_T >](#)  
Random number generator with single instance per *Scalar\_T*.

## Namespaces

- namespace [glucat](#)

## 7.46 random.h

[Go to the documentation of this file.](#)

```

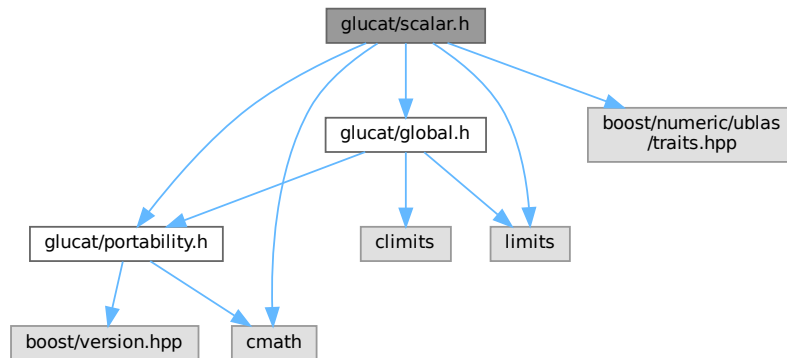
00001 #ifndef _GLUCAT_RANDOM_H
00002 #define _GLUCAT_RANDOM_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 random.h : Random number generator with single instance per Scalar_T
00006 -----
00007 begin : 2010-03-28
00008 copyright : (C) 2001-2012 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
00030 *****/
00031 See also Arvind Raja's original header comments and references in glucat.h
00032 *****/
00033
00034 #include <random>
00035
00036 namespace glucat
00037 {
00038 // Enforce singleton
00039 // Reference: A. Alexandrescu, "Modern C++ Design", Chapter 6
00040 template< typename Scalar_T >
00041 class random_generator
00042 {
00043 private:
00044 friend class friend_for_private_destructor;
00045 public:
00046 static auto generator() -> random_generator& { static random_generator g; return g;}
00047 random_generator(const random_generator&) = delete;
00048 auto operator= (const random_generator&) -> random_generator& = delete;
00049 private:
00050 static const unsigned long seed = 19590921UL;
00051
00052 std::mt19937 uint_gen;
00053 std::uniform_real_distribution<double> uniform_dist;
00054 std::normal_distribution<double> normal_dist;
00055
00056 random_generator() :
00057 uint_gen(), uniform_dist(0.0, 1.0), normal_dist(0.0, 1.0)
00058 { this->uint_gen.seed(seed); }
00059
00060 ~random_generator() = default;
00061
00062 public:
00063 auto uniform() -> Scalar_T
00064 { return Scalar_T(this->uniform_dist(this->uint_gen)); }
00065 auto normal() -> Scalar_T
00066 { return Scalar_T(this->normal_dist(this->uint_gen)); }
00067 };
00068 }
00069
00070 #endif // _GLUCAT_RANDOM_H

```

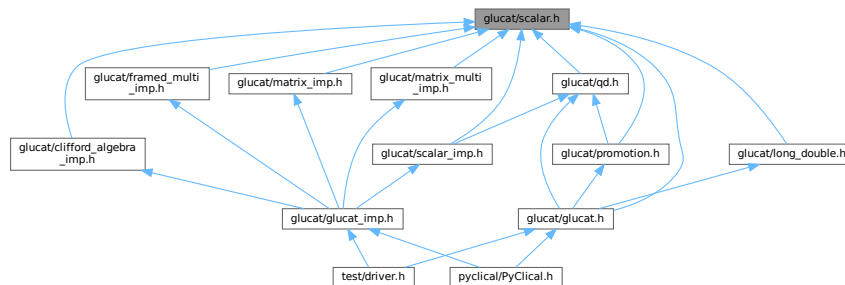
## 7.47 glucat/scalar.h File Reference

```
#include "glucat/portability.h"
#include "glucat/global.h"
#include <boost/numeric/ublas/traits.hpp>
#include <cmath>
#include <limits>
```

Include dependency graph for scalar.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class `glucat::numeric_traits< Scalar_T >`  
*Extra traits which extend numeric limits.*
- struct `glucat::numeric_traits< Scalar_T >::promoted`  
*Extra traits which extend numeric limits.*
- struct `glucat::numeric_traits< Scalar_T >::demoted`  
*Demoted type for long double.*

### Namespaces

- namespace `glucat`

## Functions

- `template<typename Scalar_T>`  
`auto glucat::log2 (const Scalar_T &x) -> Scalar_T`  
*Log base 2 of scalar.*

## 7.48 `scalar.h`

[Go to the documentation of this file.](#)

```
00001 #ifndef _GLUCAT_SCALAR_H
00002 #define _GLUCAT_SCALAR_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 scalar.h : Define functions for scalar_t
00006 -----
00007 begin : 2001-12-20
00008 copyright : (C) 2001-2016 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
00030 *****/
00031 See also Arvind Raja's original header comments and references in glucat.h
00032 *****/
00033
00034 #include "glucat/portability.h"
00035 #include "glucat/global.h"
00036
00037 #include <boost/numeric/ublas/traits.hpp>
00038
00039 #include <cmath>
00040 #include <limits>
00041
00042 namespace glucat
00043 {
00044 // Reference: [AA], 2.4, p. 30-31
00045 template< typename Scalar_T >
00046 class numeric_traits
00047 {
00048 private:
00049 inline
00050 static
00051 auto
00052 isInf(const Scalar_T& val, bool_to_type<false>) -> bool
00053 { return false; }
00054
00055 inline
00056 static
00057 auto
00058 isInf(const Scalar_T& val, bool_to_type<true>) -> bool
00059 { return _GLUCAT_ISINF(val); }
00060
00061 inline
00062 static
00063 auto
00064 isNaN(const Scalar_T& val, bool_to_type<false>) -> bool
00065 { return false; }
00066
00067 inline
00068 static
00069 auto
00070 isNaN(const Scalar_T& val, bool_to_type<true>) -> bool
00071 { return false; }
00072 };
00073
00074 inline
00075 static
00076 auto
00077 isInf(const Scalar_T& val, bool_to_type<true>) -> bool
00078 { return _GLUCAT_ISINF(val); }
00079
00080 inline
00081 static
00082 auto
00083 isNaN(const Scalar_T& val, bool_to_type<true>) -> bool
00084 { return false; }
00085 }
```



```

00076 { return _GLUCAT_ISNAN(val); }
00077
00078 public:
00080 inline
00081 static
00082 auto
00083 isInf(const Scalar_T& val) -> bool
00084 {
00085 return isInf(val,
00086 bool_to_type< std::numeric_limits<Scalar_T>::has_infinity >());
00087 }
00088
00090 inline
00091 static
00092 auto
00093 isNaN(const Scalar_T& val) -> bool
00094 {
00095 return isNaN(val,
00096 bool_to_type< std::numeric_limits<Scalar_T>::has_quiet_NaN >());
00097 }
00098
00100 inline
00101 static
00102 auto
00103 isNaN_or_isInf(const Scalar_T& val) -> bool
00104 {
00105 return isNaN(val,
00106 bool_to_type< std::numeric_limits<Scalar_T>::has_quiet_NaN >())
00107 || isInf(val,
00108 bool_to_type< std::numeric_limits<Scalar_T>::has_infinity >());
00109 }
00110
00112 inline
00113 static
00114 auto
00115 NaN() -> Scalar_T
00116 {
00117 return std::numeric_limits<Scalar_T>::has_quiet_NaN
00118 ? std::numeric_limits<Scalar_T>::quiet_NaN()
00119 : Scalar_T(std::log(0.0));
00120 }
00121
00123 inline
00124 static
00125 auto
00126 to_int(const Scalar_T& val) -> int
00127 { return static_cast<int>(val); }
00128
00130 inline
00131 static
00132 auto
00133 to_double(const Scalar_T& val) -> double
00134 { return static_cast<double>(val); }
00135
00137 template <typename Other_Scalar_T >
00138 inline
00139 static
00140 auto
00141 to_scalar_t(const Other_Scalar_T& val) -> Scalar_T
00142 { return static_cast<Scalar_T>(val); }
00143
00145 struct promoted {using type = double;};
00146
00148 struct demoted {using type = float;};
00149
00151 inline
00152 static
00153 auto
00154 fmod(const Scalar_T& lhs, const Scalar_T& rhs) -> Scalar_T
00155 { return std::fmod(lhs, rhs); }
00156
00158 inline
00159 static
00160 auto
00161 conj(const Scalar_T& val) -> Scalar_T
00162 { return val; }
00163
00165 inline
00166 static
00167 auto
00168 real(const Scalar_T& val) -> Scalar_T
00169 { return val; }
00170
00172 inline
00173 static
00174 auto
00175 imag(const Scalar_T& val) -> Scalar_T

```

```

00176 { return Scalar_T(0); }
00177
00179 inline
00180 static
00181 auto
00182 abs(const Scalar_T& val) -> Scalar_T
00183 { return boost::numeric::ublas::type_traits<Scalar_T>::UBLAS_ABS(val); }
00184
00186 inline
00187 static
00188 auto
00189 pi() -> Scalar_T
00190 { return Scalar_T(3.14159265358979323); }
00191
00193 inline
00194 static
00195 auto
00196 ln_2() -> Scalar_T
00197 { return Scalar_T(0.693147180559945309); }
00198
00200 inline
00201 static
00202 auto
00203 pow(const Scalar_T& val, int n) -> Scalar_T
00204 { return std::pow(val, n); }
00205
00207 inline
00208 static
00209 auto
00210 sqrt(const Scalar_T& val) -> Scalar_T
00211 { return boost::numeric::ublas::type_traits<Scalar_T>::UBLAS_SQRT(val); }
00212
00214 inline
00215 static
00216 auto
00217 exp(const Scalar_T& val) -> Scalar_T
00218 { return std::exp(val); }
00219
00221 inline
00222 static
00223 auto
00224 log(const Scalar_T& val) -> Scalar_T
00225 { return std::log(val); }
00226
00228 inline
00229 static
00230 auto
00231 log2(const Scalar_T& val) -> Scalar_T
00232 { return log(val)/ln_2(); }
00233
00235 inline
00236 static
00237 auto
00238 cos(const Scalar_T& val) -> Scalar_T
00239 { return std::cos(val); }
00240
00242 inline
00243 static
00244 auto
00245 acos(const Scalar_T& val) -> Scalar_T
00246 { return std::acos(val); }
00247
00249 inline
00250 static
00251 auto
00252 cosh(const Scalar_T& val) -> Scalar_T
00253 { return std::cosh(val); }
00254
00256 inline
00257 static
00258 auto
00259 sin(const Scalar_T& val) -> Scalar_T
00260 { return std::sin(val); }
00261
00263 inline
00264 static
00265 auto
00266 asin(const Scalar_T& val) -> Scalar_T
00267 { return std::asin(val); }
00268
00270 inline
00271 static
00272 auto
00273 sinh(const Scalar_T& val) -> Scalar_T
00274 { return std::sinh(val); }
00275
00277 inline

```

```

00278 static
00279 auto
00280 tan(const Scalar_T& val) -> Scalar_T
00281 { return std::tan(val); }
00282
00283
00284 inline
00285 static
00286 auto
00287 atan(const Scalar_T& val) -> Scalar_T
00288 { return std::atan(val); }
00289
00290
00291 inline
00292 static
00293 auto
00294 tanh(const Scalar_T& val) -> Scalar_T
00295 { return std::tanh(val); }
00296
00297 };
00298
00299
00300 template< typename Scalar_T >
00301 inline
00302 auto
00303 log2(const Scalar_T& x) -> Scalar_T
00304 { return numeric_traits<Scalar_T>::log2(x); }
00305 }
00306
00307 #endif // _GLUCAT_SCALAR_H

```

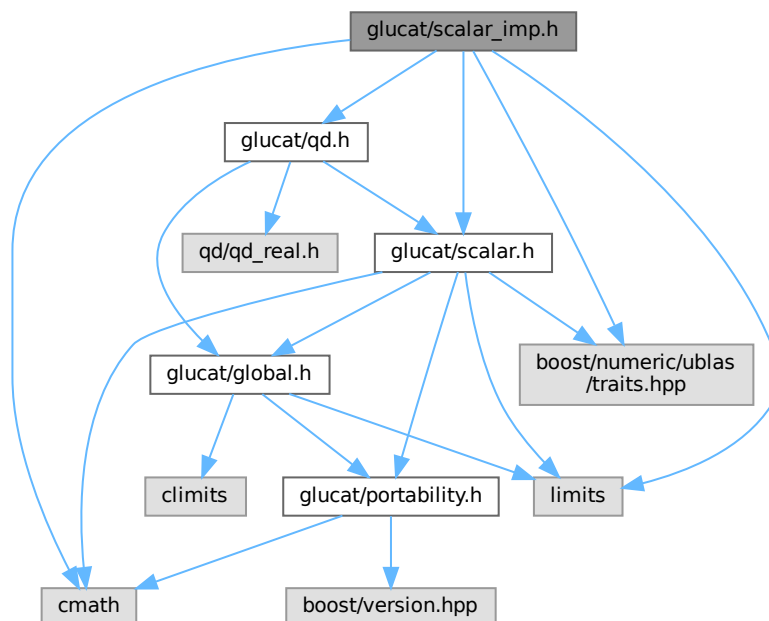
## 7.49 glucat/scalar\_imp.h File Reference

```

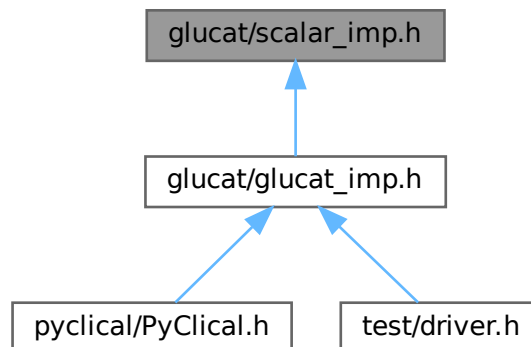
#include "glucat/scalar.h"
#include "glucat/qd.h"
#include <boost/numeric/ublas/traits.hpp>
#include <cmath>
#include <limits>

```

Include dependency graph for scalar\_imp.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace `glucat`

## Functions

- `template<typename Scalar_T >`  
`auto glucat::to\_promote (const Scalar_T &val) -> typename numeric\_traits< Scalar_T >::promoted::type`  
*Cast to promote.*
- `template<typename Scalar_T >`  
`auto glucat::to\_demote (const Scalar_T &val) -> typename numeric\_traits< Scalar_T >::demoted::type`  
*Cast to demote.*

## 7.50 scalar\_imp.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_SCALAR_IMP_H
00002 #define _GLUCAT_SCALAR_IMP_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 scalar_imp.h : Define functions for scalar_t
00006 -----
00007 begin : 2001-12-20
00008 copyright : (C) 2001-2014 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023

```

```

00024 *****
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations, Birkhauser, 1996."
00030 *****
00031 See also Arvind Raja's original header comments and references in glucat.h
00032 *****/
00033
00034 #include "glucat/scalar.h"
00035 #include "glucat/qd.h"
00036
00037 #include <boost/numeric/ublas/traits.hpp>
00038
00039 #include <cmath>
00040 #include <limits>
00041
00042 namespace glucat
00043 {
00044 // Reference: [AA], 2.4, p. 30-31
00045
00046 template< >
00047 template< typename Other_Scalar_T >
00048 inline
00049 auto
00050 numeric_traits<float>::
00051 to_scalar_t(const Other_Scalar_T& val) -> float
00052 { return static_cast<float>(numeric_traits<Other_Scalar_T>::to_double(val)); }
00053
00054 template< >
00055 template< typename Other_Scalar_T >
00056 inline
00057 auto
00058 numeric_traits<double>::
00059 to_scalar_t(const Other_Scalar_T& val) -> double
00060 { return numeric_traits<Other_Scalar_T>::to_double(val); }
00061
00062 #if defined(_GLUCAT_USE_QD)
00063 template< >
00064 template< >
00065 inline
00066 auto
00067 numeric_traits<long double>::
00068 to_scalar_t(const dd_real& val) -> long double
00069 { return static_cast<long double>(val.x[0]) + static_cast<long double>(val.x[1]); }
00070
00071 template< >
00072 template< >
00073 inline
00074 auto
00075 numeric_traits<long double>::
00076 to_scalar_t(const qd_real& val) -> long double
00077 { return static_cast<long double>(val.x[0]) + static_cast<long double>(val.x[1]); }
00078
00079 template< >
00080 template< >
00081 inline
00082 auto
00083 numeric_traits<dd_real>::
00084 to_scalar_t(const long double& val) -> dd_real
00085 { return {double(val), double(val - static_cast<long double>(double(val)))}; }
00086
00087 template< >
00088 template< >
00089 inline
00090 auto
00091 numeric_traits<dd_real>::
00092 to_scalar_t(const qd_real& val) -> dd_real
00093 { return {val.x[0], val.x[1]}; }
00094
00095 template< >
00096 template< >
00097 inline
00098 auto
00099 numeric_traits<qd_real>::
00100 to_scalar_t(const long double& val) -> qd_real
00101 { return {double(val), double(val - static_cast<long double>(double(val))), 0.0, 0.0}; }
00102
00103 template< >
00104 template< >
00105 inline
00106 auto
00107 numeric_traits<qd_real>::
00108 to_scalar_t(const dd_real& val) -> qd_real
00109 { return {val.x[0], val.x[1], 0.0, 0.0}; }
00110
00111 #endif
00112 }

```

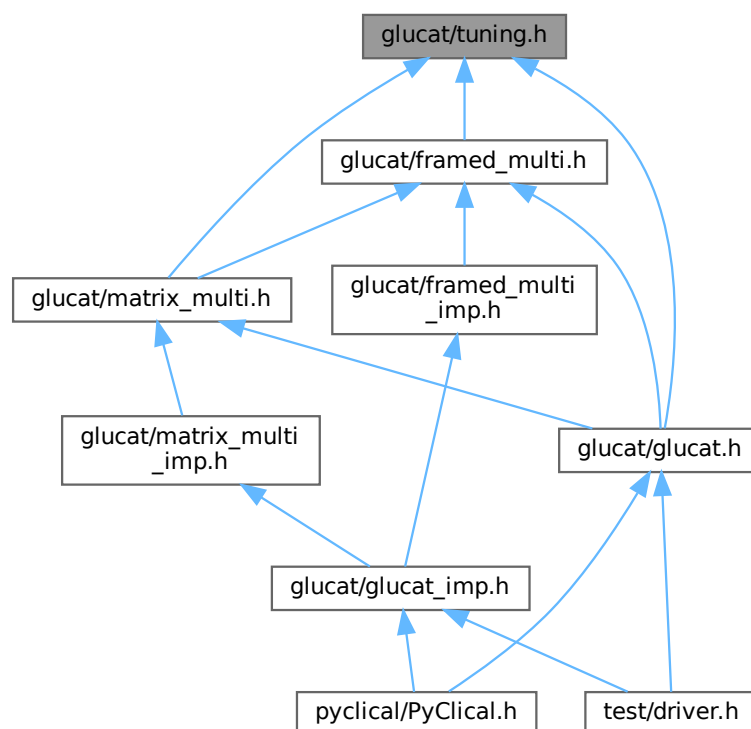
```

00120
00122 template< typename Scalar_T >
00123 inline
00124 auto
00125 to_promote(const Scalar_T& val) -> typename numeric_traits<Scalar_T>::promoted::type
00126 {
00127 using promoted_scalar_t = typename numeric_traits<Scalar_T>::promoted::type;
00128 return numeric_traits<promoted_scalar_t>::to_scalar_t(val);
00129 }
00130
00132 template< typename Scalar_T >
00133 inline
00134 auto
00135 to_demote(const Scalar_T& val) -> typename numeric_traits<Scalar_T>::demoted::type
00136 {
00137 using demoted_scalar_t = typename numeric_traits<Scalar_T>::demoted::type;
00138 return numeric_traits<demoted_scalar_t>::to_scalar_t(val);
00139 }
00140 }
00141
00142 #endif // _GLUCAT_SCALAR_IMP_H

```

## 7.51 glucat/tuning.h File Reference

This graph shows which files directly or indirectly include this file:



### Functions

- `_GLUCAT_CTAssert` (std::numeric\_limits< unsigned int >::radix==2, CannotSetThresholds) namespace glucat

## 7.51.1 Function Documentation

### 7.51.1.1 `_GLUCAT_CTAssert()`

```
_GLUCAT_CTAssert (
 std::numeric_limits< unsigned int >::radix == 2,
 CannotSetThresholds)
```

Base class for policies

Precision policy

Tuning policy

Minimum index count needed to invoke matrix multiplication algorithm

Maximum steps of iterative refinement in division algorithm

Maximum number of steps in cyclic reduction square root iteration

Maximum number of steps in Denman-Beavers square root iteration

Maximum number of incomplete square roots in cascade log algorithm

Maximum number of steps in incomplete square root within cascade log algorithm

Maximum index count of folded frames in basis cache

Minimum map size needed to invoke generalized FFT

Minimum matrix dimension needed to invoke inverse generalized FFT

Minimum size needed for to invoke faster products algorithms

Denominator of proportion of different bits allowed in approximate equality

Extra number of different bits allowed in approximate equality

Precision used for exp, log and sqrt functions

Definition at line 35 of file [tuning.h](#).

## 7.52 tuning.h

[Go to the documentation of this file.](#)

```

00001 #ifndef GLUCAT_TUNING_H
00002 #define GLUCAT_TUNING_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 tuning.h : Policy classes to control tuning
00006
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2021 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 // If radix of int is not 2, we can't easily set thresholds
00035 #GLUCAT_CTAssert(std::numeric_limits<unsigned int>::radix == 2, CannotSetThresholds)
00036
00037 namespace glucat
00038 {
00039 struct policy{};
00040
00041 enum precision_t
00042 {
00043 precision_demoted,
00044 precision_same,
00045 precision_promoted
00046 };
00047
00048 // Tuning policy default constants
00049
00050 const unsigned int Tuning_Default_Mult_Matrix_Threshold = 8;
00051 const unsigned int Tuning_Default_Div_Max_Steps = 4;
00052 const unsigned int Tuning_Default_CR_Sqrt_Max_Steps = 256;
00053 const unsigned int Tuning_Default_DB_Sqrt_Max_Steps = 256;
00054 const unsigned int Tuning_Default_Log_Max_Outer_Steps = 256;
00055 const unsigned int Tuning_Default_Log_Max_Inner_Steps = 32;
00056 const unsigned int Tuning_Default_Basis_Max_Count = 12;
00057 const unsigned int Tuning_Default_Fast_Size_Threshold = 1 << 6;
00058 const unsigned int Tuning_Default_Inv_Fast_Dim_Threshold = 1 << 3;
00059 const unsigned int Tuning_Default_Products_Size_Threshold = 1 << 22;
00060 const unsigned int Tuning_Default_Denom_Different_Bits = 8;
00061 const unsigned int Tuning_Default_Extra_Different_Bits = 8;
00062 const precision_t Tuning_Default_Function_Precision = precision_same;
00063
00064 template
00065 <
00066 unsigned int Mult_Matrix_Threshold = Tuning_Default_Mult_Matrix_Threshold,
00067 unsigned int Div_Max_Steps = Tuning_Default_Div_Max_Steps,
00068 unsigned int CR_Sqrt_Max_Steps = Tuning_Default_CR_Sqrt_Max_Steps,
00069 unsigned int DB_Sqrt_Max_Steps = Tuning_Default_DB_Sqrt_Max_Steps,
00070 unsigned int Log_Max_Outer_Steps = Tuning_Default_Log_Max_Outer_Steps,
00071 unsigned int Log_Max_Inner_Steps = Tuning_Default_Log_Max_Inner_Steps,
00072 unsigned int Basis_Max_Count = Tuning_Default_Basis_Max_Count,
00073 unsigned int Fast_Size_Threshold = Tuning_Default_Fast_Size_Threshold,
00074 unsigned int Inv_Fast_Dim_Threshold = Tuning_Default_Inv_Fast_Dim_Threshold,
00075 unsigned int Products_Size_Threshold = Tuning_Default_Products_Size_Threshold,
00076 unsigned int Denom_Different_Bits = Tuning_Default_Denom_Different_Bits,
00077 unsigned int Extra_Different_Bits = Tuning_Default_Extra_Different_Bits,
00078 precision_t Function_Precision = Tuning_Default_Function_Precision
00079 >
00080 struct tuning : policy
00081 {
00082 using tune_p = tuning
00083 <

```



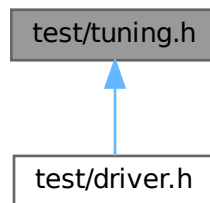
```

00086 Mult_Matrix_Threshold,
00087 Div_Max_Steps,
00088 CR_Sqrt_Max_Steps,
00089 DB_Sqrt_Max_Steps,
00090 Log_Max_Outer_Steps,
00091 Log_Max_Inner_Steps,
00092 Basis_Max_Count,
00093 Fast_Size_Threshold,
00094 Inv_Fast_Dim_Threshold,
00095 Products_Size_Threshold,
00096 Denom_Different_Bits,
00097 Extra_Different_Bits,
00098 Function_Precision
00099 >;
00100 // Tuning for multiplication
00102 enum { mult_matrix_threshold = Mult_Matrix_Threshold };
00103 // Tuning for division
00105 enum { div_max_steps = Div_Max_Steps };
00106 // Tuning for sqrt
00108 enum { cr_sqrt_max_steps = CR_Sqrt_Max_Steps };
00110 enum { db_sqrt_max_steps = DB_Sqrt_Max_Steps };
00111 // Tuning for log
00113 enum { log_max_outer_steps = Log_Max_Outer_Steps };
00115 enum { log_max_inner_steps = Log_Max_Inner_Steps };
00116 // Tuning for basis cache
00118 enum { basis_max_count = Basis_Max_Count };
00119 // Tuning for FFT
00121 enum { fast_size_threshold = Fast_Size_Threshold };
00123 enum { inv_fast_dim_threshold = Inv_Fast_Dim_Threshold };
00124 // Tuning for products (other than geometric product)
00126 enum { products_size_threshold = Products_Size_Threshold };
00127 // Tuning for precision of exp, log and sqrt functions
00129 enum { denom_different_bits = Denom_Different_Bits };
00131 enum { extra_different_bits = Extra_Different_Bits };
00133 static const precision_t function_precision = Function_Precision;
00134 };
00135
00136 using tuning_demoted = tuning
00137 <
00138 Tuning_Default_Mult_Matrix_Threshold,
00139 Tuning_Default_Div_Max_Steps,
00140 Tuning_Default_CR_Sqrt_Max_Steps,
00141 Tuning_Default_DB_Sqrt_Max_Steps,
00142 Tuning_Default_Log_Max_Outer_Steps,
00143 Tuning_Default_Log_Max_Inner_Steps,
00144 Tuning_Default_Basis_Max_Count,
00145 Tuning_Default_Fast_Size_Threshold,
00146 Tuning_Default_Inv_Fast_Dim_Threshold,
00147 Tuning_Default_Products_Size_Threshold,
00148 Tuning_Default_Denom_Different_Bits,
00149 Tuning_Default_Extra_Different_Bits,
00150 precision_demoted
00151 >;
00152
00153 using tuning_promoted = tuning
00154 <
00155 Tuning_Default_Mult_Matrix_Threshold,
00156 Tuning_Default_Div_Max_Steps,
00157 Tuning_Default_CR_Sqrt_Max_Steps,
00158 Tuning_Default_DB_Sqrt_Max_Steps,
00159 Tuning_Default_Log_Max_Outer_Steps,
00160 Tuning_Default_Log_Max_Inner_Steps,
00161 Tuning_Default_Basis_Max_Count,
00162 Tuning_Default_Fast_Size_Threshold,
00163 Tuning_Default_Inv_Fast_Dim_Threshold,
00164 Tuning_Default_Products_Size_Threshold,
00165 Tuning_Default_Denom_Different_Bits,
00166 Tuning_Default_Extra_Different_Bits,
00167 precision_promoted
00168 >;
00169 }
00170
00171 #endif // GLUCAT_TUNING_H

```

## 7.53 test/tuning.h File Reference

This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [glucat](#)

### Typedefs

- using [glucat::tuning\\_slow](#)
- using [glucat::tuning\\_naive](#)
- using [glucat::tuning\\_fast](#)

### Variables

- const unsigned int [glucat::Tuning\\_Int\\_Digits](#) = std::numeric\_limits<int>::digits
- const unsigned int [glucat::Tuning\\_Max\\_Threshold](#) = 1 << [Tuning\\_Int\\_Digits](#)
- const unsigned int [glucat::Tuning\\_Slow\\_Mult\\_Matrix\\_Threshold](#) = [Tuning\\_Max\\_Threshold](#)
- const unsigned int [glucat::Tuning\\_Slow\\_Basis\\_Max\\_Count](#) = 0
- const unsigned int [glucat::Tuning\\_Slow\\_Fast\\_Size\\_Threshold](#) = [Tuning\\_Max\\_Threshold](#)
- const unsigned int [glucat::Tuning\\_Slow\\_Inv\\_Fast\\_Dim\\_Threshold](#) = [Tuning\\_Max\\_Threshold](#)
- const unsigned int [glucat::Tuning\\_Slow\\_Products\\_Size\\_Threshold](#) = [Tuning\\_Max\\_Threshold](#)
- const unsigned int [glucat::Tuning\\_Naive\\_Mult\\_Matrix\\_Threshold](#) = 0
- const unsigned int [glucat::Tuning\\_Naive\\_Basis\\_Max\\_Count](#) = [Tuning\\_Max\\_Threshold](#)
- const unsigned int [glucat::Tuning\\_Naive\\_Fast\\_Size\\_Threshold](#) = [Tuning\\_Max\\_Threshold](#)
- const unsigned int [glucat::Tuning\\_Naive\\_Inv\\_Fast\\_Dim\\_Threshold](#) = [Tuning\\_Max\\_Threshold](#)
- const unsigned int [glucat::Tuning\\_Fast\\_Mult\\_Matrix\\_Threshold](#) = 0
- const unsigned int [glucat::Tuning\\_Fast\\_Div\\_Max\\_Steps](#) = 0
- const unsigned int [glucat::Tuning\\_Fast\\_CR\\_Sqrt\\_Max\\_Steps](#) = 256
- const unsigned int [glucat::Tuning\\_Fast\\_DB\\_Sqrt\\_Max\\_Steps](#) = 256
- const unsigned int [glucat::Tuning\\_Fast\\_Log\\_Max\\_Outer\\_Steps](#) = 16
- const unsigned int [glucat::Tuning\\_Fast\\_Log\\_Max\\_Inner\\_Steps](#) = 8
- const unsigned int [glucat::Tuning\\_Fast\\_Basis\\_Max\\_Count](#) = 1
- const unsigned int [glucat::Tuning\\_Fast\\_Fast\\_Size\\_Threshold](#) = 0
- const unsigned int [glucat::Tuning\\_Fast\\_Inv\\_Fast\\_Dim\\_Threshold](#) = 0
- const unsigned int [glucat::Tuning\\_Fast\\_Products\\_Size\\_Threshold](#) = 0

## 7.54 tuning.h

[Go to the documentation of this file.](#)

```

00001 #ifndef GLUCAT_TEST_TUNING_H
00002 #define GLUCAT_TEST_TUNING_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 tuning.h : Class definitions to control test tuning
00006
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2021 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 namespace glucat
00035 {
00036 const unsigned int Tuning_Int_Digits = std::numeric_limits<int>::digits;
00037 const unsigned int Tuning_Max_Threshold = 1 « Tuning_Int_Digits;
00038
00039 // Specific tuning policy constants and tuning policies
00040
00041 const unsigned int Tuning_Slow_Mult_Matrix_Threshold = Tuning_Max_Threshold;
00042 const unsigned int Tuning_Slow_Basis_Max_Count = 0;
00043 const unsigned int Tuning_Slow_Fast_Size_Threshold = Tuning_Max_Threshold;
00044 const unsigned int Tuning_Slow_Inv_Fast_Dim_Threshold = Tuning_Max_Threshold;
00045 const unsigned int Tuning_Slow_Products_Size_Threshold = Tuning_Max_Threshold;
00046
00047 using tuning_slow = tuning
00048 <
00049 Tuning_Slow_Mult_Matrix_Threshold,
00050 Tuning_Default_Div_Max_Steps,
00051 Tuning_Default_CR_Sqrt_Max_Steps,
00052 Tuning_Default_DB_Sqrt_Max_Steps,
00053 Tuning_Default_Log_Max_Outer_Steps,
00054 Tuning_Default_Log_Max_Inner_Steps,
00055 Tuning_Slow_Basis_Max_Count,
00056 Tuning_Slow_Fast_Size_Threshold,
00057 Tuning_Slow_Inv_Fast_Dim_Threshold,
00058 Tuning_Slow_Products_Size_Threshold,
00059 Tuning_Default_Denom_Different_Bits,
00060 Tuning_Default_Extra_Different_Bits,
00061 Tuning_Default_Function_Precision
00062 >;
00063
00064 const unsigned int Tuning_Naive_Mult_Matrix_Threshold = 0;
00065 const unsigned int Tuning_Naive_Basis_Max_Count = Tuning_Max_Threshold;
00066 const unsigned int Tuning_Naive_Fast_Size_Threshold = Tuning_Max_Threshold;
00067 const unsigned int Tuning_Naive_Inv_Fast_Dim_Threshold = Tuning_Max_Threshold;
00068
00069 using tuning_naive = tuning
00070 <
00071 Tuning_Naive_Mult_Matrix_Threshold,
00072 Tuning_Default_Div_Max_Steps,
00073 Tuning_Default_CR_Sqrt_Max_Steps,
00074 Tuning_Default_DB_Sqrt_Max_Steps,
00075 Tuning_Default_Log_Max_Outer_Steps,
00076 Tuning_Default_Log_Max_Inner_Steps,
00077 Tuning_Naive_Basis_Max_Count,
00078 Tuning_Naive_Fast_Size_Threshold,
00079 Tuning_Naive_Inv_Fast_Dim_Threshold,
00080 Tuning_Default_Products_Size_Threshold,
00081 Tuning_Default_Denom_Different_Bits,
00082 Tuning_Default_Extra_Different_Bits,

```

```

00083 Tuning_Default_Function_Precision
00084 >;
00085
00086 const unsigned int Tuning_Fast_Mult_Matrix_Threshold = 0;
00087 const unsigned int Tuning_Fast_Div_Max_Steps = 0;
00088 const unsigned int Tuning_Fast_CR_Sqrt_Max_Steps = 256;
00089 const unsigned int Tuning_Fast_DB_Sqrt_Max_Steps = 256;
00090 const unsigned int Tuning_Fast_Log_Max_Outer_Steps = 16;
00091 const unsigned int Tuning_Fast_Log_Max_Inner_Steps = 8;
00092 const unsigned int Tuning_Fast_Basis_Max_Count = 1;
00093 const unsigned int Tuning_Fast_Fast_Size_Threshold = 0;
00094 const unsigned int Tuning_Fast_Inv_Fast_Dim_Threshold = 0;
00095 const unsigned int Tuning_Fast_Products_Size_Threshold = 0;
00096
00097 using tuning_fast = tuning
00098 <
00099 Tuning_Fast_Mult_Matrix_Threshold,
00100 Tuning_Fast_Div_Max_Steps,
00101 Tuning_Fast_CR_Sqrt_Max_Steps,
00102 Tuning_Fast_DB_Sqrt_Max_Steps,
00103 Tuning_Fast_Log_Max_Outer_Steps,
00104 Tuning_Fast_Log_Max_Inner_Steps,
00105 Tuning_Fast_Basis_Max_Count,
00106 Tuning_Fast_Fast_Size_Threshold,
00107 Tuning_Fast_Inv_Fast_Dim_Threshold,
00108 Tuning_Fast_Products_Size_Threshold,
00109 Tuning_Default_Denom_Different_Bits,
00110 Tuning_Default_Extra_Different_Bits,
00111 Tuning_Default_Function_Precision
00112 >;
00113 }
00114 #endif // GLUCAT_TEST_TUNING_H

```

## 7.55 pyclical/glucat.pxd File Reference

### Namespaces

- namespace [glucat](#)

## 7.56 glucat.pxd

[Go to the documentation of this file.](#)

```

00001 # -*- coding: utf-8 -*-
00002 # cython: language_level=3
00003 #
00004 # PyClical: Python interface to GluCat:
00005 # Generic library of universal Clifford algebra templates
00006 #
00007 # glucat.pxd: Basic Cython definitions
00008 # corresponding to C++ definitions from PyClical.h.
00009 # Kept as a separate module from PyClical.pxd to avoid namespace clashes.
00010 #
00011 # copyright : (C) 2008-2012 by Paul C. Leopardi
00012 #
00013 # This library is free software: you can redistribute it and/or modify
00014 # it under the terms of the GNU Lesser General Public License as published
00015 # by the Free Software Foundation, either version 3 of the License, or
00016 # (at your option) any later version.
00017 #
00018 # This library is distributed in the hope that it will be useful,
00019 # but WITHOUT ANY WARRANTY; without even the implied warranty of
00020 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00021 # GNU Lesser General Public License for more details.
00022 #
00023 # You should have received a copy of the GNU Lesser General Public License
00024 # along with this library. If not, see <http://www.gnu.org/licenses/>.
00025
00026 from libcpp.vector cimport vector
00027
00028 cdef extern from "PyClical.h":
00029
00030 cdef cppclass String:
00031 char* c_str()
00032

```

```

00033 cdef cppclass IndexSet:
00034 IndexSet ()
00035 IndexSet (IndexSet Ist) except+
00036 IndexSet (int idx) except+
00037 IndexSet (char* str) except+
00038 inline bint operator==(IndexSet Rhs)
00039 inline bint operator!=(IndexSet Rhs)
00040 inline bint operator<(IndexSet Rhs)
00041 inline IndexSet invert "operator~"()
00042 inline bint getitem "operator[]"(int idx)
00043 inline IndexSet set()
00044 inline IndexSet set(int idx) except+
00045 inline IndexSet set(int idx, int val) except+
00046 inline IndexSet reset()
00047 inline IndexSet reset(int idx) except+
00048 int count()
00049 int count_pos()
00050 int count_neg()
00051 int min()
00052 int max()
00053 int sign_of_mult(IndexSet Rhs)
00054 int sign_of_square()
00055 int hash_fn()
00056
00057 int compare(IndexSet Lhs, IndexSet Rhs)
00058 int min_neg(IndexSet Ist)
00059 int max_pos(IndexSet Ist)
00060
00061 ctypedef double scalar_t
00062
00063 cdef cppclass Clifford:
00064 Clifford ()
00065 Clifford (Clifford Clf) except+
00066 Clifford (Clifford Clf, IndexSet ist) except+
00067 Clifford (scalar_t scr) except+
00068 Clifford (char* str) except+
00069 Clifford (IndexSet ist, scalar_t scr) except+
00070 Clifford (vector[scalar_t] vec, IndexSet ist) except+
00071 bint operator==(Clifford Rhs)
00072 bint operator!=(Clifford Rhs)
00073 Clifford neg "operator-"()
00074 scalar_t getitem "operator[]"(IndexSet Ist)
00075 Clifford call "operator()"(int grade)
00076 scalar_t scalar()
00077 Clifford pure()
00078 Clifford even()
00079 Clifford odd()
00080 vector[scalar_t] vector_part()
00081 vector[scalar_t] vector_part(IndexSet frm) except+
00082 Clifford involute()
00083 Clifford reverse()
00084 Clifford conj()
00085 Clifford random(IndexSet Ist, scalar_t fill)
00086 scalar_t norm()
00087 scalar_t quad()
00088 IndexSet frame()
00089 scalar_t max_abs()
00090 Clifford inv()
00091 Clifford pow(int m)
00092 Clifford outer_pow(int m)
00093 Clifford truncated(scalar_t limit)
00094 bint isinf()
00095 bint isnan()
00096 void write(char* msg)
00097
00098 scalar_t error_squared_tol(Clipford Clf)
00099 scalar_t error_squared(Clipford Lhs, Clifford Rhs, scalar_t threshold)
00100 bint approx_equal(Clipford Lhs, Clifford Rhs, scalar_t threshold, scalar_t tol)
00101 scalar_t scalar(Clipford Clf)
00102 scalar_t real(Clipford Clf)
00103 scalar_t imag(Clipford Clf)
00104 Clifford pure(Clipford Clf)
00105 Clifford even(Clipford Clf)
00106 Clifford odd(Clipford Clf)
00107 Clifford involute(Clipford Clf)
00108 Clifford reverse(Clipford Clf)
00109 Clifford conj(Clipford Clf)
00110 scalar_t norm(Clipford Clf)
00111 scalar_t abs(Clipford Clf)
00112 scalar_t max_abs(Clipford Clf)
00113 scalar_t quad(Clipford Clf)
00114 Clifford inv(Clipford Clf)
00115 Clifford pow(Clipford Clf, int m)
00116 Clifford outer_pow(Clipford Clf, int m)
00117
00118 Clifford complexifier(Clipford Clf)
00119 Clifford sqrt(Clipford Clf, Clifford I) except+

```

```

00120 Clifford sqrt(Clifford Clf)
00121 Clifford exp(Clifford Clf)
00122 Clifford log(Clifford Clf, Clifford I) except+
00123 Clifford log(Clifford Clf)
00124 Clifford cos(Clifford Clf, Clifford I) except+
00125 Clifford cos(Clifford Clf)
00126 Clifford acos(Clifford Clf, Clifford I) except+
00127 Clifford acos(Clifford Clf)
00128 Clifford cosh(Clifford Clf)
00129 Clifford acosh(Clifford Clf, Clifford I) except+
00130 Clifford acosh(Clifford Clf)
00131 Clifford sin(Clifford Clf, Clifford I) except+
00132 Clifford sin(Clifford Clf)
00133 Clifford asin(Clifford Clf, Clifford I) except+
00134 Clifford asin(Clifford Clf)
00135 Clifford sinh(Clifford Clf)
00136 Clifford asinh(Clifford Clf, Clifford I) except+
00137 Clifford asinh(Clifford Clf)
00138 Clifford tan(Clifford Clf, Clifford I) except+
00139 Clifford tan(Clifford Clf)
00140 Clifford atan(Clifford Clf, Clifford I) except+
00141 Clifford atan(Clifford Clf)
00142 Clifford tanh(Clifford Clf)
00143 Clifford atanh(Clifford Clf, Clifford I) except+
00144 Clifford atanh(Clifford Clf)
00145
00146 cdef extern from "PyClical.h" namespace "cga3":
00147 Clifford agc3(Clifford Clf)
00148 Clifford cga3(Clifford Clf)
00149 Clifford cga3std(Clifford Clf)

```

## 7.57 pyclical/PyClical.h File Reference

```

#include "glucat/glucat_config.h"
#include "glucat/glucat.h"
#include "glucat/glucat_imp.h"
#include <iostream>
#include <sstream>
#include <iomanip>
#include <limits>

```

Include dependency graph for PyClical.h:



### Namespaces

- namespace `cga3`  
*Definitions for 3D Conformal Geometric Algebra [DL].*

### Typedefs

- using `String` = `std::string`
- using `IndexSet` = `index_set<lo_ndx, hi_ndx>`
- using `scalar_t` = `double`
- using `Clifford` = `matrix_multi<scalar_t, lo_ndx, hi_ndx, tuning_promoted>`

## Functions

- `template<typename Scalar_T >`  
`PyObject * PyFloat_FromDouble (Scalar_T v)`
- `template<typename Index_Set_T >`  
`String index_set_to_repr (const Index_Set_T &ist)`  
*The "official" string representation of Index\_Set\_T ist.*
- `template<typename Index_Set_T >`  
`String index_set_to_str (const Index_Set_T &ist)`  
*The "informal" string representation of Index\_Set\_T ist.*
- `template<typename Multivector_T >`  
`String clifford_to_repr (const Multivector_T &mv)`  
*The "official" string representation of Multivector\_T mv.*
- `template<typename Multivector_T >`  
`String clifford_to_str (const Multivector_T &mv)`  
*The "informal" string representation of Multivector\_T mv.*
- `template<typename Multivector_T >`  
`Multivector_T cga3::cga3 (const Multivector_T &x)`  
*Convert Euclidean 3D vector to Conformal Geometric Algebra null vector [DL (10.50)].*
- `template<typename Multivector_T >`  
`Multivector_T cga3::cga3std (const Multivector_T &X)`  
*Convert CGA3 null vector to standard Conformal Geometric Algebra null vector [DL (10.52)].*
- `template<typename Multivector_T >`  
`Multivector_T cga3::agc3 (const Multivector_T &X)`  
*Convert CGA3 null vector to Euclidean 3D vector [DL (10.50)].*

## Variables

- `String glucat_package_version = GLUCAT_PACKAGE_VERSION`
- `const index_t lo_ndx = DEFAULT_LO`
- `const index_t hi_ndx = DEFAULT_HI`
- `const scalar_t epsilon = std::numeric_limits<scalar_t>::epsilon()`

## 7.57.1 Typedef Documentation

### 7.57.1.1 Clifford

```
using Clifford = matrix_multi<scalar_t, lo_ndx, hi_ndx, tuning_promoted>
```

Definition at line 148 of file [PyClical.h](#).

### 7.57.1.2 IndexSet

```
using IndexSet = index_set<lo_ndx, hi_ndx>
```

Definition at line 145 of file [PyClical.h](#).

### 7.57.1.3 scalar\_t

```
using scalar_t = double
```

Definition at line 147 of file [PyClical.h](#).

### 7.57.1.4 String

```
using String = std::string
```

Definition at line 51 of file [PyClical.h](#).

## 7.57.2 Function Documentation

### 7.57.2.1 clifford\_to\_repr()

```
template<typename Multivector_T >
String clifford_to_repr (
 const Multivector_T & mv) [inline]
```

The “official” string representation of Multivector\_T mv.

Definition at line 75 of file [PyClical.h](#).

Referenced by [PyClical.clifford::\\_\\_repr\\_\\_\(\)](#).

### 7.57.2.2 clifford\_to\_str()

```
template<typename Multivector_T >
String clifford_to_str (
 const Multivector_T & mv) [inline]
```

The “informal” string representation of Multivector\_T mv.

Definition at line 86 of file [PyClical.h](#).

References [glucat::abs\(\)](#).

Referenced by [PyClical.clifford::\\_\\_str\\_\\_\(\)](#).

### 7.57.2.3 index\_set\_to\_repr()

```
template<typename Index_Set_T >
String index_set_to_repr (
 const Index_Set_T & ist) [inline]
```

The “official” string representation of Index\_Set\_T ist.

Definition at line 57 of file [PyClical.h](#).

Referenced by [PyClical.index\\_set::\\_\\_repr\\_\\_\(\)](#).



#### 7.57.2.4 index\_set\_to\_str()

```
template<typename Index_Set_T >
String index_set_to_str (
 const Index_Set_T & ist) [inline]
```

The "informal" string representation of Index\_Set\_T ist.

Definition at line 66 of file [PyClical.h](#).

Referenced by [PyClical.index\\_set::\\_\\_str\\_\\_\(\)](#).

#### 7.57.2.5 PyFloat\_FromDouble()

```
template<typename Scalar_T >
PyObject * PyFloat_FromDouble (
 Scalar_T v) [inline]
```

Create a PyFloatObject object from Scalar\_T v. Needed because Scalar\_T might not be the same as double.

Definition at line 45 of file [PyClical.h](#).

### 7.57.3 Variable Documentation

#### 7.57.3.1 epsilon

```
const scalar_t epsilon = std::numeric_limits<scalar_t>::epsilon()
```

Definition at line 150 of file [PyClical.h](#).

Referenced by [glucat::cascade\\_log\(\)](#), and [glucat::matrix::classify\\_eigenvalues\(\)](#).

#### 7.57.3.2 glucat\_package\_version

```
String glucat_package_version = GLUCAT_PACKAGE_VERSION
```

Definition at line 53 of file [PyClical.h](#).

#### 7.57.3.3 hi\_ndx

```
const index_t hi_ndx = DEFAULT_HI
```

Definition at line 144 of file [PyClical.h](#).

#### 7.57.3.4 lo\_ndx

```
const index_t lo_ndx = DEFAULT_LO
```

Definition at line 143 of file [PyClical.h](#).

## 7.58 PyClical.h

Go to the documentation of this file.

```

00001 /*****
00002 GluCat : Generic library of universal Clifford algebra templates
00003 PyClical.h : C++ definitions needed by PyClical
00004 -----
00005 copyright : (C) 2008-2021 by Paul C. Leopardi
00006 ****
00007
00008 This library is free software: you can redistribute it and/or modify
00009 it under the terms of the GNU Lesser General Public License as published
00010 by the Free Software Foundation, either version 3 of the License, or
00011 (at your option) any later version.
00012
00013 This library is distributed in the hope that it will be useful,
00014 but WITHOUT ANY WARRANTY; without even the implied warranty of
00015 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00016 GNU Lesser General Public License for more details.
00017
00018 You should have received a copy of the GNU Lesser General Public License
00019 along with this library. If not, see <http://www.gnu.org/licenses/>.
00020
00021 ****
00022 This library is based on a prototype written by Arvind Raja and was
00023 licensed under the LGPL with permission of the author. See Arvind Raja,
00024 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00025 in Ablamowicz, Lounesto and Parra (eds.)
00026 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00027 ****
00028 See also Arvind Raja's original header comments in glucat/glucat.h
00029 ****
00030 // References for algorithms:
00031 // [DL]:
00032 // C. Doran and A. Lasenby, "Geometric algebra for physicists", Cambridge, 2003.
00033
00034 #include "glucat/glucat_config.h"
00035 #include "glucat/glucat.h"
00036 #include "glucat/glucat_imp.h"
00037 #include <iostream>
00038 #include <sstream>
00039 #include <iomanip>
00040 #include <limits>
00041
00042 template<typename Scalar_T>
00043 inline PyObject* PyFloat_FromDouble(Scalar_T v)
00044 { return ::PyFloat_FromDouble(glucat::numeric_traits<Scalar_T>::to_double(v)); }
00045
00046 // String representations for use by PyClical Python classes.
00047
00048 using String = std::string;
00049
00050 String glucat_package_version = GLUCAT_PACKAGE_VERSION;
00051
00052 template<typename Index_Set_T>
00053 inline String index_set_to_repr(const Index_Set_T& ist)
00054 {
00055 std::ostringstream os;
00056 os << "index_set(" << ist << ")";
00057 return os.str();
00058 }
00059
00060 template<typename Index_Set_T>
00061 inline String index_set_to_str(const Index_Set_T& ist)
00062 {
00063 std::ostringstream os;
00064 os << ist;
00065 return os.str();
00066 }
00067
00068 template<typename Multivector_T>
00069 inline String clifford_to_repr(const Multivector_T& mv)
00070 {
00071 using scalar_t = typename Multivector_T::scalar_t;
00072 std::ostringstream os;
00073 os << std::setprecision(std::numeric_limits<scalar_t>::digits10 + 1);
00074 os << "clifford(\"" << mv << "\")";
00075 return os.str();
00076 }
00077
00078 template<typename Multivector_T>
00079 inline String clifford_to_str(const Multivector_T& mv)
00080 {
00081 using scalar_t = typename Multivector_T::scalar_t;

```

```

00089 std::ostringstream os;
00090 if (abs(mv) < std::numeric_limits<scalar_t>::epsilon())
00091 os << 0.0;
00092 else
00093 os << std::setprecision(4) << mv.truncated(scalar_t(1.0e-4));
00094 return os.str();
00095 }
00096
00097
00098 namespace cga3
00099 {
00100 template<typename Multivector_T>
00101 inline Multivector_T cga3(const Multivector_T& x)
00102 {
00103 using cl = Multivector_T;
00104 using ist = typename cl::index_set_t;
00105 static const cl ninf3 = cl(ist(4)) + cl(ist(-1));
00106
00107 return (cl(ist(4)) - x) * ninf3 * (x - cl(ist(4)));
00108 }
00109
00110 template<typename Multivector_T>
00111 inline Multivector_T cga3std(const Multivector_T& X)
00112 {
00113 using cl = Multivector_T;
00114 using ist = typename cl::index_set_t;
00115 using scalar_t = typename cl::scalar_t;
00116 static const cl ninf3 = cl(ist(4)) + cl(ist(-1));
00117
00118 return scalar_t(-2.0) * X / (X & ninf3);
00119 }
00120
00121 template<typename Multivector_T>
00122 inline Multivector_T agc3(const Multivector_T& X)
00123 {
00124 using cl = Multivector_T;
00125 using ist = typename cl::index_set_t;
00126 using scalar_t = typename cl::scalar_t;
00127
00128 const cl& cga3stdX = cga3std(X);
00129 return (cl(ist(1))*cga3stdX[ist(1)] +
00130 cl(ist(2))*cga3stdX[ist(2)] +
00131 cl(ist(3))*cga3stdX[ist(3)]) / scalar_t(2.0);
00132 }
00133 }
00134
00135 // Specifications of the IndexSet and Clifford C++ classes for use with PyClical.
00136
00137 using namespace glucat;
00138 const index_t lo_ndx = DEFAULT_LO;
00139 const index_t hi_ndx = DEFAULT_HI;
00140 using IndexSet = index_set<lo_ndx, hi_ndx>;
00141
00142 using scalar_t = double;
00143 using Clifford = matrix_multi<scalar_t, lo_ndx, hi_ndx, tuning_promoted>;
00144
00145 const scalar_t epsilon = std::numeric_limits<scalar_t>::epsilon();
00146
00147 // Do not warn about unused values. This affects clang++ as well as g++.
00148 #pragma GCC diagnostic ignored "-Wunused-value"
00149 #if defined(__clang__)
00150 // Do not warn about unused functions. The affects clang++ only.
00151 #pragma clang diagnostic ignored "-Wunused-function"
00152 // Do not warn about unneeded internal declarations. The affects clang++ only.
00153 #pragma clang diagnostic ignored "-Wunneeded-internal-declaration"
00154 #endif

```

## 7.59 pyclical/PyClical.pxd File Reference

### Namespaces

- namespace [PyClical](#)

## 7.60 PyClicl.pxd

[Go to the documentation of this file.](#)

```

00001 # -*- coding: utf-8 -*-
00002 # cython: language_level=3
00003 #
00004 # PyClicl: Python interface to GluCat:
00005 # Generic library of universal Clifford algebra templates
00006 #
00007 # PyClicl.pxd: Basic Cython definitions for PyClicl
00008 # corresponding to C++ definitions from PyClicl.h.
00009 #
00010 # copyright : (C) 2008-2021 by Paul C. Leopardi
00011 #
00012 # This library is free software: you can redistribute it and/or modify
00013 # it under the terms of the GNU Lesser General Public License as published
00014 # by the Free Software Foundation, either version 3 of the license, or
00015 # (at your option) any later version.
00016 #
00017 # This library is distributed in the hope that it will be useful,
00018 # but WITHOUT ANY WARRANTY; without even the implied warranty of
00019 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00020 # GNU Lesser General Public License for more details.
00021 #
00022 # You should have received a copy of the GNU Lesser General Public License
00023 # along with this library. If not, see <http://www.gnu.org/licenses/>.
00024 #
00025 cimport glucat
00026 from glucat cimport IndexSet, String, Clifford, scalar_t, vector
00027 from libcpp.string cimport string
00028
00029 cdef extern from "PyClicl.h":
00030 string glucat_package_version
00031
00032 IndexSet operator&(IndexSet Lhs, IndexSet Rhs)
00033 IndexSet operator|(IndexSet Lhs, IndexSet Rhs)
00034 IndexSet operator^(IndexSet Lhs, IndexSet Rhs)
00035
00036 string index_set_to_repr(IndexSet& Ist)
00037 string index_set_to_str(IndexSet& Ist)
00038
00039 Clifford operator+(Clifford Lhs, Clifford Rhs)
00040 Clifford operator-(Clifford Lhs, Clifford Rhs)
00041 Clifford operator*(Clifford Lhs, Clifford Rhs)
00042 Clifford operator&(Clifford Lhs, Clifford Rhs)
00043 Clifford operator%(Clifford Lhs, Clifford Rhs)
00044 Clifford operator^(Clifford Lhs, Clifford Rhs)
00045 Clifford operator/(Clifford Lhs, Clifford Rhs)
00046 Clifford operator|(Clifford Lhs, Clifford Rhs)
00047
00048 string clifford_to_repr(Clifford& Clf)
00049 string clifford_to_str(Clifford& Clf)
00050
00051 const scalar_t epsilon

```

## 7.61 pyclicl/PyClicl.pyx File Reference

### Classes

- class [PyClicl.index\\_set](#)
- class [PyClicl.clifford](#)

### Namespaces

- namespace [PyClicl](#)

### Functions

- [PyClicl.index\\_set\\_hidden\\_doctests](#) ()
- [PyClicl.clifford\\_hidden\\_doctests](#) ()
- [PyClicl.e](#) (obj)
- [PyClicl.istpq](#) (p, q)
- [PyClicl.\\_test](#) ()

## Variables

- `PyClical.__version__` = `str(glucat_package_version,'utf-8')`
- `PyClical.lhs`
- `PyClical.rhs`
- `PyClical.threshold` = `error_squared_tol(rhs)` if threshold is `None` else threshold
- `PyClical.None`
- `PyClical.tol` = `error_squared_tol(rhs)` if tol is `None` else tol
- `PyClical.obj`
- `PyClical.i`
- `PyClical.ixt`
- `PyClical.fill`
- `PyClical.scalar_epsilon` = `epsilon`
- `float PyClical.pi` = `atan(clifford(1.0)) * 4.0`
- `float PyClical.tau` = `atan(clifford(1.0)) * 8.0`
- `PyClical.cl` = `clifford`
- `PyClical.ist` = `index_set`
- `PyClical.ninf3` = `e(4) + e(-1)`
- `PyClical.nbar3` = `e(4) - e(-1)`

## 7.62 PyClical.pyx

[Go to the documentation of this file.](#)

```

00001 # -*- coding: utf-8 -*-
00002 # cython: language_level=3
00003 # distutils: language = c++
00004 #
00005 # PyClical: Python interface to GluCat:
00006 # Generic library of universal Clifford algebra templates
00007 #
00008 # PyClical.pyx: Cython definitions visible from Python.
00009 #
00010 # copyright : (C) 2008-2021 by Paul C. Leopardi
00011 #
00012 # This library is free software: you can redistribute it and/or modify
00013 # it under the terms of the GNU Lesser General Public License as published
00014 # by the Free Software Foundation, either version 3 of the License, or
00015 # (at your option) any later version.
00016 #
00017 # This library is distributed in the hope that it will be useful,
00018 # but WITHOUT ANY WARRANTY; without even the implied warranty of
00019 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00020 # GNU Lesser General Public License for more details.
00021 #
00022 # You should have received a copy of the GNU Lesser General Public License
00023 # along with this library. If not, see <http://www.gnu.org/licenses/>.
00024 #
00025 # References for definitions:
00026 # [DL]:
00027 # C. Doran and A. Lasenby, "Geometric algebra for physicists", Cambridge, 2003.
00028 #
00029 import math
00030 import numbers
00031 import collections
00032 #
00033 from PyClical cimport *
00034 #
00035 __version__ = str(glucat_package_version,'utf-8')
00036 #
00037 # Forward reference
00038 cdef class index_set
00039 #
00040 cdef inline IndexSet toIndexSet(obj):
00041 """
00042 Return the C++ IndexSet instance wrapped by index_set(obj).
00043 """
00044 return index_set(obj).instance[0]
00045 #
00046 cdef class index_set:
00047 """
00048 Python class index_set wraps C++ class IndexSet.

```

```

00049 """
00050 cdef IndexSet *instance # Wrapped instance of C++ class IndexSet.
00051
00052 cdef inline wrap(index_set self, IndexSet other):
00053 """
00054 Wrap an instance of the C++ class IndexSet.
00055 """
00056 self.instanceinstance[0] = other
00057 return self
00058
00059 cdef inline IndexSet unwrap(index_set self):
00060 """
00061 Return the wrapped C++ IndexSet instance.
00062 """
00063 return self.instanceinstance[0]
00064
00065 cpdef copy(index_set self):
00066 """
00067 Copy this index_set object.
00068
00069 >> s=index_set(1); t=s.copy(); print(t)
00070 {1}
00071 """
00072 return index_set(self)
00073
00074 def __cinit__(self, other = 0):
00075 """
00076 Construct an object of type index_set.
00077
00078 >> print(index_set(1))
00079 {1}
00080 >> print(index_set({1,2}))
00081 {1,2}
00082 >> print(index_set(index_set({1,2})))
00083 {1,2}
00084 >> print(index_set({1,2}))
00085 {1,2}
00086 >> print(index_set({1,2,1}))
00087 {1,2}
00088 >> print(index_set("{1,2,1}"))
00089 {1,2}
00090 >> print(index_set(""))
00091 {}
00092 """
00093 error_msg_prefix = "Cannot initialize index_set object from"
00094 if isinstance(other, index_set):
00095 self.instanceinstance = new IndexSet((<index_set>other).unwrap())
00096 elif isinstance(other, numbers.Integral):
00097 self.instanceinstance = new IndexSet(<int>other)
00098 elif isinstance(other, (set, frozenset)):
00099 try:
00100 self.instanceinstance = new IndexSet()
00101 for idx in other:
00102 self[idx] = True
00103 except IndexError:
00104 raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")
00105 except (RuntimeError, TypeError):
00106 raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
00107 elif isinstance(other, str):
00108 try:
00109 bother = other.encode("UTF-8")
00110 self.instanceinstance = new IndexSet(<char *>bother)
00111 except RuntimeError:
00112 raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
00113 else:
00114 raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
00115
00116 def __dealloc__(self):
00117 """
00118 Clean up by deallocating the instance of C++ class IndexSet.
00119 """
00120 del self.instanceinstance
00121
00122 def __richcmp__(lhs, rhs, int op):
00123 """
00124 Compare two objects of class index_set.
00125
00126 >> index_set(1) == index_set({1})
00127 True
00128 >> index_set({1}) != index_set({1})
00129 False
00130 >> index_set({1}) != index_set({2})
00131 True
00132 >> index_set({1}) == index_set({2})
00133 False
00134 >> index_set({1}) < index_set({2})
00135 True

```

```

00136 >> index_set({1}) <= index_set({2})
00137 True
00138 >> index_set({1}) > index_set({2})
00139 False
00140 >> index_set({1}) >= index_set({2})
00141 False
00142 """
00143 if (lhs is None) or (rhs is None):
00144 eq = bool(lhs is rhs)
00145 if op == 2: # ==
00146 return eq
00147 elif op == 3: # !=
00148 return not eq
00149 else:
00150 if op == 0: # <
00151 return False
00152 elif op == 1: # <=
00153 return eq
00154 elif op == 4: # >
00155 return False
00156 elif op == 5: # >=
00157 return eq
00158 else:
00159 return NotImplemented
00160 else:
00161 eq = bool(toIndexSet(lhs) == toIndexSet(rhs))
00162 if op == 2: # ==
00163 return eq
00164 elif op == 3: # !=
00165 return not eq
00166 else:
00167 lt = bool(toIndexSet(lhs) < toIndexSet(rhs))
00168 if op == 0: # <
00169 return lt
00170 elif op == 1: # <=
00171 return lt or eq
00172 elif op == 4: # >
00173 return not (lt or eq)
00174 elif op == 5: # >=
00175 return not lt
00176 else:
00177 return NotImplemented
00178
00179 def __setitem__(self, idx, val):
00180 """
00181 Set the value of an index_set object at index idx to value val.
00182
00183 >> s=index_set({1}); s[2] = True; print(s)
00184 {1,2}
00185 >> s=index_set({1,2}); s[1] = False; print(s)
00186 {2}
00187 """
00188 self.instanceinstance.set(idx, val)
00189 return
00190
00191 def __getitem__(self, idx):
00192 """
00193 Get the value of an index_set object at an index.
00194
00195 >> index_set({1})[1]
00196 True
00197 >> index_set({1})[2]
00198 False
00199 >> index_set({2})[-1]
00200 False
00201 >> index_set({2})[1]
00202 False
00203 >> index_set({2})[2]
00204 True
00205 >> index_set({2})[33]
00206 False
00207 """
00208 return self.instanceinstance.getitem(idx)
00209
00210 def __contains__(self, idx):
00211 """
00212 Check that an index_set object contains the index idx: idx in self.
00213
00214 >> 1 in index_set({1})
00215 True
00216 >> 2 in index_set({1})
00217 False
00218 >> -1 in index_set({2})
00219 False
00220 >> 1 in index_set({2})
00221 False
00222 >> 2 in index_set({2})

```

```

00223 True
00224 >> 33 in index_set({2})
00225 False
00226 """
00227 return self.instanceinstance.getitem(idx)
00228
00229 def __iter__(self):
00230 """
00231 Iterate over the indices of an index_set.
00232
00233 >> for i in index_set({-3,4,7}):print(i, end=",")
00234 -3,4,7,
00235 """
00236 for idx in range(self.min(), self.max()+1):
00237 if idx in self:
00238 yield idx
00239
00240 def __invert__(self):
00241 """
00242 Set complement: not.
00243
00244 >>
00245 print(~index_set({-16,-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}))
00246 {-32,-31,-30,-29,-28,-27,-26,-25,-24,-23,-22,-21,-20,-19,-18,-17,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32}
00247 """
00248 return index_set().wrap(self.instanceinstance.invert())
00249
00250 def __xor__(lhs, rhs):
00251 """
00252 Symmetric set difference: exclusive or.
00253
00254 >> print(index_set({1}) ^ index_set({2}))
00255 {1,2}
00256 >> print(index_set({1,2}) ^ index_set({2}))
00257 {1}
00258 """
00259 return index_set().wrap(toIndexSet(lhs) ^ toIndexSet(rhs))
00260
00261 def __ixor__(self, rhs):
00262 """
00263 Symmetric set difference: exclusive or.
00264
00265 >> x = index_set({1}); x ^= index_set({2}); print(x)
00266 {1,2}
00267 >> x = index_set({1,2}); x ^= index_set({2}); print(x)
00268 {1}
00269 """
00270 return self.wrap(self.unwrap() ^ toIndexSet(rhs))
00271
00272 def __and__(lhs, rhs):
00273 """
00274 Set intersection: and.
00275
00276 >> print(index_set({1}) & index_set({2}))
00277 {}
00278 >> print(index_set({1,2}) & index_set({2}))
00279 {2}
00280 """
00281 return index_set().wrap(toIndexSet(lhs) & toIndexSet(rhs))
00282
00283 def __iand__(self, rhs):
00284 """
00285 Set intersection: and.
00286
00287 >> x = index_set({1}); x &= index_set({2}); print(x)
00288 {}
00289 >> x = index_set({1,2}); x &= index_set({2}); print(x)
00290 {2}
00291 """
00292 return self.wrap(self.unwrap() & toIndexSet(rhs))
00293
00294 def __or__(lhs, rhs):
00295 """
00296 Set union: or.
00297
00298 >> print(index_set({1}) | index_set({2}))
00299 {1,2}
00300 >> print(index_set({1,2}) | index_set({2}))
00301 {1,2}
00302 """
00303 return index_set().wrap(toIndexSet(lhs) | toIndexSet(rhs))
00304
00305 def __ior__(self, rhs):
00306 """
00307 Set union: or.

```



```

00308 >> x = index_set({1}); x |= index_set({2}); print(x)
00309 {1,2}
00310 >> x = index_set({1,2}); x |= index_set({2}); print(x)
00311 {1,2}
00312 """
00313 return self.wrap(self.unwrap() | toIndexSet(rhs))
00314
00315 def count(self):
00316 """
00317 Cardinality: Number of indices included in set.
00318
00319 >> index_set({-1,1,2}).count()
00320 3
00321 """
00322 return self.instanceinstance.count()
00323
00324 def count_neg(self):
00325 """
00326 Number of negative indices included in set.
00327
00328 >> index_set({-1,1,2}).count_neg()
00329 1
00330 """
00331 return self.instanceinstance.count_neg()
00332
00333 def count_pos(self):
00334 """
00335 Number of positive indices included in set.
00336
00337 >> index_set({-1,1,2}).count_pos()
00338 2
00339 """
00340 return self.instanceinstance.count_pos()
00341
00342 def min(self):
00343 """
00344 Minimum member.
00345
00346 >> index_set({-1,1,2}).min()
00347 -1
00348 """
00349 return self.instanceinstance.min()
00350
00351 def max(self):
00352 """
00353 Maximum member.
00354
00355 >> index_set({-1,1,2}).max()
00356 2
00357 """
00358 return self.instanceinstance.max()
00359
00360 def hash_fn(self):
00361 """
00362 Hash function.
00363 """
00364 return self.instanceinstance.hash_fn()
00365
00366 def sign_of_mult(self, rhs):
00367 """
00368 Sign of geometric product of two Clifford basis elements.
00369
00370 >> s = index_set({1,2}); t=index_set({-1}); s.sign_of_mult(t)
00371 1
00372 """
00373 return self.instanceinstance.sign_of_mult(toIndexSet(rhs))
00374
00375 def sign_of_square(self):
00376 """
00377 Sign of geometric square of a Clifford basis element.
00378
00379 >> s = index_set({1,2}); s.sign_of_square()
00380 -1
00381 """
00382 return self.instanceinstance.sign_of_square()
00383
00384 def __repr__(self):
00385 """
00386 The "official" string representation of self.
00387
00388 >> index_set({1,2}).__repr__()
00389 'index_set({1,2})'
00390 >> repr(index_set({1,2}))
00391 'index_set({1,2})'
00392 """
00393 return index_set_to_repr(self.unwrap()).decode()
00394

```

```

00395 def __str__(self):
00396 """
00397 The "informal" string representation of self.
00398
00399 >> index_set({1,2}).__str__()
00400 '{1,2}'
00401 >> str(index_set({1,2}))
00402 '{1,2}'
00403 """
00404 return index_set_to_str(self.unwrap()).decode()
00405
00406 def index_set_hidden_doctests():
00407 """
00408 Tests for functions that Doctest cannot see.
00409
00410 For index_set.__cinit__: Construct index_set.
00411
00412 >> print(index_set(1))
00413 {1}
00414 >> print(index_set({1,2}))
00415 {1,2}
00416 >> print(index_set(index_set({1,2})))
00417 {1,2}
00418 >> print(index_set({1,2}))
00419 {1,2}
00420 >> print(index_set({1,2,1}))
00421 {1,2}
00422 >> print(index_set({1,2,1}))
00423 {1,2}
00424 >> print(index_set(""))
00425 {}
00426 >> print(index_set("{}"))
00427 Traceback (most recent call last):
00428 ...
00429 ValueError: Cannot initialize index_set object from invalid string '{}'.
00430 >> print(index_set("{1}"))
00431 Traceback (most recent call last):
00432 ...
00433 ValueError: Cannot initialize index_set object from invalid string '{1}'.
00434 >> print(index_set("{1,2,100}"))
00435 Traceback (most recent call last):
00436 ...
00437 ValueError: Cannot initialize index_set object from invalid string '{1,2,100}'.
00438 >> print(index_set({1,2,100}))
00439 Traceback (most recent call last):
00440 ...
00441 IndexError: Cannot initialize index_set object from invalid {1, 2, 100}.
00442 >> print(index_set([1,2]))
00443 Traceback (most recent call last):
00444 ...
00445 TypeError: Cannot initialize index_set object from <class 'list'>.
00446
00447 For index_set.__richcmp__: Compare two objects of class index_set.
00448
00449 >> index_set(1) == index_set({1})
00450 True
00451 >> index_set({1}) != index_set({1})
00452 False
00453 >> index_set({1}) != index_set({2})
00454 True
00455 >> index_set({1}) == index_set({2})
00456 False
00457 >> index_set({1}) < index_set({2})
00458 True
00459 >> index_set({1}) <= index_set({2})
00460 True
00461 >> index_set({1}) > index_set({2})
00462 False
00463 >> index_set({1}) >= index_set({2})
00464 False
00465 >> None == index_set({1,2})
00466 False
00467 >> None != index_set({1,2})
00468 True
00469 >> None < index_set({1,2})
00470 False
00471 >> None <= index_set({1,2})
00472 False
00473 >> None > index_set({1,2})
00474 False
00475 >> None >= index_set({1,2})
00476 False
00477 >> index_set({1,2}) == None
00478 False
00479 >> index_set({1,2}) != None
00480 True
00481 >> index_set({1,2}) < None

```

```

00482 False
00483 >> index_set({1,2}) <= None
00484 False
00485 >> index_set({1,2}) > None
00486 False
00487 >> index_set({1,2}) >= None
00488 False
00489 """
00490 return
00491
00492 cpdef inline compare(lhs,rhs):
00493 """
00494 "lexicographic compare" eg. {3,4,5} is less than {3,7,8};
00495 -1 if a<b, +1 if a>b, 0 if a==b.
00496
00497 >> compare(index_set({1,2}),index_set({-1,3}))
00498 -1
00499 >> compare(index_set({-1,4}),index_set({-1,3}))
00500 1
00501 """
00502 return glucat.compare(toIndexSet(lhs), toIndexSet(rhs))
00503
00504 cpdef inline min_neg(obj):
00505 """
00506 Minimum negative index, or 0 if none.
00507
00508 >> min_neg(index_set({1,2}))
00509 0
00510 """
00511 return glucat.min_neg(toIndexSet(obj))
00512
00513 cpdef inline max_pos(obj):
00514 """
00515 Maximum positive index, or 0 if none.
00516
00517 >> max_pos(index_set({1,2}))
00518 2
00519 """
00520 return glucat.max_pos(toIndexSet(obj))
00521
00522 cdef inline vector[scalar_t] list_to_vector(lst):
00523 """
00524 Create a C++ std::vector[scalar_t] from an iterable Python object.
00525 """
00526 cdef vector[scalar_t] v
00527 for s in lst:
00528 v.push_back(<scalar_t>s)
00529 return v
00530
00531 # Forward reference.
00532 cdef class clifford
00533
00534 cdef inline Clifford toClifford(obj):
00535 return clifford(obj).instance[0]
00536
00537 cdef class clifford:
00538 """
00539 Python class clifford wraps C++ class Clifford.
00540 """
00541 cdef Clifford *instance # Wrapped instance of C++ class Clifford.
00542
00543 cdef inline wrap(clifford self, Clifford other):
00544 """
00545 Wrap an instance of the C++ class Clifford.
00546 """
00547 self.instance[0] = other
00548 return self
00549
00550 cdef inline Clifford unwrap(clifford self):
00551 """
00552 Return the wrapped C++ Clifford instance.
00553 """
00554 return self.instance[0]
00555
00556 cpdef copy(clifford self):
00557 """
00558 Copy this clifford object.
00559
00560 >> x=clifford("1{2}"); y=x.copy(); print(y)
00561 {2}
00562 """
00563 return clifford(self)
00564
00565 def __cinit__(self, other = 0, ixt = None):
00566 """
00567 Construct an object of type clifford.
00568

```

```

00569 >> print(clifford(2))
00570 2
00571 >> print(clifford(2.0))
00572 2
00573 >> print(clifford(1.0e-1))
00574 0.1
00575 >> print(clifford("2"))
00576 2
00577 >> print(clifford("2{1,2,3}"))
00578 2{1,2,3}
00579 >> print(clifford(clifford("2{1,2,3}")))
00580 2{1,2,3}
00581 >> print(clifford("-{1}"))
00582 -{1}
00583 >> print(clifford(2, index_set({1,2})))
00584 2{1,2}
00585 >> print(clifford([2,3], index_set({1,2})))
00586 2{1}+3{2}
00587 """
00588 error_msg_prefix = "Cannot initialize clifford object from"
00589 if ixt is None:
00590 try:
00591 if isinstance(other, clifford):
00592 self.instance = new Clifford((<clifford>other).unwrap())
00593 elif isinstance(other, index_set):
00594 self.instance = new Clifford((<index_set>other).unwrap(), <scalar_t>1.0)
00595 elif isinstance(other, numbers.Real):
00596 self.instance = new Clifford(<scalar_t>other)
00597 elif isinstance(other, str):
00598 try:
00599 bother = other.encode("UTF-8")
00600 self.instance = new Clifford(<char *>bother)
00601 except RuntimeError:
00602 raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
00603 else:
00604 raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
00605 except RuntimeError as err:
00606 raise ValueError(error_msg_prefix + " " + str(type(other))
00607 + " value " + repr(other) + ":"
00608 + "\n\t" + str(err))
00609 elif isinstance(ixt, index_set):
00610 if isinstance(other, numbers.Real):
00611 self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
00612 elif isinstance(other, collections.abc.Sequence):
00613 self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
00614 else:
00615 raise TypeError(error_msg_prefix + " (" + str(type(other))
00616 + ", " + repr(ixt) + ").")
00617 else:
00618 raise TypeError(error_msg_prefix + " (" + str(type(other))
00619 + ", " + str(type(ixt)) + ").")
00620
00621 def __dealloc__(self):
00622 """
00623 Clean up by deallocating the instance of C++ class Clifford.
00624 """
00625 del self.instance
00626
00627 def __contains__(self, x):
00628 """
00629 Not applicable.
00630
00631 >> x=clifford(index_set({-3,4,7})); -3 in x
00632 Traceback (most recent call last):
00633 ...
00634 TypeError: Not applicable.
00635 """
00636 raise TypeError("Not applicable.")
00637
00638 def __iter__(self):
00639 """
00640 Not applicable.
00641
00642 >> for a in clifford(index_set({-3,4,7})):print(a, end=",")
00643 Traceback (most recent call last):
00644 ...
00645 TypeError: Not applicable.
00646 """
00647 raise TypeError("Not applicable.")
00648
00649 def reframe(self, ixt):
00650 """
00651 Put self into a larger frame, containing the union of self.frame() and index set ixt.
00652 This can be used to make multiplication faster, by multiplying within a common frame.
00653
00654 >> clifford("2+3{1}").reframe(index_set({1,2,3}))
00655 clifford("2+3{1}")

```

```

00656 >> s=index_set({1,2,3});t=index_set({-3,-2,-1});x=random_clifford(s); x.reframe(t).frame() ==
(s|t);
00657 True
00658 """
00659 error_msg_prefix = "Cannot reframe"
00660 if isinstance(ixt, index_set):
00661 try:
00662 result = clifford()
00663 result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
00664 except RuntimeError as err:
00665 raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
00666 + str(ixt) + ":",
00667 + "\n\t" + str(err))
00668 else:
00669 raise TypeError(error_msg_prefix + " using (" + str(type(ixt)) + ").")
00670 return result
00671
00672 def __richcmp__(lhs, rhs, int op):
00673 """
00674 Compare objects of type clifford.
00675
00676 >> clifford("{1}") == clifford("1{1}")
00677 True
00678 >> clifford("{1}") != clifford("1.0{1}")
00679 False
00680 >> clifford("{1}") != clifford("1.0")
00681 True
00682 >> clifford("{1,2}") == None
00683 False
00684 >> clifford("{1,2}") != None
00685 True
00686 >> None == clifford("{1,2}")
00687 False
00688 >> None != clifford("{1,2}")
00689 True
00690 """
00691 if op == 2: # ==
00692 if (lhs is None) or (rhs is None):
00693 return bool(lhs is rhs)
00694 else:
00695 return bool(toClifford(lhs) == toClifford(rhs))
00696 elif op == 3: # !=
00697 if (lhs is None) or (rhs is None):
00698 return not bool(lhs is rhs)
00699 else:
00700 return bool(toClifford(lhs) != toClifford(rhs))
00701 elif isinstance(lhs, clifford) or isinstance(rhs, clifford):
00702 raise TypeError("This comparison operator is not implemented for "
00703 + str(type(lhs)) + ", " + str(type(rhs)) + ".")
00704 else:
00705 return NotImplemented
00706
00707 def __getitem__(self, ixt):
00708 """
00709 Subscripting: map from index set to scalar coordinate.
00710
00711 >> clifford("{1}")[index_set(1)]
00712 1.0
00713 >> clifford("{1}")[index_set({1})]
00714 1.0
00715 >> clifford("{1}")[index_set({1,2})]
00716 0.0
00717 >> clifford("2{1,2}")[index_set({1,2})]
00718 2.0
00719 """
00720 return self.instance.getitem(toIndexSet(ixt))
00721
00722 def __neg__(self):
00723 """
00724 Unary -.
00725
00726 >> print(-clifford("{1}"))
00727 -{1}
00728 """
00729 return clifford().wrap(self.instance.neg())
00730
00731 def __pos__(self):
00732 """
00733 Unary +.
00734
00735 >> print(+clifford("{1}"))
00736 {1}
00737 """
00738 return clifford(self)
00739
00740 def __add__(lhs, rhs):
00741 """

```

```

00742 Geometric sum.
00743
00744 >> print(clifford(1) + clifford("{2}"))
00745 1+{2}
00746 >> print(clifford("{1}") + clifford("{2}"))
00747 {1}+{2}
00748 """
00749 return clifford().wrap(toClifford(lhs) + toClifford(rhs))
00750
00751 def __iadd__(self, rhs):
00752 """
00753 Geometric sum.
00754
00755 >> x = clifford(1); x += clifford("{2}"); print(x)
00756 1+{2}
00757 """
00758 return self.wrap(self.unwrap() + toClifford(rhs))
00759
00760 def __sub__(lhs, rhs):
00761 """
00762 Geometric difference.
00763
00764 >> print(clifford(1) - clifford("{2}"))
00765 1-{2}
00766 >> print(clifford("{1}") - clifford("{2}"))
00767 {1}-{2}
00768 """
00769 return clifford().wrap(toClifford(lhs) - toClifford(rhs))
00770
00771 def __isub__(self, rhs):
00772 """
00773 Geometric difference.
00774
00775 >> x = clifford(1); x -= clifford("{2}"); print(x)
00776 1-{2}
00777 """
00778 return self.wrap(self.unwrap() - toClifford(rhs))
00779
00780 def __mul__(lhs, rhs):
00781 """
00782 Geometric product.
00783
00784 >> print(clifford("{1}") * clifford("{2}"))
00785 {1,2}
00786 >> print(clifford(2) * clifford("{2}"))
00787 2{2}
00788 >> print(clifford("{1}") * clifford("{1,2}"))
00789 {2}
00790 """
00791 return clifford().wrap(toClifford(lhs) * toClifford(rhs))
00792
00793 def __imul__(self, rhs):
00794 """
00795 Geometric product.
00796
00797 >> x = clifford(2); x *= clifford("{2}"); print(x)
00798 2{2}
00799 >> x = clifford("{1}"); x *= clifford("{2}"); print(x)
00800 {1,2}
00801 >> x = clifford("{1}"); x *= clifford("{1,2}"); print(x)
00802 {2}
00803 """
00804 return self.wrap(self.unwrap() * toClifford(rhs))
00805
00806 def __mod__(lhs, rhs):
00807 """
00808 Contraction.
00809
00810 >> print(clifford("{1}") % clifford("{2}"))
00811 0
00812 >> print(clifford(2) % clifford("{2}"))
00813 2{2}
00814 >> print(clifford("{1}") % clifford("{1}"))
00815 1
00816 >> print(clifford("{1}") % clifford("{1,2}"))
00817 {2}
00818 """
00819 return clifford().wrap(toClifford(lhs) % toClifford(rhs))
00820
00821 def __imod__(self, rhs):
00822 """
00823 Contraction.
00824
00825 >> x = clifford("{1}"); x %= clifford("{2}"); print(x)
00826 0
00827 >> x = clifford(2); x %= clifford("{2}"); print(x)
00828 2{2}

```

```

00829 >> x = clifford("{1}"); x %= clifford("{1}"); print(x)
00830 1
00831 >> x = clifford("{1}"); x %= clifford("{1,2}"); print(x)
00832 {2}
00833 """
00834 return self.wrap(self.unwrap() % toClifford(rhs))
00835
00836 def __and__(lhs, rhs):
00837 """
00838 Inner product.
00839
00840 >> print(clifford("{1}") & clifford("{2}"))
00841 0
00842 >> print(clifford(2) & clifford("{2}"))
00843 0
00844 >> print(clifford("{1}") & clifford("{1}"))
00845 1
00846 >> print(clifford("{1}") & clifford("{1,2}"))
00847 {2}
00848 """
00849 return clifford().wrap(toClifford(lhs) & toClifford(rhs))
00850
00851 def __iand__(self, rhs):
00852 """
00853 Inner product.
00854
00855 >> x = clifford("{1}"); x &= clifford("{2}"); print(x)
00856 0
00857 >> x = clifford(2); x &= clifford("{2}"); print(x)
00858 0
00859 >> x = clifford("{1}"); x &= clifford("{1}"); print(x)
00860 1
00861 >> x = clifford("{1}"); x &= clifford("{1,2}"); print(x)
00862 {2}
00863 """
00864 return self.wrap(self.unwrap() & toClifford(rhs))
00865
00866 def __xor__(lhs, rhs):
00867 """
00868 Outer product.
00869
00870 >> print(clifford("{1}") ^ clifford("{2}"))
00871 {1,2}
00872 >> print(clifford(2) ^ clifford("{2}"))
00873 2{2}
00874 >> print(clifford("{1}") ^ clifford("{1}"))
00875 0
00876 >> print(clifford("{1}") ^ clifford("{1,2}"))
00877 0
00878 """
00879 return clifford().wrap(toClifford(lhs) ^ toClifford(rhs))
00880
00881 def __ixor__(self, rhs):
00882 """
00883 Outer product.
00884
00885 >> x = clifford("{1}"); x ^= clifford("{2}"); print(x)
00886 {1,2}
00887 >> x = clifford(2); x ^= clifford("{2}"); print(x)
00888 2{2}
00889 >> x = clifford("{1}"); x ^= clifford("{1}"); print(x)
00890 0
00891 >> x = clifford("{1}"); x ^= clifford("{1,2}"); print(x)
00892 0
00893 """
00894 return self.wrap(self.unwrap() ^ toClifford(rhs))
00895
00896 def __truediv__(lhs, rhs):
00897 """
00898 Geometric quotient.
00899
00900 >> print(clifford("{1}") / clifford("{2}"))
00901 {1,2}
00902 >> print(clifford(2) / clifford("{2}"))
00903 2{2}
00904 >> print(clifford("{1}") / clifford("{1}"))
00905 1
00906 >> print(clifford("{1}") / clifford("{1,2}"))
00907 -{2}
00908 """
00909 return clifford().wrap(toClifford(lhs) / toClifford(rhs))
00910
00911 def __idiv__(self, rhs):
00912 """
00913 Geometric quotient.
00914
00915 >> x = clifford("{1}"); x /= clifford("{2}"); print(x)

```

```

00916 {1,2}
00917 >> x = clifford(2); x /= clifford("{2}"); print(x)
00918 2{2}
00919 >> x = clifford("{1}"); x /= clifford("{1}"); print(x)
00920 1
00921 >> x = clifford("{1}"); x /= clifford("{1,2}"); print(x)
00922 -{2}
00923 """
00924 return self.wrap(self.unwrap() / toClifford(rhs))
00925
00926 def inv(self):
00927 """
00928 Geometric multiplicative inverse.
00929
00930 >> x = clifford("{1}"); print(x.inv())
00931 {1}
00932 >> x = clifford(2); print(x.inv())
00933 0.5
00934 >> x = clifford("{1,2}"); print(x.inv())
00935 -{1,2}
00936 """
00937 return clifford().wrap(self.instance.inv())
00938
00939 def __or__(lhs, rhs):
00940 """
00941 Transform left hand side, using right hand side as a transformation.
00942
00943 >> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); print(y|x)
00944 -{1}
00945 >> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); print(y|exp(x))
00946 -{1}
00947 """
00948 return clifford().wrap(toClifford(lhs) | toClifford(rhs))
00949
00950 def __ior__(self, rhs):
00951 """
00952 Transform left hand side, using right hand side as a transformation.
00953
00954 >> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); y|=x; print(y)
00955 -{1}
00956 >> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); y|=exp(x); print(y)
00957 -{1}
00958 """
00959 return self.wrap(self.unwrap() | toClifford(rhs))
00960
00961 def __pow__(self, m, dummy):
00962 """
00963 Power: self to the m.
00964
00965 >> x=clifford("{1}"); print(x ** 2)
00966 1
00967 >> x=clifford("2"); print(x ** 2)
00968 4
00969 >> x=clifford("2+{1}"); print(x ** 0)
00970 1
00971 >> x=clifford("2+{1}"); print(x ** 1)
00972 2+{1}
00973 >> x=clifford("2+{1}"); print(x ** 2)
00974 5+4{1}
00975 >> i=clifford("{1,2}"); print(exp(pi/2) * (i ** i))
00976 1
00977 """
00978 return pow(self, m)
00979
00980 def pow(self, m):
00981 """
00982 Power: self to the m.
00983
00984 >> x=clifford("{1}"); print(x.pow(2))
00985 1
00986 >> x=clifford("2"); print(x.pow(2))
00987 4
00988 >> x=clifford("2+{1}"); print(x.pow(0))
00989 1
00990 >> x=clifford("2+{1}"); print(x.pow(1))
00991 2+{1}
00992 >> x=clifford("2+{1}"); print(x.pow(2))
00993 5+4{1}
00994 >> print(clifford("1+{1}+{1,2}").pow(3))
00995 1+3{1}+3{1,2}
00996 >> i=clifford("{1,2}"); print(exp(pi/2) * i.pow(i))
00997 1
00998 """
00999 if isinstance(m, numbers.Integral):
01000 return clifford().wrap(self.instance.pow(m))
01001 else:
01002 return exp(m * log(self))

```



```

01003
01004 def outer_pow(self, m):
01005 """
01006 Outer product power.
01007
01008 >> x=clifford("2+{1}"); print(x.outer_pow(0))
01009 1
01010 >> x=clifford("2+{1}"); print(x.outer_pow(1))
01011 2+{1}
01012 >> x=clifford("2+{1}"); print(x.outer_pow(2))
01013 4+4{1}
01014 >> print(clifford("1+{1}+{1,2}").outer_pow(3))
01015 1+3{1}+3{1,2}
01016
01017 """
01018 return clifford().wrap(self.instance.outer_pow(m))
01019
01020 def __call__(self, grade):
01021 """
01022 Pure grade-vector part.
01023
01024 >> print(clifford("{1}") (1))
01025 {1}
01026 >> print(clifford("{1}") (0))
01027 0
01028 >> print(clifford("1+{1}+{1,2}") (0))
01029 1
01030 >> print(clifford("1+{1}+{1,2}") (1))
01031 {1}
01032 >> print(clifford("1+{1}+{1,2}") (2))
01033 {1,2}
01034 >> print(clifford("1+{1}+{1,2}") (3))
01035 0
01036 """
01037 return clifford().wrap(self.instance.call(grade))
01038
01039 def scalar(self):
01040 """
01041 Scalar part.
01042
01043 >> clifford("1+{1}+{1,2}").scalar()
01044 1.0
01045 >> clifford("{1,2}").scalar()
01046 0.0
01047 """
01048 return self.instance.scalar()
01049
01050 def pure(self):
01051 """
01052 Pure part.
01053
01054 >> print(clifford("1+{1}+{1,2}").pure())
01055 {1}+{1,2}
01056 >> print(clifford("{1,2}").pure())
01057 {1,2}
01058 """
01059 return clifford().wrap(self.instance.pure())
01060
01061 def even(self):
01062 """
01063 Even part of multivector, sum of even grade terms.
01064
01065 >> print(clifford("1+{1}+{1,2}").even())
01066 1+{1,2}
01067 """
01068 return clifford().wrap(self.instance.even())
01069
01070 def odd(self):
01071 """
01072 Odd part of multivector, sum of odd grade terms.
01073
01074 >> print(clifford("1+{1}+{1,2}").odd())
01075 {1}
01076 """
01077 return clifford().wrap(self.instance.odd())
01078
01079 def vector_part(self, frm = None):
01080 """
01081 Vector part of multivector, as a Python list, with respect to frm.
01082
01083 >> print(clifford("1+2{1}+3{2}+4{1,2}").vector_part())
01084 [2.0, 3.0]
01085 >> print(clifford("1+2{1}+3{2}+4{1,2}").vector_part(index_set({-1,1,2})))
01086 [0.0, 2.0, 3.0]
01087 """
01088 error_msg_prefix = "Cannot take vector part of "
01089 cdef vector[scalar_t] vec

```

```

01090 cdef int n
01091 cdef int i
01092 try:
01093 if frm is None:
01094 vec = self.instance.vector_part()
01095 else:
01096 vec = self.instance.vector_part((<index_set>frm).unwrap())
01097 n = vec.size()
01098 lst = [0.0]*n
01099 for i in xrange(n):
01100 lst[i] = vec[i]
01101 return lst
01102 except RuntimeError as err:
01103 raise ValueError(error_msg_prefix + str(self) + " using invalid "
01104 + repr(frm) + " as frame:\n\t"
01105 + str(err))
01106
01107 def involute(self):
01108 """
01109 Main involution, each {i} is replaced by -{i} in each term,
01110 eg. clifford("{1}") -> -clifford("{1}").
01111
01112 >> print(clifford("{1}").involute())
01113 -{1}
01114 >> print((clifford("{2}") * clifford("{1}")).involute())
01115 -{1,2}
01116 >> print((clifford("{1}") * clifford("{2}")).involute())
01117 {1,2}
01118 >> print(clifford("1+{1}+{1,2}").involute())
01119 1-{1}+{1,2}
01120 """
01121 return clifford().wrap(self.instance.involute())
01122
01123 def reverse(self):
01124 """
01125 Reversion, eg. clifford("{1}")*clifford("{2}") -> clifford("{2}")*clifford("{1}").
01126
01127 >> print(clifford("{1}").reverse())
01128 {1}
01129 >> print((clifford("{2}") * clifford("{1}")).reverse())
01130 {1,2}
01131 >> print((clifford("{1}") * clifford("{2}")).reverse())
01132 -{1,2}
01133 >> print(clifford("1+{1}+{1,2}").reverse())
01134 1+{1}-{1,2}
01135 """
01136 return clifford().wrap(self.instance.reverse())
01137
01138 def conj(self):
01139 """
01140 Conjugation, reverse o involute == involute o reverse.
01141
01142 >> print((clifford("{1}")).conj())
01143 -{1}
01144 >> print((clifford("{2}") * clifford("{1}")).conj())
01145 {1,2}
01146 >> print((clifford("{1}") * clifford("{2}")).conj())
01147 -{1,2}
01148 >> print(clifford("1+{1}+{1,2}").conj())
01149 1-{1}-{1,2}
01150 """
01151 return clifford().wrap(self.instance.conj())
01152
01153 def quad(self):
01154 """
01155 Quadratic form == (rev(x)*x)(0).
01156
01157 >> print(clifford("1+{1}+{1,2}").quad())
01158 3.0
01159 >> print(clifford("1+{-1}+{1,2}+{1,2,3}").quad())
01160 2.0
01161 """
01162 return self.instance.quad()
01163
01164 def norm(self):
01165 """
01166 Norm == sum of squares of coordinates.
01167
01168 >> clifford("1+{1}+{1,2}").norm()
01169 3.0
01170 >> clifford("1+{-1}+{1,2}+{1,2,3}").norm()
01171 4.0
01172 """
01173 return self.instance.norm()
01174
01175 def abs(self):
01176 """

```

```

01177 Absolute value: square root of norm.
01178
01179 »> clifford("1+{-1}+{1,2}+{1,2,3}").abs()
01180 2.0
01181 """
01182 return glucat.abs(self.unwrap())
01183
01184 def max_abs(self):
01185 """
01186 Maximum of absolute values of components of multivector: multivector infinity norm.
01187
01188 »> clifford("1+{-1}+{1,2}+{1,2,3}").max_abs()
01189 1.0
01190 »> clifford("3+2{1}+{1,2}").max_abs()
01191 3.0
01192 """
01193 return self.instance.max_abs()
01194
01195 def truncated(self, limit):
01196 """
01197 Remove all terms of self with relative size smaller than limit.
01198
01199 »> clifford("1e8+{1}+1e-8{1,2}").truncated(1.0e-6)
01200 clifford("100000000")
01201 »> clifford("1e4+{1}+1e-4{1,2}").truncated(1.0e-6)
01202 clifford("10000+{1}")
01203 """
01204 return clifford().wrap(self.instance.truncated(limit))
01205
01206 def isinf(self):
01207 """
01208 Check if a multivector contains any infinite values.
01209
01210 »> clifford().isinf()
01211 False
01212 """
01213 return self.instance.isnan()
01214
01215 def isnan(self):
01216 """
01217 Check if a multivector contains any IEEE NaN values.
01218
01219 »> clifford().isnan()
01220 False
01221 """
01222 return self.instance.isnan()
01223
01224 def frame(self):
01225 """
01226 Subalgebra generated by all generators of terms of given multivector.
01227
01228 »> print(clifford("1+3{-1}+2{1,2}+4{-2,7}").frame())
01229 {-2,-1,1,2,7}
01230 »> s=clifford("1+3{-1}+2{1,2}+4{-2,7}").frame(); type(s)
01231 <class 'PyClical.index_set'>
01232 """
01233 return index_set().wrap(self.instance.frame())
01234
01235 def __repr__(self):
01236 """
01237 The "official" string representation of self.
01238
01239 »> clifford("1+3{-1}+2{1,2}+4{-2,7}").__repr__()
01240 'clifford("1+3{-1}+2{1,2}+4{-2,7}")'
01241 """
01242 return clifford_to_repr(self.unwrap()).decode()
01243
01244 def __str__(self):
01245 """
01246 The "informal" string representation of self.
01247
01248 »> clifford("1+3{-1}+2{1,2}+4{-2,7}").__str__()
01249 '1+3{-1}+2{1,2}+4{-2,7}'
01250 """
01251 return clifford_to_str(self.unwrap()).decode()
01252
01253 def clifford_hidden_doctests():
01254 """
01255 Tests for functions that Doctest cannot see.
01256
01257 For clifford.__cinit__: Construct an object of type clifford.
01258
01259 »> print(clifford(2))
01260 2
01261 »> print(clifford(2.0))
01262 2
01263 »> print(clifford(1.0e-1))

```

```

01264 0.1
01265 >> print(clifford("2"))
01266 2
01267 >> print(clifford("2{1,2,3}"))
01268 2{1,2,3}
01269 >> print(clifford(clifford("2{1,2,3}")))
01270 2{1,2,3}
01271 >> print(clifford("-{1}"))
01272 -{1}
01273 >> print(clifford(2,index_set({1,2})))
01274 2{1,2}
01275 >> print(clifford([2,3],index_set({1,2})))
01276 2{1}+3{2}
01277 >> print(clifford([1,2]))
01278 Traceback (most recent call last):
01279 ...
01280 TypeError: Cannot initialize clifford object from <class 'list'>.
01281 >> print(clifford(None))
01282 Traceback (most recent call last):
01283 ...
01284 TypeError: Cannot initialize clifford object from <class 'NoneType'>.
01285 >> print(clifford(None,[1,2]))
01286 Traceback (most recent call last):
01287 ...
01288 TypeError: Cannot initialize clifford object from (<class 'NoneType'>, <class 'list'>).
01289 >> print(clifford([1,2],[1,2]))
01290 Traceback (most recent call last):
01291 ...
01292 TypeError: Cannot initialize clifford object from (<class 'list'>, <class 'list'>).
01293 >> print(clifford(""))
01294 Traceback (most recent call last):
01295 ...
01296 ValueError: Cannot initialize clifford object from invalid string "".
01297 >> print(clifford("{}"))
01298 Traceback (most recent call last):
01299 ...
01300 ValueError: Cannot initialize clifford object from invalid string '{}'.
01301 >> print(clifford("{1}"))
01302 Traceback (most recent call last):
01303 ...
01304 ValueError: Cannot initialize clifford object from invalid string '{1}'.
01305 >> print(clifford("{}+"))
01306 Traceback (most recent call last):
01307 ...
01308 ValueError: Cannot initialize clifford object from invalid string '+'.
01309 >> print(clifford("{}-"))
01310 Traceback (most recent call last):
01311 ...
01312 ValueError: Cannot initialize clifford object from invalid string '-'.
01313 >> print(clifford("{1}+"))
01314 Traceback (most recent call last):
01315 ...
01316 ValueError: Cannot initialize clifford object from invalid string '{1}+'.
01317
01318 For clifford.__richcmp__: Compare objects of type clifford.
01319
01320 >> clifford("{1}") == clifford("1{1}")
01321 True
01322 >> clifford("{1}") != clifford("1.0{1}")
01323 False
01324 >> clifford("{1}") != clifford("1.0")
01325 True
01326 >> clifford("{1,2}") == None
01327 False
01328 >> clifford("{1,2}") != None
01329 True
01330 >> None == clifford("{1,2}")
01331 False
01332 >> None != clifford("{1,2}")
01333 True
01334 """
01335 return
01336
01337 cpdef inline error_squared_tol(obj):
01338 """
01339 Quadratic norm error tolerance relative to a specific multivector.
01340
01341 >> print(error_squared_tol(clifford("{1}")) * 3.0 - error_squared_tol(clifford("1{1}-2{2}+3{3}")))
01342 0.0
01343 """
01344 return glucat.error_squared_tol(toClifford(obj))
01345
01346 cpdef inline error_squared(lhs, rhs, threshold):
01347 """
01348 Relative or absolute error using the quadratic norm.
01349
01350 >> err2=scalar_epsilon*scalar_epsilon

```

```

01351
01352 >> print(error_squared(clifford("{1}"), clifford("1{1}"), err2))
01353 0.0
01354 >> print(error_squared(clifford("1{1}-3{2}+4{3}"), clifford("{1}"), err2))
01355 25.0
01356 """
01357 return glucat.error_squared(toClifford(lhs), toClifford(rhs), <scalar_t>threshold)
01358
01359 cpdef inline approx_equal(lhs, rhs, threshold=None, tol=None):
01360 """
01361 Test for approximate equality of multivectors.
01362
01363 >> err2=scalar_epsilon*scalar_epsilon
01364
01365 >> print(approx_equal(clifford("{1}"), clifford("1{1}")))
01366 True
01367 >> print(approx_equal(clifford("1{1}-3{2}+4{3}"), clifford("{1}")))
01368 False
01369 >> print(approx_equal(clifford("1{1}-3{2}+4{3}+0.001"), clifford("1{1}-3{2}+4{3}"), err2, err2))
01370 False
01371 >> print(approx_equal(clifford("1{1}-3{2}+4{3}+1.0e-30"), clifford("1{1}-3{2}+4{3}"), err2, err2))
01372 True
01373 """
01374 threshold = error_squared_tol(rhs) if threshold is None else threshold
01375 tol = error_squared_tol(rhs) if tol is None else tol
01376 return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold, <scalar_t>tol)
01377
01378 cpdef inline inv(obj):
01379 """
01380 Geometric multiplicative inverse.
01381
01382 >> print(inv(clifford("{1}")))
01383 {1}
01384 >> print(inv(clifford("{-1}")))
01385 -{-1}
01386 >> print(inv(clifford("{-2,-1}")))
01387 -{-2,-1}
01388 >> print(inv(clifford("{-1}+{1}")))
01389 nan
01390 """
01391 return clifford(obj).inv()
01392
01393 cpdef inline scalar(obj):
01394 """
01395 Scalar part.
01396
01397 >> scalar(clifford("1+{1}+{1,2}"))
01398 1.0
01399 >> scalar(clifford("{1,2}"))
01400 0.0
01401 """
01402 return clifford(obj).scalar()
01403
01404 cpdef inline real(obj):
01405 """
01406 Real part: synonym for scalar part.
01407
01408 >> real(clifford("1+{1}+{1,2}"))
01409 1.0
01410 >> real(clifford("{1,2}"))
01411 0.0
01412 """
01413 return clifford(obj).scalar()
01414
01415 cpdef inline imag(obj):
01416 """
01417 Imaginary part: deprecated (always 0).
01418
01419 >> imag(clifford("1+{1}+{1,2}"))
01420 0.0
01421 >> imag(clifford("{1,2}"))
01422 0.0
01423 """
01424 return 0.0
01425
01426 cpdef inline pure(obj):
01427 """
01428 Pure part
01429
01430 >> print(pure(clifford("1+{1}+{1,2}")))
01431 {1}+{1,2}
01432 >> print(pure(clifford("{1,2}")))
01433 {1,2}
01434 """
01435 return clifford(obj).pure()
01436
01437 cpdef inline even(obj):

```

```

01438 """
01439 Even part of multivector, sum of even grade terms.
01440
01441 >> print(even(clifford("1+{1}+{1,2}")))
01442 1+{1,2}
01443 """
01444 return clifford(obj).even()
01445
01446 cpdef inline odd(obj):
01447 """
01448 Odd part of multivector, sum of odd grade terms.
01449
01450 >> print(odd(clifford("1+{1}+{1,2}")))
01451 {1}
01452 """
01453 return clifford(obj).odd()
01454
01455 cpdef inline involute(obj):
01456 """
01457 Main involution, each {i} is replaced by -{i} in each term, eg. {1}*{2} -> (-{2})*(-{1})
01458
01459 >> print(involute(clifford("{1}")))
01460 -{1}
01461 >> print(involute(clifford("{2}") * clifford("{1}")))
01462 -{1,2}
01463 >> print(involute(clifford("{1}") * clifford("{2}")))
01464 {1,2}
01465 >> print(involute(clifford("1+{1}+{1,2}")))
01466 1-{1}+{1,2}
01467 """
01468 return clifford(obj).involute()
01469
01470 cpdef inline reverse(obj):
01471 """
01472 Reversion, eg. {1}*{2} -> {2}*{1}
01473
01474 >> print(reverse(clifford("{1}")))
01475 {1}
01476 >> print(reverse(clifford("{2}") * clifford("{1}")))
01477 {1,2}
01478 >> print(reverse(clifford("{1}") * clifford("{2}")))
01479 -{1,2}
01480 >> print(reverse(clifford("1+{1}+{1,2}")))
01481 1+{1}-{1,2}
01482 """
01483 return clifford(obj).reverse()
01484
01485 cpdef inline conj(obj):
01486 """
01487 Conjugation, reverse o involute == involute o reverse.
01488
01489 >> print(conj(clifford("{1}")))
01490 -{1}
01491 >> print(conj(clifford("{2}") * clifford("{1}")))
01492 {1,2}
01493 >> print(conj(clifford("{1}") * clifford("{2}")))
01494 -{1,2}
01495 >> print(conj(clifford("1+{1}+{1,2}")))
01496 1-{1}-{1,2}
01497 """
01498 return clifford(obj).conj()
01499
01500 cpdef inline quad(obj):
01501 """
01502 Quadratic form == (rev(x)*x)(0).
01503
01504 >> print(quad(clifford("1+{1}+{1,2}")))
01505 3.0
01506 >> print(quad(clifford("1+{-1}+{1,2}+{1,2,3}")))
01507 2.0
01508 """
01509 return clifford(obj).quad()
01510
01511 cpdef inline norm(obj):
01512 """
01513 norm == sum of squares of coordinates.
01514
01515 >> norm(clifford("1+{1}+{1,2}")))
01516 3.0
01517 >> norm(clifford("1+{-1}+{1,2}+{1,2,3}")))
01518 4.0
01519 """
01520 return clifford(obj).norm()
01521
01522 cpdef inline abs(obj):
01523 """
01524 Absolute value of multivector: multivector 2-norm.

```

```

01525
01526 >> abs(clifford("1+{-1}+{1,2}+{1,2,3}"))
01527 2.0
01528 """
01529 return glucat.abs(toClifford(obj))
01530
01531 cpdef inline max_abs(obj):
01532 """
01533 Maximum absolute value of coordinates multivector: multivector infinity-norm.
01534
01535 >> max_abs(clifford("1+{-1}+{1,2}+{1,2,3}"))
01536 1.0
01537 >> max_abs(clifford("3+2{1}+{1,2}"))
01538 3.0
01539 """
01540
01541 return glucat.max_abs(toClifford(obj))
01542
01543 cpdef inline pow(obj, m):
01544 """
01545 Integer power of multivector: obj to the m.
01546
01547 >> x=clifford("{1}"); print(pow(x,2))
01548 1
01549 >> x=clifford("2"); print(pow(x,2))
01550 4
01551 >> x=clifford("2+{1}"); print(pow(x,0))
01552 1
01553 >> x=clifford("2+{1}"); print(pow(x,1))
01554 2+{1}
01555 >> x=clifford("2+{1}"); print(pow(x,2))
01556 5+4{1}
01557 >> print(pow(clifford("1+{1}+{1,2}"),3))
01558 1+3{1}+3{1,2}
01559 >> i=clifford("{1,2}"); print(exp(pi/2) * pow(i, i))
01560 1
01561 """
01562 try:
01563 math.pow(obj, m)
01564 except:
01565 return clifford(obj).pow(m)
01566
01567 cpdef inline outer_pow(obj, m):
01568 """
01569 Outer product power of multivector.
01570
01571 >> print(outer_pow(clifford("1+{1}+{1,2}"),3))
01572 1+3{1}+3{1,2}
01573 """
01574 return clifford(obj).outer_pow(m)
01575
01576 cpdef inline complexifier(obj):
01577 """
01578 Square root of -1 which commutes with all members of the frame of the given multivector.
01579
01580 >> print(complexifier(clifford(index_set({1}))))
01581 {1,2,3}
01582 >> print(complexifier(clifford(index_set({-1}))))
01583 {-1}
01584 >> print(complexifier(index_set({1})))
01585 {1,2,3}
01586 >> print(complexifier(index_set({-1})))
01587 {-1}
01588 """
01589 return clifford().wrap(glucat.complexifier(toClifford(obj)))
01590
01591 cpdef inline sqrt(obj, i = None):
01592 """
01593 Square root of multivector with optional complexifier.
01594
01595 >> print(sqrt(-1))
01596 {-1}
01597 >> print(sqrt(clifford("2{-1}")))
01598 1+{-1}
01599 >> j=sqrt(-1,complexifier(index_set({1}))); print(j); print(j*j)
01600 {1,2,3}
01601 -1
01602 >> j=sqrt(-1,"{1,2,3}"); print(j); print(j*j)
01603 {1,2,3}
01604 -1
01605 """
01606 if not (i is None):
01607 return clifford().wrap(glucat.sqrt(toClifford(obj), toClifford(i)))
01608 else:
01609 try:
01610 return math.sqrt(obj)
01611 except:

```

```

01612 return clifford().wrap(glucat.sqrt(toClifford(obj)))
01613
01614 cpdef inline exp(obj):
01615 """
01616 Exponential of multivector.
01617
01618 >> x=clifford("{1,2}") * pi/4; print(exp(x))
01619 0.7071+0.7071{1,2}
01620 >> x=clifford("{1,2}") * pi/2; print(exp(x))
01621 {1,2}
01622 """
01623 try:
01624 return math.exp(obj)
01625 except:
01626 return clifford().wrap(glucat.exp(toClifford(obj)))
01627
01628 cpdef inline log(obj,i = None):
01629 """
01630 Natural logarithm of multivector with optional complexifier.
01631
01632 >> x=clifford("{-1}"); print((log(x,"{-1}") * 2/pi))
01633 {-1}
01634 >> x=clifford("{1,2}"); print((log(x,"{1,2,3}") * 2/pi))
01635 {1,2}
01636 >> x=clifford("{1,2}"); print((log(x) * 2/pi))
01637 {1,2}
01638 >> x=clifford("{1,2}"); print((log(x,"{1,2}") * 2/pi))
01639 Traceback (most recent call last):
01640 ...
01641 RuntimeError: check_complex(val, i): i is not a valid complexifier for val
01642 """
01643 if not (i is None):
01644 return clifford().wrap(glucat.log(toClifford(obj), toClifford(i)))
01645 else:
01646 try:
01647 return math.log(obj)
01648 except:
01649 return clifford().wrap(glucat.log(toClifford(obj)))
01650
01651 cpdef inline cos(obj,i = None):
01652 """
01653 Cosine of multivector with optional complexifier.
01654
01655 >> x=clifford("{1,2}"); print(cos(acos(x),"{1,2,3}"))
01656 {1,2}
01657 >> x=clifford("{1,2}"); print(cos(acos(x)))
01658 {1,2}
01659 """
01660 if not (i is None):
01661 return clifford().wrap(glucat.cos(toClifford(obj), toClifford(i)))
01662 else:
01663 try:
01664 return math.cos(obj)
01665 except:
01666 return clifford().wrap(glucat.cos(toClifford(obj)))
01667
01668 cpdef inline acos(obj,i = None):
01669 """
01670 Inverse cosine of multivector with optional complexifier.
01671
01672 >> x=clifford("{1,2}"); print(cos(acos(x),"{1,2,3}"))
01673 {1,2}
01674 >> x=clifford("{1,2}"); print(cos(acos(x),"{-1,1,2,3,4}"))
01675 {1,2}
01676 >> print(acos(0) / pi)
01677 0.5
01678 >> x=clifford("{1,2}"); print(cos(acos(x)))
01679 {1,2}
01680 """
01681 if not (i is None):
01682 return clifford().wrap(glucat.acos(toClifford(obj), toClifford(i)))
01683 else:
01684 try:
01685 return math.acos(obj)
01686 except:
01687 return clifford().wrap(glucat.acos(toClifford(obj)))
01688
01689 cpdef inline cosh(obj):
01690 """
01691 Hyperbolic cosine of multivector.
01692
01693 >> x=clifford("{1,2}") * pi; print(cosh(x))
01694 -1
01695 >> x=clifford("{1,2,3}"); print(cosh(acosh(x)))
01696 {1,2,3}
01697 >> x=clifford("{1,2}"); print(cosh(acosh(x)))
01698 {1,2}

```



```

01699 """
01700 try:
01701 return math.cosh(obj)
01702 except:
01703 return clifford().wrap(glucat.cosh(toClifford(obj)))
01704
01705 cpdef inline acosh(obj,i = None):
01706 """
01707 Inverse hyperbolic cosine of multivector with optional complexifier.
01708
01709 >> print(acosh(0,"{-2,-1,1}"))
01710 1.571{-2,-1,1}
01711 >> x=clifford("{1,2,3}"); print(cosh(acosh(x,"{-1,1,2,3,4}")))
01712 {1,2,3}
01713 >> print(acosh(0))
01714 1.571{-1}
01715 >> x=clifford("{1,2,3}"); print(cosh(acosh(x)))
01716 {1,2,3}
01717 >> x=clifford("{1,2}"); print(cosh(acosh(x)))
01718 {1,2}
01719 """
01720 if not (i is None):
01721 return clifford().wrap(glucat.acosh(toClifford(obj), toClifford(i)))
01722 else:
01723 try:
01724 return math.acosh(obj)
01725 except:
01726 return clifford().wrap(glucat.acosh(toClifford(obj)))
01727
01728 cpdef inline sin(obj,i = None):
01729 """
01730 Sine of multivector with optional complexifier.
01731
01732 >> s="{1}"; x=clifford(s); print(asin(sin(x,s),s))
01733 {-1}
01734 >> s="{1}"; x=clifford(s); print(asin(sin(x,s),"{-2,-1,1}"))
01735 {-1}
01736 >> x=clifford("{1,2,3}"); print(asin(sin(x)))
01737 {1,2,3}
01738 """
01739 if not (i is None):
01740 return clifford().wrap(glucat.sin(toClifford(obj), toClifford(i)))
01741 else:
01742 try:
01743 return math.sin(obj)
01744 except:
01745 return clifford().wrap(glucat.sin(toClifford(obj)))
01746
01747 cpdef inline asin(obj,i = None):
01748 """
01749 Inverse sine of multivector with optional complexifier.
01750
01751 >> s="{1}"; x=clifford(s); print(asin(sin(x,s),s))
01752 {-1}
01753 >> s="{1}"; x=clifford(s); print(asin(sin(x,s),"{-2,-1,1}"))
01754 {-1}
01755 >> print(asin(1) / pi)
01756 0.5
01757 >> x=clifford("{1,2,3}"); print(asin(sin(x)))
01758 {1,2,3}
01759 """
01760 if not (i is None):
01761 return clifford().wrap(glucat.asin(toClifford(obj), toClifford(i)))
01762 else:
01763 try:
01764 return math.asin(obj)
01765 except:
01766 return clifford().wrap(glucat.asin(toClifford(obj)))
01767
01768 cpdef inline sinh(obj):
01769 """
01770 Hyperbolic sine of multivector.
01771
01772 >> x=clifford("{1,2}") * pi/2; print(sinh(x))
01773 {1,2}
01774 >> x=clifford("{1,2}") * pi/6; print(sinh(x))
01775 0.5{1,2}
01776 """
01777 try:
01778 return math.sinh(obj)
01779 except:
01780 return clifford().wrap(glucat.sinh(toClifford(obj)))
01781
01782 cpdef inline asinh(obj,i = None):
01783 """
01784 Inverse hyperbolic sine of multivector with optional complexifier.
01785

```

```

01786 >> x=clifford("{1,2}"); print(asinh(x,"{1,2,3}") * 2/pi)
01787 {1,2}
01788 >> x=clifford("{1,2}"); print(asinh(x) * 2/pi)
01789 {1,2}
01790 >> x=clifford("{1,2}") / 2; print(asinh(x) * 6/pi)
01791 {1,2}
01792 """
01793 if not (i is None):
01794 return clifford().wrap(glucat.asinh(toClifford(obj), toClifford(i)))
01795 else:
01796 try:
01797 return math.asinh(obj)
01798 except:
01799 return clifford().wrap(glucat.asinh(toClifford(obj)))
01800
01801 cpdef inline tan(obj,i = None):
01802 """
01803 Tangent of multivector with optional complexifier.
01804
01805 >> x=clifford("{1,2}"); print(tan(x,"{1,2,3}"))
01806 0.7616{1,2}
01807 >> x=clifford("{1,2}"); print(tan(x))
01808 0.7616{1,2}
01809 """
01810 if not (i is None):
01811 return clifford().wrap(glucat.tan(toClifford(obj), toClifford(i)))
01812 else:
01813 try:
01814 return math.tan(obj)
01815 except:
01816 return clifford().wrap(glucat.tan(toClifford(obj)))
01817
01818 cpdef inline atan(obj,i = None):
01819 """
01820 Inverse tangent of multivector with optional complexifier.
01821
01822 >> s=index_set({1,2,3}); x=clifford("{1}"); print(tan(atan(x,s),s))
01823 {1}
01824 >> x=clifford("{1}"); print(tan(atan(x)))
01825 {1}
01826 """
01827 if not (i is None):
01828 return clifford().wrap(glucat.atan(toClifford(obj), toClifford(i)))
01829 else:
01830 try:
01831 return math.atan(obj)
01832 except:
01833 return clifford().wrap(glucat.atan(toClifford(obj)))
01834
01835 cpdef inline tanh(obj):
01836 """
01837 Hyperbolic tangent of multivector.
01838
01839 >> x=clifford("{1,2}") * pi/4; print(tanh(x))
01840 {1,2}
01841 """
01842 try:
01843 return math.tanh(obj)
01844 except:
01845 return clifford().wrap(glucat.tanh(toClifford(obj)))
01846
01847 cpdef inline atanh(obj,i = None):
01848 """
01849 Inverse hyperbolic tangent of multivector with optional complexifier.
01850
01851 >> s=index_set({1,2,3}); x=clifford("{1,2}"); print(tanh(atanh(x,s)))
01852 {1,2}
01853 >> x=clifford("{1,2}"); print(tanh(atanh(x)))
01854 {1,2}
01855 """
01856 if not (i is None):
01857 return clifford().wrap(glucat.atanh(toClifford(obj), toClifford(i)))
01858 else:
01859 try:
01860 return math.atanh(obj)
01861 except:
01862 return clifford().wrap(glucat.atanh(toClifford(obj)))
01863
01864 cpdef inline random_clifford(index_set ixt, fill = 1.0):
01865 """
01866 Random multivector within a frame.
01867
01868 >> print(random_clifford(index_set({-3,-1,2})).frame())
01869 {-3,-1,2}
01870 """
01871 return clifford().wrap(clifford().instance.random(ixt.unwrap(), <scalar_t>fill))
01872

```

```

01873 cpdef inline cga3(obj):
01874 """
01875 Convert Euclidean 3D multivector to Conformal Geometric Algebra using Doran and Lasenby
 definition.
01876
01877 >> x=clifford("2{1}+9{2}+{3}"); print(cga3(x))
01878 87{-1}+4{1}+18{2}+2{3}+85{4}
01879 """
01880 return clifford().wrap(glucat.cga3(toClifford(obj)))
01881
01882 cpdef inline cga3std(obj):
01883 """
01884 Convert CGA3 null vector to standard conformal null vector using Doran and Lasenby definition.
01885
01886 >> x=clifford("2{1}+9{2}+{3}"); print(cga3std(cga3(x)))
01887 87{-1}+4{1}+18{2}+2{3}+85{4}
01888 >> x=clifford("2{1}+9{2}+{3}"); print(cga3std(cga3(x))-cga3(x))
01889 0
01890 """
01891 return clifford().wrap(glucat.cga3std(toClifford(obj)))
01892
01893 cpdef inline agc3(obj):
01894 """
01895 Convert CGA3 null vector to Euclidean 3D vector using Doran and Lasenby definition.
01896
01897 >> x=clifford("2{1}+9{2}+{3}"); print(agc3(cga3(x)))
01898 2{1}+9{2}+{3}
01899 >> x=clifford("2{1}+9{2}+{3}"); print(agc3(cga3(x))-x)
01900 0
01901 """
01902 return clifford().wrap(glucat.agc3(toClifford(obj)))
01903
01904 # Some abbreviations.
01905 scalar_epsilon = epsilon
01906
01907 pi = atan(clifford(1.0)) * 4.0
01908 tau = atan(clifford(1.0)) * 8.0
01909
01910 c1 = clifford
01911 """
01912 Abbreviation for clifford.
01913
01914 >> print(c1(2))
01915 2
01916 >> print(c1(2.0))
01917 2
01918 >> print(c1(5.0e-1))
01919 0.5
01920 >> print(c1("2"))
01921 2
01922 >> print(c1("2{1,2,3}"))
01923 2{1,2,3}
01924 >> print(c1(c1("2{1,2,3}")))
01925 2{1,2,3}
01926 """
01927
01928 ist = index_set
01929 """
01930 Abbreviation for index_set.
01931
01932 >> print(ist("{1,2,3}"))
01933 {1,2,3}
01934 """
01935
01936 def e(obj):
01937 """
01938 Abbreviation for clifford(index_set(obj)).
01939
01940 >> print(e(1))
01941 {1}
01942 >> print(e(-1))
01943 {-1}
01944 >> print(e(0))
01945 1
01946 """
01947 return clifford(index_set(obj))
01948
01949 def istpq(p, q):
01950 """
01951 Abbreviation for index_set({-q,...p}).
01952
01953 >> print(istpq(2,3))
01954 {-3,-2,-1,1,2}
01955 """
01956 return index_set(set(range(-q,p+1)))
01957
01958 ninf3 = e(4) + e(-1) # Null infinity point in 3D Conformal Geometric Algebra [DL].

```

```

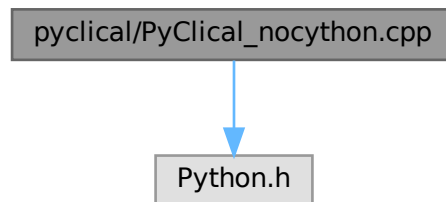
01959 nbar3 = e(4) - e(-1) # Null bar point in 3D Conformal Geometric Algebra [DL].
01960
01961 # Doctest interface.
01962 def _test():
01963 import PyClical, doctest
01964 return doctest.testmod(PyClical)
01965
01966 if __name__ == "__main__":
01967 _test()

```

## 7.63 pyclical/PyClical\_nocython.cpp File Reference

```
#include "Python.h"
```

Include dependency graph for PyClical\_nocython.cpp:



### Macros

- `#define PY_SSIZE_T_CLEAN`

### 7.63.1 Macro Definition Documentation

#### 7.63.1.1 PY\_SSIZE\_T\_CLEAN

```
#define PY_SSIZE_T_CLEAN
```

Definition at line 23 of file [PyClical\\_nocython.cpp](#).

## 7.64 PyClical\_nocython.cpp

[Go to the documentation of this file.](#)

```

00001 /* Generated by Cython 0.29.28 */
00002
00003 /* BEGIN: Cython Metadata
00004 {
00005 "distutils": {
00006 "depends": [
00007 "PyClical.h"
00008],
00009 "include_dirs": [
00010 "."
00011],

```

```

00012 "language": "c++",
00013 "name": "PyClical",
00014 "sources": [
00015 "PyClical.pyx"
00016]
00017 },
00018 "module_name": "PyClical"
00019 }
00020 END: Cython Metadata */
00021
00022 #ifndef PY_SSIZE_T_CLEAN
00023 #define PY_SSIZE_T_CLEAN
00024 #endif /* PY_SSIZE_T_CLEAN */
00025 #include "Python.h"
00026 #ifndef Py_PYTHON_H
00027 #error Python headers needed to compile C extensions, please install development version of
00028 Python.
00029 #elif PY_VERSION_HEX < 0x02060000 || (0x03000000 <= PY_VERSION_HEX && PY_VERSION_HEX < 0x03030000)
00029 #error Cython requires Python 2.6+ or Python 3.3+.
00030 #else
00031 #define CYTHON_ABI "0_29_28"
00032 #define CYTHON_HEX_VERSION 0x001D1CF0
00033 #define CYTHON_FUTURE_DIVISION 1
00034 #include <stddef.h>
00035 #ifndef offsetof
00036 #define offsetof(type, member) ((size_t) & ((type*)0) -> member)
00037 #endif
00038 #if !defined(WIN32) && !defined(MS_WINDOWS)
00039 #ifndef __stdcall
00040 #define __stdcall
00041 #endif
00042 #ifndef __cdecl
00043 #define __cdecl
00044 #endif
00045 #ifndef __fastcall
00046 #define __fastcall
00047 #endif
00048 #endif
00049 #ifndef DL_IMPORT
00050 #define DL_IMPORT(t) t
00051 #endif
00052 #ifndef DL_EXPORT
00053 #define DL_EXPORT(t) t
00054 #endif
00055 #define __PYX_COMMA ,
00056 #ifndef HAVE_LONG_LONG
00057 #if PY_VERSION_HEX >= 0x02070000
00058 #define HAVE_LONG_LONG
00059 #endif
00060 #endif
00061 #ifndef PY_LONG_LONG
00062 #define PY_LONG_LONG LONG_LONG
00063 #endif
00064 #ifndef Py_HUGE_VAL
00065 #define Py_HUGE_VAL HUGE_VAL
00066 #endif
00067 #ifdef PYPY_VERSION
00068 #define CYTHON_COMPILING_IN_PYPY 1
00069 #define CYTHON_COMPILING_IN_PYSTON 0
00070 #define CYTHON_COMPILING_IN_CPYTHON 0
00071 #undef CYTHON_USE_TYPE_SLOTS
00072 #define CYTHON_USE_TYPE_SLOTS 0
00073 #undef CYTHON_USE_PYTYPE_LOOKUP
00074 #define CYTHON_USE_PYTYPE_LOOKUP 0
00075 #if PY_VERSION_HEX < 0x03050000
00076 #undef CYTHON_USE_ASYNC_SLOTS
00077 #define CYTHON_USE_ASYNC_SLOTS 0
00078 #elif !defined(CYTHON_USE_ASYNC_SLOTS)
00079 #define CYTHON_USE_ASYNC_SLOTS 1
00080 #endif
00081 #undef CYTHON_USE_PYLIST_INTERNALS
00082 #define CYTHON_USE_PYLIST_INTERNALS 0
00083 #undef CYTHON_USE_UNICODE_INTERNALS
00084 #define CYTHON_USE_UNICODE_INTERNALS 0
00085 #undef CYTHON_USE_UNICODE_WRITER
00086 #define CYTHON_USE_UNICODE_WRITER 0
00087 #undef CYTHON_USE_PYLONG_INTERNALS
00088 #define CYTHON_USE_PYLONG_INTERNALS 0
00089 #undef CYTHON_AVOID_BORROWED_REFS
00090 #define CYTHON_AVOID_BORROWED_REFS 1
00091 #undef CYTHON_ASSUME_SAFE_MACROS
00092 #define CYTHON_ASSUME_SAFE_MACROS 0
00093 #undef CYTHON_UNPACK_METHODS
00094 #define CYTHON_UNPACK_METHODS 0
00095 #undef CYTHON_FAST_THREAD_STATE
00096 #define CYTHON_FAST_THREAD_STATE 0
00097 #undef CYTHON_FAST_PYCALL

```

```
00098 #define CYTHON_FAST_PYCALL 0
00099 #undef CYTHON_PEP489_MULTI_PHASE_INIT
00100 #define CYTHON_PEP489_MULTI_PHASE_INIT 0
00101 #undef CYTHON_USE_TP_FINALIZE
00102 #define CYTHON_USE_TP_FINALIZE 0
00103 #undef CYTHON_USE_DICT_VERSIONS
00104 #define CYTHON_USE_DICT_VERSIONS 0
00105 #undef CYTHON_USE_EXC_INFO_STACK
00106 #define CYTHON_USE_EXC_INFO_STACK 0
00107 #elif defined(PYSTON_VERSION)
00108 #define CYTHON_COMPILING_IN_PYPY 0
00109 #define CYTHON_COMPILING_IN_PYSTON 1
00110 #define CYTHON_COMPILING_IN_CPYTHON 0
00111 #ifndef CYTHON_USE_TYPE_SLOTS
00112 #define CYTHON_USE_TYPE_SLOTS 1
00113 #endif
00114 #undef CYTHON_USE_PYTYPE_LOOKUP
00115 #define CYTHON_USE_PYTYPE_LOOKUP 0
00116 #undef CYTHON_USE_ASYNC_SLOTS
00117 #define CYTHON_USE_ASYNC_SLOTS 0
00118 #undef CYTHON_USE_PYLIST_INTERNALS
00119 #define CYTHON_USE_PYLIST_INTERNALS 0
00120 #ifndef CYTHON_USE_UNICODE_INTERNALS
00121 #define CYTHON_USE_UNICODE_INTERNALS 1
00122 #endif
00123 #undef CYTHON_USE_UNICODE_WRITER
00124 #define CYTHON_USE_UNICODE_WRITER 0
00125 #undef CYTHON_USE_PYLONG_INTERNALS
00126 #define CYTHON_USE_PYLONG_INTERNALS 0
00127 #ifndef CYTHON_AVOID_BORROWED_REFS
00128 #define CYTHON_AVOID_BORROWED_REFS 0
00129 #endif
00130 #ifndef CYTHON_ASSUME_SAFE_MACROS
00131 #define CYTHON_ASSUME_SAFE_MACROS 1
00132 #endif
00133 #ifndef CYTHON_UNPACK_METHODS
00134 #define CYTHON_UNPACK_METHODS 1
00135 #endif
00136 #undef CYTHON_FAST_THREAD_STATE
00137 #define CYTHON_FAST_THREAD_STATE 0
00138 #undef CYTHON_FAST_PYCALL
00139 #define CYTHON_FAST_PYCALL 0
00140 #undef CYTHON_PEP489_MULTI_PHASE_INIT
00141 #define CYTHON_PEP489_MULTI_PHASE_INIT 0
00142 #undef CYTHON_USE_TP_FINALIZE
00143 #define CYTHON_USE_TP_FINALIZE 0
00144 #undef CYTHON_USE_DICT_VERSIONS
00145 #define CYTHON_USE_DICT_VERSIONS 0
00146 #undef CYTHON_USE_EXC_INFO_STACK
00147 #define CYTHON_USE_EXC_INFO_STACK 0
00148 #else
00149 #define CYTHON_COMPILING_IN_PYPY 0
00150 #define CYTHON_COMPILING_IN_PYSTON 0
00151 #define CYTHON_COMPILING_IN_CPYTHON 1
00152 #ifndef CYTHON_USE_TYPE_SLOTS
00153 #define CYTHON_USE_TYPE_SLOTS 1
00154 #endif
00155 #if PY_VERSION_HEX < 0x02070000
00156 #undef CYTHON_USE_PYTYPE_LOOKUP
00157 #define CYTHON_USE_PYTYPE_LOOKUP 0
00158 #elif !defined(CYTHON_USE_PYTYPE_LOOKUP)
00159 #define CYTHON_USE_PYTYPE_LOOKUP 1
00160 #endif
00161 #if PY_MAJOR_VERSION < 3
00162 #undef CYTHON_USE_ASYNC_SLOTS
00163 #define CYTHON_USE_ASYNC_SLOTS 0
00164 #elif !defined(CYTHON_USE_ASYNC_SLOTS)
00165 #define CYTHON_USE_ASYNC_SLOTS 1
00166 #endif
00167 #if PY_VERSION_HEX < 0x02070000
00168 #undef CYTHON_USE_PYLONG_INTERNALS
00169 #define CYTHON_USE_PYLONG_INTERNALS 0
00170 #elif !defined(CYTHON_USE_PYLONG_INTERNALS)
00171 #define CYTHON_USE_PYLONG_INTERNALS 1
00172 #endif
00173 #ifndef CYTHON_USE_PYLIST_INTERNALS
00174 #define CYTHON_USE_PYLIST_INTERNALS 1
00175 #endif
00176 #ifndef CYTHON_USE_UNICODE_INTERNALS
00177 #define CYTHON_USE_UNICODE_INTERNALS 1
00178 #endif
00179 #if PY_VERSION_HEX < 0x030300F0 || PY_VERSION_HEX >= 0x030B00A2
00180 #undef CYTHON_USE_UNICODE_WRITER
00181 #define CYTHON_USE_UNICODE_WRITER 0
00182 #elif !defined(CYTHON_USE_UNICODE_WRITER)
00183 #define CYTHON_USE_UNICODE_WRITER 1
00184 #endif
```

```

00185 #ifndef CYTHON_AVOID_BORROWED_REFS
00186 #define CYTHON_AVOID_BORROWED_REFS 0
00187 #endif
00188 #ifndef CYTHON_ASSUME_SAFE_MACROS
00189 #define CYTHON_ASSUME_SAFE_MACROS 1
00190 #endif
00191 #ifndef CYTHON_UNPACK_METHODS
00192 #define CYTHON_UNPACK_METHODS 1
00193 #endif
00194 #if PY_VERSION_HEX >= 0x030B00A4
00195 #undef CYTHON_FAST_THREAD_STATE
00196 #define CYTHON_FAST_THREAD_STATE 0
00197 #elif !defined(CYTHON_FAST_THREAD_STATE)
00198 #define CYTHON_FAST_THREAD_STATE 1
00199 #endif
00200 #ifndef CYTHON_FAST_PYCALL
00201 #define CYTHON_FAST_PYCALL (PY_VERSION_HEX < 0x030B00A1)
00202 #endif
00203 #ifndef CYTHON_PEP489_MULTI_PHASE_INIT
00204 #define CYTHON_PEP489_MULTI_PHASE_INIT (PY_VERSION_HEX >= 0x03050000)
00205 #endif
00206 #ifndef CYTHON_USE_TP_FINALIZE
00207 #define CYTHON_USE_TP_FINALIZE (PY_VERSION_HEX >= 0x030400a1)
00208 #endif
00209 #ifndef CYTHON_USE_DICT_VERSIONS
00210 #define CYTHON_USE_DICT_VERSIONS (PY_VERSION_HEX >= 0x030600B1)
00211 #endif
00212 #if PY_VERSION_HEX >= 0x030B00A4
00213 #undef CYTHON_USE_EXC_INFO_STACK
00214 #define CYTHON_USE_EXC_INFO_STACK 0
00215 #elif !defined(CYTHON_USE_EXC_INFO_STACK)
00216 #define CYTHON_USE_EXC_INFO_STACK (PY_VERSION_HEX >= 0x030700A3)
00217 #endif
00218 #endif
00219 #if !defined(CYTHON_FAST_PYCCALL)
00220 #define CYTHON_FAST_PYCCALL (CYTHON_FAST_PYCALL && PY_VERSION_HEX >= 0x030600B1)
00221 #endif
00222 #if CYTHON_USE_PYLONG_INTERNALS
00223 #if PY_MAJOR_VERSION < 3
00224 #include "longintrepr.h"
00225 #endif
00226 #undef SHIFT
00227 #undef BASE
00228 #undef MASK
00229 #ifdef SIZEOF_VOID_P
00230 enum { __pyx_check_sizeof_voidp = 1 / (int) (sizeof(void*) == sizeof(void*)) };
00231 #endif
00232 #endif
00233 #ifndef __has_attribute
00234 #define __has_attribute(x) 0
00235 #endif
00236 #ifndef __has_cpp_attribute
00237 #define __has_cpp_attribute(x) 0
00238 #endif
00239 #ifndef CYTHON_RESTRICT
00240 #if defined(__GNUC__)
00241 #define CYTHON_RESTRICT __restrict__
00242 #elif defined(_MSC_VER) && _MSC_VER >= 1400
00243 #define CYTHON_RESTRICT __restrict
00244 #elif defined(__STDC_VERSION__) && __STDC_VERSION__ >= 199901L
00245 #define CYTHON_RESTRICT restrict
00246 #else
00247 #define CYTHON_RESTRICT
00248 #endif
00249 #endif
00250 #ifndef CYTHON_UNUSED
00251 #if defined(__GNUC__)
00252 # if !defined(__cplusplus) || (__GNUC__ > 3 || (__GNUC__ == 3 && __GNUC_MINOR__ >= 4))
00253 # define CYTHON_UNUSED __attribute__((unused))
00254 # else
00255 # define CYTHON_UNUSED
00256 # endif
00257 # elif defined(__ICC) || (defined(__INTEL_COMPILER) && !defined(_MSC_VER))
00258 # define CYTHON_UNUSED __attribute__((unused))
00259 # else
00260 # define CYTHON_UNUSED
00261 # endif
00262 #endif
00263 #ifndef CYTHON_MAYBE_UNUSED_VAR
00264 # if defined(__cplusplus)
00265 template<class T> void CYTHON_MAYBE_UNUSED_VAR(const T&) { }
00266 # else
00267 # define CYTHON_MAYBE_UNUSED_VAR(x) (void)(x)
00268 # endif
00269 #endif
00270 #ifndef CYTHON_NCP_UNUSED
00271 # if CYTHON_COMPILING_IN_CPYTHON

```

```

00272 # define CYTHON_NCP_UNUSED
00273 # else
00274 # define CYTHON_NCP_UNUSED CYTHON_UNUSED
00275 # endif
00276 #endif
00277 #define __Pyx_void_to_None(void_result) ((void)(void_result), Py_INCREF(Py_None), Py_None)
00278 #ifdef _MSC_VER
00279 #ifndef _MSC_STDINT_H_
00280 #if _MSC_VER < 1300
00281 typedef unsigned char uint8_t;
00282 typedef unsigned int uint32_t;
00283 #else
00284 typedef unsigned __int8 uint8_t;
00285 typedef unsigned __int32 uint32_t;
00286 #endif
00287 #endif
00288 #else
00289 #include <stdint.h>
00290 #endif
00291 #ifndef CYTHON_FALLTHROUGH
00292 #if defined(__cplusplus) && __cplusplus >= 201103L
00293 #if __has_cpp_attribute(fallthrough)
00294 #define CYTHON_FALLTHROUGH [[fallthrough]]
00295 #elif __has_cpp_attribute(clang::fallthrough)
00296 #define CYTHON_FALLTHROUGH [[clang::fallthrough]]
00297 #elif __has_cpp_attribute(gnu::fallthrough)
00298 #define CYTHON_FALLTHROUGH [[gnu::fallthrough]]
00299 #endif
00300 #endif
00301 #ifndef CYTHON_FALLTHROUGH
00302 #if __has_attribute(fallthrough)
00303 #define CYTHON_FALLTHROUGH __attribute__((fallthrough))
00304 #else
00305 #define CYTHON_FALLTHROUGH
00306 #endif
00307 #endif
00308 #if defined(__clang__) && defined(__apple_build_version__)
00309 #if __apple_build_version__ < 7000000
00310 #undef CYTHON_FALLTHROUGH
00311 #define CYTHON_FALLTHROUGH
00312 #endif
00313 #endif
00314 #endif
00315
00316 #ifndef __cplusplus
00317 #error "Cython files generated with the C++ option must be compiled with a C++ compiler."
00318 #endif
00319 #ifndef CYTHON_INLINE
00320 #if defined(__clang__)
00321 #define CYTHON_INLINE __inline__ __attribute__((unused))
00322 #else
00323 #define CYTHON_INLINE inline
00324 #endif
00325 #endif
00326 template<typename T>
00327 void __Pyx_call_destructor(T& x) {
00328 x.~T();
00329 }
00330 template<typename T>
00331 class __Pyx_FakeReference {
00332 public:
00333 __Pyx_FakeReference() : ptr(NULL) { }
00334 __Pyx_FakeReference(const T& ref) : ptr(const_cast<T*>(&ref)) { }
00335 T *operator->() { return ptr; }
00336 T *operator&() { return ptr; }
00337 operator T&() { return *ptr; }
00338 template<typename U> bool operator ==(U other) { return *ptr == other; }
00339 template<typename U> bool operator !=(U other) { return *ptr != other; }
00340 private:
00341 T *ptr;
00342 };
00343
00344 #if CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x02070600 && !defined(Py_OptimizeFlag)
00345 #define Py_OptimizeFlag 0
00346 #endif
00347 #define __PYX_BUILD_PY_SSIZE_T "n"
00348 #define CYTHON_FORMAT_SSIZE_T "z"
00349 #if PY_MAJOR_VERSION < 3
00350 #define __Pyx_BUILTIN_MODULE_NAME "__builtin__"
00351 #define __Pyx_PyCode_New(a, k, l, s, f, code, c, n, v, fv, cell, fn, name, fline, lnos)\
00352 PyCode_New(a+k, l, s, f, code, c, n, v, fv, cell, fn, name, fline, lnos)
00353 #define __Pyx_DefaultClassType PyClass_Type
00354 #else
00355 #define __Pyx_BUILTIN_MODULE_NAME "builtins"
00356 #define __Pyx_DefaultClassType PyType_Type
00357 #if PY_VERSION_HEX >= 0x030B00A1
00358 static CYTHON_INLINE PyCodeObject* __Pyx_PyCode_New(int a, int k, int l, int s, int f,

```



```

00359 PyObject *code, PyObject *c, PyObject* n, PyObject
00360 *v,
00361 PyObject *fv, PyObject *cell, PyObject* fn,
00362 PyObject *name, int fline, PyObject *lnos) {
00363 PyObject *kwds=NULL, *argcount=NULL, *posonlyargcount=NULL, *kwnonlyargcount=NULL;
00364 PyObject *nlocals=NULL, *stacksize=NULL, *flags=NULL, *replace=NULL, *call_result=NULL,
00365 *empty=NULL;
00366 const char *fn_cstr=NULL;
00367 const char *name_cstr=NULL;
00368 PyCodeObject* co=NULL;
00369 PyObject *type, *value, *traceback;
00370 PyErr_Fetch(&type, &value, &traceback);
00371 if (!(kwds=PyDict_New())) goto end;
00372 if (!(argcount=PyLong_FromLong(a))) goto end;
00373 if (PyDict_SetItemString(kwds, "co_argcount", argcount) != 0) goto end;
00374 if (!(posonlyargcount=PyLong_FromLong(0))) goto end;
00375 if (PyDict_SetItemString(kwds, "co_posonlyargcount", posonlyargcount) != 0) goto end;
00376 if (!(kwnonlyargcount=PyLong_FromLong(k))) goto end;
00377 if (PyDict_SetItemString(kwds, "co_kwnonlyargcount", kwnonlyargcount) != 0) goto end;
00378 if (!(nlocals=PyLong_FromLong(l))) goto end;
00379 if (PyDict_SetItemString(kwds, "co_nlocals", nlocals) != 0) goto end;
00380 if (!(stacksize=PyLong_FromLong(s))) goto end;
00381 if (PyDict_SetItemString(kwds, "co_stacksize", stacksize) != 0) goto end;
00382 if (!(flags=PyLong_FromLong(f))) goto end;
00383 if (PyDict_SetItemString(kwds, "co_flags", flags) != 0) goto end;
00384 if (PyDict_SetItemString(kwds, "co_code", code) != 0) goto end;
00385 if (PyDict_SetItemString(kwds, "co_consts", c) != 0) goto end;
00386 if (PyDict_SetItemString(kwds, "co_names", n) != 0) goto end;
00387 if (PyDict_SetItemString(kwds, "co_varnames", v) != 0) goto end;
00388 if (PyDict_SetItemString(kwds, "co_freevars", fv) != 0) goto end;
00389 if (PyDict_SetItemString(kwds, "co_cellvars", cell) != 0) goto end;
00390 if (PyDict_SetItemString(kwds, "co_linetable", lnos) != 0) goto end;
00391 if (!(fn_cstr=PyUnicode_AsUTF8AndSize(fn, NULL))) goto end;
00392 if (!(name_cstr=PyUnicode_AsUTF8AndSize(name, NULL))) goto end;
00393 if (!(co = PyCode_NewEmpty(fn_cstr, name_cstr, fline))) goto end;
00394 if (!(replace = PyObject_GetAttrString((PyObject*)co, "replace"))) goto cleanup_code_too;
00395 if (!(empty = PyTuple_New(0))) goto cleanup_code_too; // unfortunately __pyx_empty_tuple isn't
00396 available here
00397 if (!(call_result = PyObject_Call(replace, empty, kwds))) goto cleanup_code_too;
00398 Py_XDECREF((PyObject*)co);
00399 co = (PyCodeObject*)call_result;
00400 call_result = NULL;
00401 if (0) {
00402 cleanup_code_too:
00403 Py_XDECREF((PyObject*)co);
00404 co = NULL;
00405 }
00406 end:
00407 Py_XDECREF(kwds);
00408 Py_XDECREF(argcount);
00409 Py_XDECREF(posonlyargcount);
00410 Py_XDECREF(kwnonlyargcount);
00411 Py_XDECREF(nlocals);
00412 Py_XDECREF(stacksize);
00413 Py_XDECREF(replace);
00414 Py_XDECREF(call_result);
00415 Py_XDECREF(empty);
00416 if (type) {
00417 PyErr_Restore(type, value, traceback);
00418 }
00419 return co;
00420 }
00421 #else
00422 #define __Pyx_PyCode_New(a, k, l, s, f, code, c, n, v, fv, cell, fn, name, fline, lnos)\
00423 PyCode_New(a, k, l, s, f, code, c, n, v, fv, cell, fn, name, fline, lnos)
00424 #endif
00425 #define __Pyx_DefaultClassType PyType_Type
00426 #endif
00427 #ifndef Py_TPFLAGS_CHECKTYPES
00428 #define Py_TPFLAGS_CHECKTYPES 0
00429 #endif
00430 #ifndef Py_TPFLAGS_HAVE_INDEX
00431 #define Py_TPFLAGS_HAVE_INDEX 0
00432 #endif
00433 #ifndef Py_TPFLAGS_HAVE_NEWBUFFER
00434 #define Py_TPFLAGS_HAVE_NEWBUFFER 0
00435 #endif
00436 #ifndef Py_TPFLAGS_HAVE_FINALIZE
00437 #define Py_TPFLAGS_HAVE_FINALIZE 0
00438 #endif
00439 #ifndef METH_STACKLESS
00440 #define METH_STACKLESS 0
00441 #endif
00442 #if PY_VERSION_HEX <= 0x030700A3 || !defined(METH_FASTCALL)
00443 #ifndef METH_FASTCALL
00444 #define METH_FASTCALL 0x80
00445 #endif
00446 #endif

```

```

00443 typedef PyObject *(*__Pyx_PyCFunctionFast) (PyObject *self, PyObject *const *args, Py_ssize_t
nargs);
00444 typedef PyObject *(*__Pyx_PyCFunctionFastWithKeywords) (PyObject *self, PyObject *const *args,
00445 Py_ssize_t nargs, PyObject *kwnames);
00446 #else
00447 #define __Pyx_PyCFunctionFast _PyCFunctionFast
00448 #define __Pyx_PyCFunctionFastWithKeywords _PyCFunctionFastWithKeywords
00449 #endif
00450 #if CYTHON_FAST_PYCALL
00451 #define __Pyx_PyFastCFunction_Check(func)\
00452 ((PyCFunction_Check(func) && (METH_FASTCALL == (PyCFunction_GET_FLAGS(func) & ~(METH_CLASS |
METH_STATIC | METH_COEXIST | METH_KEYWORDS | METH_STACKLESS))))))
00453 #else
00454 #define __Pyx_PyFastCFunction_Check(func) 0
00455 #endif
00456 #if CYTHON_COMPILING_IN_PYPY && !defined(PyObject_Malloc)
00457 #define PyObject_Malloc(s) PyMem_Malloc(s)
00458 #define PyObject_Free(p) PyMem_Free(p)
00459 #define PyObject_Realloc(p) PyMem_Realloc(p)
00460 #endif
00461 #if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX < 0x030400A1
00462 #define PyMem_RawMalloc(n) PyMem_Malloc(n)
00463 #define PyMem_RawRealloc(p, n) PyMem_Realloc(p, n)
00464 #define PyMem_RawFree(p) PyMem_Free(p)
00465 #endif
00466 #if CYTHON_COMPILING_IN_PYSTON
00467 #define __Pyx_PyCode_HasFreeVars(co) PyCode_HasFreeVars(co)
00468 #define __Pyx_PyFrame_SetLineNumber(frame, lineno) PyFrame_SetLineNumber(frame, lineno)
00469 #else
00470 #define __Pyx_PyCode_HasFreeVars(co) (PyCode_GetNumFree(co) > 0)
00471 #define __Pyx_PyFrame_SetLineNumber(frame, lineno) (frame)->f_lineno = (lineno)
00472 #endif
00473 #if !CYTHON_FAST_THREAD_STATE || PY_VERSION_HEX < 0x02070000
00474 #define __Pyx_PyThreadState_Current PyThreadState_GET()
00475 #elif PY_VERSION_HEX >= 0x03060000
00476 #define __Pyx_PyThreadState_Current _PyThreadState_UncheckedGet()
00477 #elif PY_VERSION_HEX >= 0x03000000
00478 #define __Pyx_PyThreadState_Current PyThreadState_GET()
00479 #else
00480 #define __Pyx_PyThreadState_Current _PyThreadState_Current
00481 #endif
00482 #if PY_VERSION_HEX < 0x030700A2 && !defined(PyThread_tss_create) && !defined(Py_tss_NEEDS_INIT)
00483 #include "pythread.h"
00484 #define Py_tss_NEEDS_INIT 0
00485 typedef int Py_tss_t;
00486 static CYTHON_INLINE int PyThread_tss_create(Py_tss_t *key) {
00487 *key = PyThread_create_key();
00488 return 0;
00489 }
00490 static CYTHON_INLINE Py_tss_t * PyThread_tss_alloc(void) {
00491 Py_tss_t *key = (Py_tss_t *)PyObject_Malloc(sizeof(Py_tss_t));
00492 *key = Py_tss_NEEDS_INIT;
00493 return key;
00494 }
00495 static CYTHON_INLINE void PyThread_tss_free(Py_tss_t *key) {
00496 PyObject_Free(key);
00497 }
00498 static CYTHON_INLINE int PyThread_tss_is_created(Py_tss_t *key) {
00499 return *key != Py_tss_NEEDS_INIT;
00500 }
00501 static CYTHON_INLINE void PyThread_tss_delete(Py_tss_t *key) {
00502 PyThread_delete_key(*key);
00503 *key = Py_tss_NEEDS_INIT;
00504 }
00505 static CYTHON_INLINE int PyThread_tss_set(Py_tss_t *key, void *value) {
00506 return PyThread_set_key_value(*key, value);
00507 }
00508 static CYTHON_INLINE void * PyThread_tss_get(Py_tss_t *key) {
00509 return PyThread_get_key_value(*key);
00510 }
00511 #endif
00512 #if CYTHON_COMPILING_IN_CPYTHON || defined(_PyDict_NewPresized)
00513 #define __Pyx_PyDict_NewPresized(n) ((n <= 8) ? PyDict_New() : _PyDict_NewPresized(n))
00514 #else
00515 #define __Pyx_PyDict_NewPresized(n) PyDict_New()
00516 #endif
00517 #if PY_MAJOR_VERSION >= 3 || CYTHON_FUTURE_DIVISION
00518 #define __Pyx_PyNumber_Divide(x, y) PyNumber_TrueDivide(x, y)
00519 #define __Pyx_PyNumber_InPlaceDivide(x, y) PyNumber_InPlaceTrueDivide(x, y)
00520 #else
00521 #define __Pyx_PyNumber_Divide(x, y) PyNumber_Divide(x, y)
00522 #define __Pyx_PyNumber_InPlaceDivide(x, y) PyNumber_InPlaceDivide(x, y)
00523 #endif
00524 #if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x030500A1 && CYTHON_USE_UNICODE_INTERNALS
00525 #define __Pyx_PyDict_GetItemStr(dict, name) _PyDict_GetItem_KnownHash(dict, name, ((PyASCIIObject *)
name)->hash)
00526 #else

```

```

00527 #define __Pyx_PyDict_GetItemStr(dict, name) PyDict_GetItem(dict, name)
00528 #endif
00529 #if PY_VERSION_HEX > 0x03030000 && defined(PyUnicode_KIND)
00530 #define CYTHON_PEP393_ENABLED 1
00531 #if defined(PyUnicode_IS_READY)
00532 #define __Pyx_PyUnicode_READY(op) (likely(PyUnicode_IS_READY(op)) ?\
00533 0 : _PyUnicode_Ready((PyObject *) (op)))
00534 #else
00535 #define __Pyx_PyUnicode_READY(op) (0)
00536 #endif
00537 #define __Pyx_PyUnicode_GET_LENGTH(u) PyUnicode_GET_LENGTH(u)
00538 #define __Pyx_PyUnicode_READ_CHAR(u, i) PyUnicode_READ_CHAR(u, i)
00539 #define __Pyx_PyUnicode_MAX_CHAR_VALUE(u) PyUnicode_MAX_CHAR_VALUE(u)
00540 #define __Pyx_PyUnicode_KIND(u) PyUnicode_KIND(u)
00541 #define __Pyx_PyUnicode_DATA(u) PyUnicode_DATA(u)
00542 #define __Pyx_PyUnicode_READ(k, d, i) PyUnicode_READ(k, d, i)
00543 #define __Pyx_PyUnicode_WRITE(k, d, i, ch) PyUnicode_WRITE(k, d, i, ch)
00544 #if defined(PyUnicode_IS_READY) && defined(PyUnicode_GET_SIZE)
00545 #if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x03090000
00546 #define __Pyx_PyUnicode_IS_TRUE(u) (0 != (likely(PyUnicode_IS_READY(u)) ?
PyUnicode_GET_LENGTH(u) : ((PyCompactUnicodeObject *) (u))->wstr_length))
00547 #else
00548 #define __Pyx_PyUnicode_IS_TRUE(u) (0 != (likely(PyUnicode_IS_READY(u)) ?
PyUnicode_GET_LENGTH(u) : PyUnicode_GET_SIZE(u)))
00549 #endif
00550 #else
00551 #define __Pyx_PyUnicode_IS_TRUE(u) (0 != PyUnicode_GET_LENGTH(u))
00552 #endif
00553 #else
00554 #define CYTHON_PEP393_ENABLED 0
00555 #define PyUnicode_1BYTE_KIND 1
00556 #define PyUnicode_2BYTE_KIND 2
00557 #define PyUnicode_4BYTE_KIND 4
00558 #define __Pyx_PyUnicode_READY(op) (0)
00559 #define __Pyx_PyUnicode_GET_LENGTH(u) PyUnicode_GET_SIZE(u)
00560 #define __Pyx_PyUnicode_READ_CHAR(u, i) ((Py_UCS4) (PyUnicode_AS_UNICODE(u)[i]))
00561 #define __Pyx_PyUnicode_MAX_CHAR_VALUE(u) ((sizeof(Py_UNICODE) == 2) ? 65535 : 1114111)
00562 #define __Pyx_PyUnicode_KIND(u) (sizeof(Py_UNICODE))
00563 #define __Pyx_PyUnicode_DATA(u) ((void*) PyUnicode_AS_UNICODE(u))
00564 #define __Pyx_PyUnicode_READ(k, d, i) ((void) (k), (Py_UCS4) (((Py_UNICODE*) d)[i]))
00565 #define __Pyx_PyUnicode_WRITE(k, d, i, ch) (((void) (k)), ((Py_UNICODE*) d)[i] = ch)
00566 #define __Pyx_PyUnicode_IS_TRUE(u) (0 != PyUnicode_GET_SIZE(u))
00567 #endif
00568 #if CYTHON_COMPILING_IN_PYPY
00569 #define __Pyx_PyUnicode_Concat(a, b) PyNumber_Add(a, b)
00570 #define __Pyx_PyUnicode_ConcatSafe(a, b) PyNumber_Add(a, b)
00571 #else
00572 #define __Pyx_PyUnicode_Concat(a, b) PyUnicode_Concat(a, b)
00573 #define __Pyx_PyUnicode_ConcatSafe(a, b) ((unlikely((a) == Py_None) || unlikely((b) == Py_None)) ?\
00574 PyNumber_Add(a, b) : __Pyx_PyUnicode_Concat(a, b))
00575 #endif
00576 #if CYTHON_COMPILING_IN_PYPY && !defined(PyUnicode_Contains)
00577 #define PyUnicode_Contains(u, s) PySequence_Contains(u, s)
00578 #endif
00579 #if CYTHON_COMPILING_IN_PYPY && !defined(PyByteArray_Check)
00580 #define PyByteArray_Check(obj) PyObject_TypeCheck(obj, &PyByteArray_Type)
00581 #endif
00582 #if CYTHON_COMPILING_IN_PYPY && !defined(PyObject_Format)
00583 #define PyObject_Format(obj, fmt) PyObject_CallMethod(obj, "__format__", "O", fmt)
00584 #endif
00585 #define __Pyx_PyString_FormatSafe(a, b) ((unlikely((a) == Py_None || (PyString_Check(b) &&
!PyString_CheckExact(b)))) ? PyNumber_Remainder(a, b) : __Pyx_PyString_Format(a, b))
00586 #define __Pyx_PyUnicode_FormatSafe(a, b) ((unlikely((a) == Py_None || (PyUnicode_Check(b) &&
!PyUnicode_CheckExact(b)))) ? PyNumber_Remainder(a, b) : PyUnicode_Format(a, b))
00587 #if PY_MAJOR_VERSION >= 3
00588 #define __Pyx_PyString_Format(a, b) PyUnicode_Format(a, b)
00589 #else
00590 #define __Pyx_PyString_Format(a, b) PyString_Format(a, b)
00591 #endif
00592 #if PY_MAJOR_VERSION < 3 && !defined(PyObject_ASCII)
00593 #define PyObject_ASCII(o) PyObject_Repr(o)
00594 #endif
00595 #if PY_MAJOR_VERSION >= 3
00596 #define PyBaseString_Type PyUnicode_Type
00597 #define PyStringObject PyUnicodeObject
00598 #define PyString_Type PyUnicode_Type
00599 #define PyString_Check PyUnicode_Check
00600 #define PyString_CheckExact PyUnicode_CheckExact
00601 #ifndef PyObject_Unicode
00602 #define PyObject_Unicode PyObject_Str
00603 #endif
00604 #endif
00605 #if PY_MAJOR_VERSION >= 3
00606 #define __Pyx_PyBaseString_Check(obj) PyUnicode_Check(obj)
00607 #define __Pyx_PyBaseString_CheckExact(obj) PyUnicode_CheckExact(obj)
00608 #else
00609 #define __Pyx_PyBaseString_Check(obj) (PyString_Check(obj) || PyUnicode_Check(obj))

```

```

00610 #define __Pyx_PyBaseString_CheckExact(obj) (PyString_CheckExact(obj) || PyUnicode_CheckExact(obj))
00611 #endif
00612 #ifndef PySet_CheckExact
00613 #define PySet_CheckExact(obj) (Py_TYPE(obj) == &PySet_Type)
00614 #endif
00615 #if PY_VERSION_HEX >= 0x030900A4
00616 #define __Pyx_SET_REFCNT(obj, refcnt) Py_SET_REFCNT(obj, refcnt)
00617 #define __Pyx_SET_SIZE(obj, size) Py_SET_SIZE(obj, size)
00618 #else
00619 #define __Pyx_SET_REFCNT(obj, refcnt) Py_REFCNT(obj) = (refcnt)
00620 #define __Pyx_SET_SIZE(obj, size) Py_SIZE(obj) = (size)
00621 #endif
00622 #if CYTHON_ASSUME_SAFE_MACROS
00623 #define __Pyx_PySequence_SIZE(seq) Py_SIZE(seq)
00624 #else
00625 #define __Pyx_PySequence_SIZE(seq) PySequence_Size(seq)
00626 #endif
00627 #if PY_MAJOR_VERSION >= 3
00628 #define PyIntObject PyLongObject
00629 #define PyInt_Type PyLong_Type
00630 #define PyInt_Check(op) PyLong_Check(op)
00631 #define PyInt_CheckExact(op) PyLong_CheckExact(op)
00632 #define PyInt_FromString PyLong_FromString
00633 #define PyInt_FromUnicode PyLong_FromUnicode
00634 #define PyInt_FromLong PyLong_FromLong
00635 #define PyInt_FromSize_t PyLong_FromSize_t
00636 #define PyInt_FromSsize_t PyLong_FromSsize_t
00637 #define PyInt_AsLong PyLong_AsLong
00638 #define PyInt_AS_LONG PyLong_AS_LONG
00639 #define PyInt_AsSsize_t PyLong_AsSsize_t
00640 #define PyInt_AsUnsignedLongMask PyLong_AsUnsignedLongMask
00641 #define PyInt_AsUnsignedLongLongMask PyLong_AsUnsignedLongLongMask
00642 #define PyNumber_Int PyNumber_Long
00643 #endif
00644 #if PY_MAJOR_VERSION >= 3
00645 #define PyBoolObject PyLongObject
00646 #endif
00647 #if PY_MAJOR_VERSION >= 3 && CYTHON_COMPILING_IN_PYPY
00648 #ifndef PyUnicode_InternFromString
00649 #define PyUnicode_InternFromString(s) PyUnicode_FromString(s)
00650 #endif
00651 #endif
00652 #if PY_VERSION_HEX < 0x030200A4
00653 typedef long Py_hash_t;
00654 #define __Pyx_PyInt_FromHash_t PyInt_FromLong
00655 #define __Pyx_PyInt_AsHash_t __Pyx_PyIndex_AsHash_t
00656 #else
00657 #define __Pyx_PyInt_FromHash_t PyInt_FromSsize_t
00658 #define __Pyx_PyInt_AsHash_t __Pyx_PyIndex_AsSsize_t
00659 #endif
00660 #if PY_MAJOR_VERSION >= 3
00661 #define __Pyx_PyMethod_New(func, self, klass) ((self) ? ((void)(klass), PyMethod_New(func, self)) :
__Pyx_NewRef(func))
00662 #else
00663 #define __Pyx_PyMethod_New(func, self, klass) PyMethod_New(func, self, klass)
00664 #endif
00665 #if CYTHON_USE_ASYNC_SLOTS
00666 #if PY_VERSION_HEX >= 0x030500B1
00667 #define __Pyx_PyAsyncMethodsStruct PyAsyncMethods
00668 #define __Pyx_PyType_AsAsync(obj) (Py_TYPE(obj)->tp_as_async)
00669 #else
00670 #define __Pyx_PyType_AsAsync(obj) ((__Pyx_PyAsyncMethodsStruct*) (Py_TYPE(obj)->tp_reserved))
00671 #endif
00672 #else
00673 #define __Pyx_PyType_AsAsync(obj) NULL
00674 #endif
00675 #ifndef __Pyx_PyAsyncMethodsStruct
00676 typedef struct {
00677 unaryfunc am_await;
00678 unaryfunc am_aiter;
00679 unaryfunc am_anext;
00680 } __Pyx_PyAsyncMethodsStruct;
00681 #endif
00682
00683 #if defined(WIN32) || defined(MS_WINDOWS)
00684 #define _USE_MATH_DEFINES
00685 #endif
00686 #include <math.h>
00687 #ifndef NAN
00688 #define __PYX_NAN() ((float) NAN)
00689 #else
00690 static CYTHON_INLINE float __PYX_NAN() {
00691 float value;
00692 memset(&value, 0xFF, sizeof(value));
00693 return value;
00694 }
00695 #endif

```

```

00696 #if defined(__CYGWIN__) && defined(_LDBL_EQ_DBL)
00697 #define __Pyx_trunc1 trunc
00698 #else
00699 #define __Pyx_trunc1 trunc1
00700 #endif
00701
00702 #define __PYX_MARK_ERR_POS(f_index, lineno) \
00703 { __pyx_filename = __pyx_f[f_index]; (void)__pyx_filename; __pyx_lineno = lineno;
00704 (void)__pyx_lineno; __pyx_clineno = __LINE__; (void)__pyx_clineno; }
00705 #define __PYX_ERR(f_index, lineno, Ln_error) \
00706 { __PYX_MARK_ERR_POS(f_index, lineno) goto Ln_error; }
00707
00707 #ifndef __PYX_EXTERN_C
00708 #ifdef __cplusplus
00709 #define __PYX_EXTERN_C extern "C"
00710 #else
00711 #define __PYX_EXTERN_C extern
00712 #endif
00713 #endif
00714
00715 #define __PYX_HAVE__PyClical
00716 #define __PYX_HAVE_API__PyClical
00717 /* Early includes */
00718 #include "ios"
00719 #include "new"
00720 #include "stdexcept"
00721 #include "typeinfo"
00722 #include <vector>
00723 #include "PyClical.h"
00724 #include <string.h>
00725 #include <string>
00726 #ifdef _OPENMP
00727 #include <omp.h>
00728 #endif /* _OPENMP */
00729
00730 #if defined(PYREX_WITHOUT_ASSERTIONS) && !defined(CYTHON_WITHOUT_ASSERTIONS)
00731 #define CYTHON_WITHOUT_ASSERTIONS
00732 #endif
00733
00734 typedef struct {PyObject **p; const char *s; const Py_ssize_t n; const char* encoding;
00735 const char is_unicode; const char is_str; const char intern; } __Pyx_StringTabEntry;
00736
00737 #define __PYX_DEFAULT_STRING_ENCODING_IS_ASCII 0
00738 #define __PYX_DEFAULT_STRING_ENCODING_IS_UTF8 0
00739 #define __PYX_DEFAULT_STRING_ENCODING_IS_DEFAULT (PY_MAJOR_VERSION >= 3 &&
00740 __PYX_DEFAULT_STRING_ENCODING_IS_UTF8)
00741 #define __PYX_DEFAULT_STRING_ENCODING ""
00742 #define __Pyx_PyObject_FromString __Pyx_PyBytes_FromString
00743 #define __Pyx_PyObject_FromStringAndSize __Pyx_PyBytes_FromStringAndSize
00744 #define __Pyx_uchar_cast(c) ((unsigned char)c)
00745 #define __Pyx_long_cast(x) ((long)x)
00746 #define __Pyx_fits_Py_ssize_t(v, type, is_signed) (\
00747 (sizeof(type) < sizeof(Py_ssize_t)) || \
00748 (sizeof(type) > sizeof(Py_ssize_t) && \
00749 likely(v < (type)PY_SSIZE_T_MAX || \
00750 v == (type)PY_SSIZE_T_MAX) && \
00751 (!is_signed || likely(v > (type)PY_SSIZE_T_MIN || \
00752 v == (type)PY_SSIZE_T_MIN))) || \
00753 (sizeof(type) == sizeof(Py_ssize_t) && \
00754 (is_signed || likely(v < (type)PY_SSIZE_T_MAX || \
00755 v == (type)PY_SSIZE_T_MAX))))
00756 static CYTHON_INLINE int __Pyx_is_valid_index(Py_ssize_t i, Py_ssize_t limit) {
00757 return (size_t) i < (size_t) limit;
00758 }
00759
00759 #if defined (__cplusplus) && __cplusplus >= 201103L
00760 #include <cstdint>
00761 #define __Pyx_sst_abs(value) std::abs(value)
00762 #elif SIZEOF_INT >= SIZEOF_SIZE_T
00763 #define __Pyx_sst_abs(value) abs(value)
00764 #elif SIZEOF_LONG >= SIZEOF_SIZE_T
00765 #define __Pyx_sst_abs(value) labs(value)
00766 #elif defined (_MSC_VER)
00767 #define __Pyx_sst_abs(value) ((Py_ssize_t)_abs64(value))
00768 #elif defined (__STDC_VERSION__) && __STDC_VERSION__ >= 199901L
00769 #define __Pyx_sst_abs(value) llabs(value)
00770 #elif defined (__GNUC__)
00771 #define __Pyx_sst_abs(value) __builtin_llabs(value)
00772 #else
00773 #define __Pyx_sst_abs(value) ((value<0) ? -value : value)
00774 #endif
00775
00775 static CYTHON_INLINE const char* __Pyx_PyObject_AsString(PyObject*);
00776 static CYTHON_INLINE const char* __Pyx_PyObject_AsStringAndSize(PyObject*, Py_ssize_t* length);
00777 #define __Pyx_PyByteArray_FromString(s) PyByteArray_FromStringAndSize((const char*)s, strlen((const
00778 char*)s))
00779 #define __Pyx_PyByteArray_FromStringAndSize(s, l) PyByteArray_FromStringAndSize((const char*)s, l)
00780 #define __Pyx_PyBytes_FromString PyBytes_FromString
00781 #define __Pyx_PyBytes_FromStringAndSize PyBytes_FromStringAndSize

```

```

00780 static CYTHON_INLINE PyObject* __Pyx_PyUnicode_FromString(const char*);
00781 #if PY_MAJOR_VERSION < 3
00782 #define __Pyx_PyStr_FromString __Pyx_PyBytes_FromString
00783 #define __Pyx_PyStr_FromStringAndSize __Pyx_PyBytes_FromStringAndSize
00784 #else
00785 #define __Pyx_PyStr_FromString __Pyx_PyUnicode_FromString
00786 #define __Pyx_PyStr_FromStringAndSize __Pyx_PyUnicode_FromStringAndSize
00787 #endif
00788 #define __Pyx_PyBytes_AsWritableString(s) ((char*) PyBytes_AS_STRING(s))
00789 #define __Pyx_PyBytes_AsWritableSString(s) ((signed char*) PyBytes_AS_STRING(s))
00790 #define __Pyx_PyBytes_AsWritableUString(s) ((unsigned char*) PyBytes_AS_STRING(s))
00791 #define __Pyx_PyBytes_AsString(s) ((const char*) PyBytes_AS_STRING(s))
00792 #define __Pyx_PyBytes_AsSString(s) ((const signed char*) PyBytes_AS_STRING(s))
00793 #define __Pyx_PyBytes_AsUString(s) ((const unsigned char*) PyBytes_AS_STRING(s))
00794 #define __Pyx_PyObject_AsWritableString(s) ((char*) __Pyx_PyObject_AsString(s))
00795 #define __Pyx_PyObject_AsWritableSString(s) ((signed char*) __Pyx_PyObject_AsString(s))
00796 #define __Pyx_PyObject_AsWritableUString(s) ((unsigned char*) __Pyx_PyObject_AsString(s))
00797 #define __Pyx_PyObject_AsSString(s) ((const signed char*) __Pyx_PyObject_AsString(s))
00798 #define __Pyx_PyObject_AsUString(s) ((const unsigned char*) __Pyx_PyObject_AsString(s))
00799 #define __Pyx_PyObject_FromCString(s) __Pyx_PyObject_FromString((const char*)s)
00800 #define __Pyx_PyBytes_FromCString(s) __Pyx_PyBytes_FromString((const char*)s)
00801 #define __Pyx_PyByteArray_FromCString(s) __Pyx_PyByteArray_FromString((const char*)s)
00802 #define __Pyx_PyStr_FromCString(s) __Pyx_PyStr_FromString((const char*)s)
00803 #define __Pyx_PyUnicode_FromCString(s) __Pyx_PyUnicode_FromString((const char*)s)
00804 static CYTHON_INLINE size_t __Pyx_Py_UNICODE_strlen(const Py_UNICODE *u) {
00805 const Py_UNICODE *u_end = u;
00806 while (*u_end++) ;
00807 return (size_t)(u_end - u - 1);
00808 }
00809 #define __Pyx_PyUnicode_FromUnicode(u) PyUnicode_FromUnicode(u, __Pyx_Py_UNICODE_strlen(u))
00810 #define __Pyx_PyUnicode_FromUnicodeAndLength PyUnicode_FromUnicode
00811 #define __Pyx_PyUnicode_AsUnicode PyUnicode_AsUnicode
00812 #define __Pyx_NewRef(obj) (Py_INCREF(obj), obj)
00813 #define __Pyx_Owned_Py_None(b) __Pyx_NewRef(Py_None)
00814 static CYTHON_INLINE PyObject * __Pyx_PyBool_FromLong(long b);
00815 static CYTHON_INLINE int __Pyx_PyObject_IsTrue(PyObject*);
00816 static CYTHON_INLINE int __Pyx_PyObject_IsTrueAndDecref(PyObject*);
00817 static CYTHON_INLINE PyObject* __Pyx_PyNumber_IntOrLong(PyObject* x);
00818 #define __Pyx_PySequence_Tuple(obj)\
00819 (likely(PyTuple_CheckExact(obj)) ? __Pyx_NewRef(obj) : PySequence_Tuple(obj))
00820 static CYTHON_INLINE Py_ssize_t __Pyx_PyIndex_AsSsize_t(PyObject*);
00821 static CYTHON_INLINE PyObject * __Pyx_PyInt_FromSize_t(size_t);
00822 static CYTHON_INLINE Py_hash_t __Pyx_PyIndex_AsHash_t(PyObject*);
00823 #if CYTHON_ASSUME_SAFE_MACROS
00824 #define __pyx_PyFloat_AsDouble(x) (PyFloat_CheckExact(x) ? PyFloat_AS_DOUBLE(x) : PyFloat_AsDouble(x))
00825 #else
00826 #define __pyx_PyFloat_AsDouble(x) PyFloat_AsDouble(x)
00827 #endif
00828 #define __pyx_PyFloat_AsFloat(x) ((float) __pyx_PyFloat_AsDouble(x))
00829 #if PY_MAJOR_VERSION >= 3
00830 #define __Pyx_PyNumber_Int(x) (PyLong_CheckExact(x) ? __Pyx_NewRef(x) : PyNumber_Long(x))
00831 #else
00832 #define __Pyx_PyNumber_Int(x) (PyInt_CheckExact(x) ? __Pyx_NewRef(x) : PyNumber_Int(x))
00833 #endif
00834 #define __Pyx_PyNumber_Float(x) (PyFloat_CheckExact(x) ? __Pyx_NewRef(x) : PyNumber_Float(x))
00835 #if PY_MAJOR_VERSION < 3 && __PYX_DEFAULT_STRING_ENCODING_IS_ASCII
00836 static int __Pyx_sys_getdefaultencoding_not_ascii;
00837 static int __Pyx_init_sys_getdefaultencoding_params(void) {
00838 PyObject* sys;
00839 PyObject* default_encoding = NULL;
00840 PyObject* ascii_chars_u = NULL;
00841 PyObject* ascii_chars_b = NULL;
00842 const char* default_encoding_c;
00843 sys = PyImport_ImportModule("sys");
00844 if (!sys) goto bad;
00845 default_encoding = PyObject_CallMethod(sys, (char*) "getdefaultencoding", NULL);
00846 Py_DECREF(sys);
00847 if (!default_encoding) goto bad;
00848 default_encoding_c = PyBytes_AsString(default_encoding);
00849 if (!default_encoding_c) goto bad;
00850 if (strcmp(default_encoding_c, "ascii") == 0) {
00851 __Pyx_sys_getdefaultencoding_not_ascii = 0;
00852 } else {
00853 char ascii_chars[128];
00854 int c;
00855 for (c = 0; c < 128; c++) {
00856 ascii_chars[c] = c;
00857 }
00858 __Pyx_sys_getdefaultencoding_not_ascii = 1;
00859 ascii_chars_u = PyUnicode_DecodeASCII(ascii_chars, 128, NULL);
00860 if (!ascii_chars_u) goto bad;
00861 ascii_chars_b = PyUnicode_AsEncodedString(ascii_chars_u, default_encoding_c, NULL);
00862 if (!ascii_chars_b || !PyBytes_Check(ascii_chars_b) || memcmp(ascii_chars,
PyBytes_AS_STRING(ascii_chars_b), 128) != 0) {
00863 PyErr_Format(
00864 PyExc_ValueError,
00865 "This module compiled with c_string_encoding=ascii, but default encoding '%.200s' is

```

```

 not a superset of ascii.",
00866 default_encoding_c);
00867 goto bad;
00868 }
00869 Py_DECREF(ascii_chars_u);
00870 Py_DECREF(ascii_chars_b);
00871 }
00872 Py_DECREF(default_encoding);
00873 return 0;
00874 bad:
00875 Py_XDECREF(default_encoding);
00876 Py_XDECREF(ascii_chars_u);
00877 Py_XDECREF(ascii_chars_b);
00878 return -1;
00879 }
00880 #endif
00881 #if __PYX_DEFAULT_STRING_ENCODING_IS_DEFAULT && PY_MAJOR_VERSION >= 3
00882 #define __Pyx_PyUnicode_FromStringAndSize(c_str, size) PyUnicode_DecodeUTF8(c_str, size, NULL)
00883 #else
00884 #define __Pyx_PyUnicode_FromStringAndSize(c_str, size) PyUnicode_Decode(c_str, size,
__PYX_DEFAULT_STRING_ENCODING, NULL)
00885 #if __PYX_DEFAULT_STRING_ENCODING_IS_DEFAULT
00886 static char* __PYX_DEFAULT_STRING_ENCODING;
00887 static int __Pyx_init_sys_getdefaultencoding_params(void) {
00888 PyObject* sys;
00889 PyObject* default_encoding = NULL;
00890 char* default_encoding_c;
00891 sys = PyImport_ImportModule("sys");
00892 if (!sys) goto bad;
00893 default_encoding = PyObject_CallMethod(sys, (char*) (const char*) "getdefaultencoding", NULL);
00894 Py_DECREF(sys);
00895 if (!default_encoding) goto bad;
00896 default_encoding_c = PyBytes_AsString(default_encoding);
00897 if (!default_encoding_c) goto bad;
00898 __PYX_DEFAULT_STRING_ENCODING = (char*) malloc(strlen(default_encoding_c) + 1);
00899 if (!__PYX_DEFAULT_STRING_ENCODING) goto bad;
00900 strcpy(__PYX_DEFAULT_STRING_ENCODING, default_encoding_c);
00901 Py_DECREF(default_encoding);
00902 return 0;
00903 bad:
00904 Py_XDECREF(default_encoding);
00905 return -1;
00906 }
00907 #endif
00908 #endif
00909
00910
00911 /* Test for GCC > 2.95 */
00912 #if defined(__GNUC__) && (__GNUC__ > 2 || (__GNUC__ == 2 && (__GNUC_MINOR__ > 95)))
00913 #define likely(x) __builtin_expect(!!(x), 1)
00914 #define unlikely(x) __builtin_expect(!!(x), 0)
00915 #else /* !__GNUC__ or GCC < 2.95 */
00916 #define likely(x) (x)
00917 #define unlikely(x) (x)
00918 #endif /* __GNUC__ */
00919 static CYTHON_INLINE void __Pyx_pretend_to_initialize(void* ptr) { (void)ptr; }
00920
00921 static PyObject * __pyx_m = NULL;
00922 static PyObject * __pyx_d;
00923 static PyObject * __pyx_b;
00924 static PyObject * __pyx_cython_runtime = NULL;
00925 static PyObject * __pyx_empty_tuple;
00926 static PyObject * __pyx_empty_bytes;
00927 static PyObject * __pyx_empty_unicode;
00928 static int __pyx_lineno;
00929 static int __pyx_clineno = 0;
00930 static const char * __pyx_cfilenm= __FILE__;
00931 static const char * __pyx_filename;
00932
00933
00934 static const char * __pyx_f[] = {
00935 "PyClical.pyx",
00936 "stringsource",
00937 };
00938
00939 /*--- Type declarations ---*/
00940 struct __pyx_obj_8PyClical_index_set;
00941 struct __pyx_obj_8PyClical_clifford;
00942 struct __pyx_obj_8PyClical__pyx_scope_struct____iter____;
00943 struct __pyx_opt_args_8PyClical_approx_equal;
00944 struct __pyx_opt_args_8PyClical_sqrt;
00945 struct __pyx_opt_args_8PyClical_log;
00946 struct __pyx_opt_args_8PyClical_cos;
00947 struct __pyx_opt_args_8PyClical_acos;
00948 struct __pyx_opt_args_8PyClical_acosh;
00949 struct __pyx_opt_args_8PyClical_sin;
00950 struct __pyx_opt_args_8PyClical_asin;

```



```

00951 struct __pyx_opt_args_8PyClical_asinh;
00952 struct __pyx_opt_args_8PyClical_tan;
00953 struct __pyx_opt_args_8PyClical_atan;
00954 struct __pyx_opt_args_8PyClical_atanh;
00955 struct __pyx_opt_args_8PyClical_random_clifford;
00956
00957 /* "PyClical.pyx":1359
00958 * return glucat.error_squared(toClifford(lhs), toClifford(rhs), <scalar_t>threshold)
00959 *
00960 * cpdef inline approx_equal(lhs, rhs, threshold=None, tol=None): # ««««««««
00961 * """
00962 * Test for approximate equality of multivectors.
00963 */
00964 struct __pyx_opt_args_8PyClical_approx_equal {
00965 int __pyx_n;
00966 PyObject *threshold;
00967 PyObject *tol;
00968 };
00969
00970 /* "PyClical.pyx":1591
00971 * return clifford().wrap(glucat.complexifier(toClifford(obj)))
00972 *
00973 * cpdef inline sqrt(obj, i = None): # ««««««««
00974 * """
00975 * Square root of multivector with optional complexifier.
00976 */
00977 struct __pyx_opt_args_8PyClical_sqrt {
00978 int __pyx_n;
00979 PyObject *i;
00980 };
00981
00982 /* "PyClical.pyx":1628
00983 * return clifford().wrap(glucat.exp(toClifford(obj)))
00984 *
00985 * cpdef inline log(obj,i = None): # ««««««««
00986 * """
00987 * Natural logarithm of multivector with optional complexifier.
00988 */
00989 struct __pyx_opt_args_8PyClical_log {
00990 int __pyx_n;
00991 PyObject *i;
00992 };
00993
00994 /* "PyClical.pyx":1651
00995 * return clifford().wrap(glucat.log(toClifford(obj)))
00996 *
00997 * cpdef inline cos(obj,i = None): # ««««««««
00998 * """
00999 * Cosine of multivector with optional complexifier.
01000 */
01001 struct __pyx_opt_args_8PyClical_cos {
01002 int __pyx_n;
01003 PyObject *i;
01004 };
01005
01006 /* "PyClical.pyx":1668
01007 * return clifford().wrap(glucat.cos(toClifford(obj)))
01008 *
01009 * cpdef inline acos(obj,i = None): # ««««««««
01010 * """
01011 * Inverse cosine of multivector with optional complexifier.
01012 */
01013 struct __pyx_opt_args_8PyClical_acos {
01014 int __pyx_n;
01015 PyObject *i;
01016 };
01017
01018 /* "PyClical.pyx":1705
01019 * return clifford().wrap(glucat.cosh(toClifford(obj)))
01020 *
01021 * cpdef inline acosh(obj,i = None): # ««««««««
01022 * """
01023 * Inverse hyperbolic cosine of multivector with optional complexifier.
01024 */
01025 struct __pyx_opt_args_8PyClical_acosh {
01026 int __pyx_n;
01027 PyObject *i;
01028 };
01029
01030 /* "PyClical.pyx":1728
01031 * return clifford().wrap(glucat.acosh(toClifford(obj)))
01032 *
01033 * cpdef inline sin(obj,i = None): # ««««««««
01034 * """
01035 * Sine of multivector with optional complexifier.
01036 */
01037 struct __pyx_opt_args_8PyClical_sin {

```



```

01038 int __pyx_n;
01039 PyObject *i;
01040 };
01041
01042 /* "PyClical.pyx":1747
01043 * return clifford().wrap(glucat.sin(toClifford(obj)))
01044 *
01045 * cpdef inline asin(obj,i = None): # ««««««««
01046 * """
01047 * Inverse sine of multivector with optional complexifier.
01048 */
01049 struct __pyx_opt_args_8PyClical_asin {
01050 int __pyx_n;
01051 PyObject *i;
01052 };
01053
01054 /* "PyClical.pyx":1782
01055 * return clifford().wrap(glucat.sinh(toClifford(obj)))
01056 *
01057 * cpdef inline asinh(obj,i = None): # ««««««««
01058 * """
01059 * Inverse hyperbolic sine of multivector with optional complexifier.
01060 */
01061 struct __pyx_opt_args_8PyClical_asinh {
01062 int __pyx_n;
01063 PyObject *i;
01064 };
01065
01066 /* "PyClical.pyx":1801
01067 * return clifford().wrap(glucat.asinh(toClifford(obj)))
01068 *
01069 * cpdef inline tan(obj,i = None): # ««««««««
01070 * """
01071 * Tangent of multivector with optional complexifier.
01072 */
01073 struct __pyx_opt_args_8PyClical_tan {
01074 int __pyx_n;
01075 PyObject *i;
01076 };
01077
01078 /* "PyClical.pyx":1818
01079 * return clifford().wrap(glucat.tan(toClifford(obj)))
01080 *
01081 * cpdef inline atan(obj,i = None): # ««««««««
01082 * """
01083 * Inverse tangent of multivector with optional complexifier.
01084 */
01085 struct __pyx_opt_args_8PyClical_atan {
01086 int __pyx_n;
01087 PyObject *i;
01088 };
01089
01090 /* "PyClical.pyx":1847
01091 * return clifford().wrap(glucat.tanh(toClifford(obj)))
01092 *
01093 * cpdef inline atanh(obj,i = None): # ««««««««
01094 * """
01095 * Inverse hyperbolic tangent of multivector with optional complexifier.
01096 */
01097 struct __pyx_opt_args_8PyClical_atanh {
01098 int __pyx_n;
01099 PyObject *i;
01100 };
01101
01102 /* "PyClical.pyx":1864
01103 * return clifford().wrap(glucat.atanh(toClifford(obj)))
01104 *
01105 * cpdef inline random_clifford(index_set ixt, fill = 1.0): # ««««««««
01106 * """
01107 * Random multivector within a frame.
01108 */
01109 struct __pyx_opt_args_8PyClical_random_clifford {
01110 int __pyx_n;
01111 PyObject *fill;
01112 };
01113
01114 /* "PyClical.pyx":38
01115 *
01116 * # Forward reference
01117 * cdef class index_set # ««««««««
01118 *
01119 * cdef inline IndexSet toIndexSet(obj):
01120 */
01121 struct __pyx_obj_8PyClical_index_set {
01122 PyObject_HEAD
01123 struct __pyx_vtabstruct_8PyClical_index_set *__pyx_vtab;
01124 IndexSet *instance;

```

```

01125 };
01126
01127
01128 /* "PyClicl.pyx":532
01129 *
01130 * # Forward reference.
01131 * cdef class clifford # ««««««««
01132 *
01133 * cdef inline Clifford toClifford(obj):
01134 */
01135 struct __pyx_obj_8PyClicl_clifford {
01136 PyObject_HEAD
01137 struct __pyx_vtabstruct_8PyClicl_clifford *__pyx_vtab;
01138 Clifford *instance;
01139 };
01140
01141
01142 /* "PyClicl.pyx":229
01143 * return self.instance.getitem(idx)
01144 *
01145 * def __iter__(self): # ««««««««
01146 * """
01147 * Iterate over the indices of an index_set.
01148 */
01149 struct __pyx_obj_8PyClicl__pyx_scope_struct__iter__ {
01150 PyObject_HEAD
01151 PyObject *__pyx_v_idx;
01152 struct __pyx_obj_8PyClicl_index_set *__pyx_v_self;
01153 PyObject *__pyx_t_0;
01154 Py_ssize_t __pyx_t_1;
01155 PyObject *(*__pyx_t_2)(PyObject *);
01156 };
01157
01158
01159
01160 /* "PyClicl.pyx":46
01161 * return index_set(obj).instance[0]
01162 *
01163 * cdef class index_set: # ««««««««
01164 * """
01165 * Python class index_set wraps C++ class IndexSet.
01166 */
01167
01168 struct __pyx_vtabstruct_8PyClicl_index_set {
01169 PyObject *(*wrap)(struct __pyx_obj_8PyClicl_index_set *, IndexSet);
01170 IndexSet (*unwrap)(struct __pyx_obj_8PyClicl_index_set *);
01171 PyObject *(*copy)(struct __pyx_obj_8PyClicl_index_set *, int __pyx_skip_dispatch);
01172 };
01173 static struct __pyx_vtabstruct_8PyClicl_index_set *__pyx_vtabptr_8PyClicl_index_set;
01174 static CYTHON_INLINE PyObject *__pyx_f_8PyClicl_9index_set_wrap(struct __pyx_obj_8PyClicl_index_set
*, IndexSet);
01175 static CYTHON_INLINE IndexSet __pyx_f_8PyClicl_9index_set_unwrap(struct __pyx_obj_8PyClicl_index_set
*);
01176
01177
01178 /* "PyClicl.pyx":537
01179 * return clifford(obj).instance[0]
01180 *
01181 * cdef class clifford: # ««««««««
01182 * """
01183 * Python class clifford wraps C++ class Clifford.
01184 */
01185
01186 struct __pyx_vtabstruct_8PyClicl_clifford {
01187 PyObject *(*wrap)(struct __pyx_obj_8PyClicl_clifford *, Clifford);
01188 Clifford (*unwrap)(struct __pyx_obj_8PyClicl_clifford *);
01189 PyObject *(*copy)(struct __pyx_obj_8PyClicl_clifford *, int __pyx_skip_dispatch);
01190 };
01191 static struct __pyx_vtabstruct_8PyClicl_clifford *__pyx_vtabptr_8PyClicl_clifford;
01192 static CYTHON_INLINE PyObject *__pyx_f_8PyClicl_8clifford_wrap(struct __pyx_obj_8PyClicl_clifford *,
Clifford);
01193 static CYTHON_INLINE Clifford __pyx_f_8PyClicl_8clifford_unwrap(struct __pyx_obj_8PyClicl_clifford
*);
01194
01195 /* --- Runtime support code (head) --- */
01196 /* Refnanny.proto */
01197 #ifndef CYTHON_REFNANNY
01198 #define CYTHON_REFNANNY 0
01199 #endif
01200 #if CYTHON_REFNANNY
01201 typedef struct {
01202 void (*INCREf)(void*, PyObject*, int);
01203 void (*DECREf)(void*, PyObject*, int);
01204 void (*GOTREf)(void*, PyObject*, int);
01205 void (*GIVEREF)(void*, PyObject*, int);
01206 void* (*SetupContext)(const char*, int, const char*);
01207 void (*FinishContext)(void**);
01208 };

```

```

01208 } __Pyx_RefNannyAPIStruct;
01209 static __Pyx_RefNannyAPIStruct *__Pyx_RefNanny = NULL;
01210 static __Pyx_RefNannyAPIStruct *__Pyx_RefNannyImportAPI(const char *modname);
01211 #define __Pyx_RefNannyDeclarations void *__pyx_refnanny = NULL;
01212 #ifndef WITH_THREAD
01213 #define __Pyx_RefNannySetupContext(name, acquire_gil)\
01214 if (acquire_gil) {\
01215 PyGILState_STATE __pyx_gilstate_save = PyGILState_Ensure();\
01216 __pyx_refnanny = __Pyx_RefNanny->SetupContext((name), __LINE__, __FILE__);\
01217 PyGILState_Release(__pyx_gilstate_save);\
01218 } else {\
01219 __pyx_refnanny = __Pyx_RefNanny->SetupContext((name), __LINE__, __FILE__);\
01220 }
01221 #else
01222 #define __Pyx_RefNannySetupContext(name, acquire_gil)\
01223 __pyx_refnanny = __Pyx_RefNanny->SetupContext((name), __LINE__, __FILE__)
01224 #endif
01225 #define __Pyx_RefNannyFinishContext()\
01226 __Pyx_RefNanny->FinishContext(&__pyx_refnanny)
01227 #define __Pyx_INCREF(r) __Pyx_RefNanny->INCRREF(__pyx_refnanny, (PyObject *) (r), __LINE__)
01228 #define __Pyx_DECREF(r) __Pyx_RefNanny->DECRREF(__pyx_refnanny, (PyObject *) (r), __LINE__)
01229 #define __Pyx_GOTREF(r) __Pyx_RefNanny->GOTREF(__pyx_refnanny, (PyObject *) (r), __LINE__)
01230 #define __Pyx_GIVEREF(r) __Pyx_RefNanny->GIVEREF(__pyx_refnanny, (PyObject *) (r), __LINE__)
01231 #define __Pyx_XINCRREF(r) do { if((r) != NULL) {__Pyx_INCREF(r); }} while(0)
01232 #define __Pyx_XDECREF(r) do { if((r) != NULL) {__Pyx_DECREF(r); }} while(0)
01233 #define __Pyx_XGOTREF(r) do { if((r) != NULL) {__Pyx_GOTREF(r); }} while(0)
01234 #define __Pyx_XGIVEREF(r) do { if((r) != NULL) {__Pyx_GIVEREF(r); }} while(0)
01235 #else
01236 #define __Pyx_RefNannyDeclarations
01237 #define __Pyx_RefNannySetupContext(name, acquire_gil)
01238 #define __Pyx_RefNannyFinishContext()
01239 #define __Pyx_INCREF(r) Py_INCREF(r)
01240 #define __Pyx_DECREF(r) Py_DECREF(r)
01241 #define __Pyx_GOTREF(r)
01242 #define __Pyx_GIVEREF(r)
01243 #define __Pyx_XINCRREF(r) Py_XINCRREF(r)
01244 #define __Pyx_XDECREF(r) Py_XDECREF(r)
01245 #define __Pyx_XGOTREF(r)
01246 #define __Pyx_XGIVEREF(r)
01247 #endif
01248 #define __Pyx_XDECREF_SET(r, v) do {\
01249 PyObject *tmp = (PyObject *) r;\
01250 r = v; __Pyx_XDECREF(tmp);\
01251 } while (0)
01252 #define __Pyx_DECREF_SET(r, v) do {\
01253 PyObject *tmp = (PyObject *) r;\
01254 r = v; __Pyx_DECREF(tmp);\
01255 } while (0)
01256 #define __Pyx_CLEAR(r) do { PyObject* tmp = ((PyObject*)(r)); r = NULL; __Pyx_DECREF(tmp);} while(0)
01257 #define __Pyx_XCLEAR(r) do { if((r) != NULL) {PyObject* tmp = ((PyObject*)(r)); r = NULL; __Pyx_DECREF(tmp);} } while(0)
01258
01259 /* PyObjectGetAttrStr.proto */
01260 #if CYTHON_USE_TYPE_SLOTS
01261 static CYTHON_INLINE PyObject* __Pyx_PyObject_GetAttrStr(PyObject* obj, PyObject* attr_name);
01262 #else
01263 #define __Pyx_PyObject_GetAttrStr(o,n) PyObject_GetAttr(o,n)
01264 #endif
01265
01266 /* GetBuiltinName.proto */
01267 static PyObject* __Pyx_GetBuiltinName(PyObject *name);
01268
01269 /* PyCFunctionFastCall.proto */
01270 #if CYTHON_FAST_PYCCALL
01271 static CYTHON_INLINE PyObject* __Pyx_PyCFunction_FastCall(PyObject *func, PyObject **args, Py_ssize_t nargs);
01272 #else
01273 #define __Pyx_PyCFunction_FastCall(func, args, nargs) (assert(0), NULL)
01274 #endif
01275
01276 /* PyFunctionFastCall.proto */
01277 #if CYTHON_FAST_PYCALL
01278 #define __Pyx_PyFunction_FastCall(func, args, nargs)\
01279 __Pyx_PyFunction_FastCallDict((func), (args), (nargs), NULL)
01280 #if 1 || PY_VERSION_HEX < 0x030600B1
01281 static PyObject* __Pyx_PyFunction_FastCallDict(PyObject *func, PyObject **args, Py_ssize_t nargs, PyObject *kwargs);
01282 #else
01283 #define __Pyx_PyFunction_FastCallDict(func, args, nargs, kwargs) _PyFunction_FastCallDict(func, args, nargs, kwargs)
01284 #endif
01285 #define __Pyx_BUILD_ASSERT_EXPR(cond)\
01286 (sizeof(char [1 - 2*(cond)]) - 1)
01287 #ifndef Py_MEMBER_SIZE
01288 #define Py_MEMBER_SIZE(type, member) sizeof(((type *)0)->member)
01289 #endif

```

```

01290 #if CYTHON_FAST_PYCALL
01291 static size_t __pyx_pyframe_localsplus_offset = 0;
01292 #include "frameobject.h"
01293 #define __Pyx_PyFrame_Initialize_Offsets()\
01294 ((void)__Pyx_BUILD_ASSERT_EXPR(sizeof(PyFrameObject) == offsetof(PyFrameObject, f_localsplus) +
Py_MEMBER_SIZE(PyFrameObject, f_localsplus)),\
01295 (void)(__pyx_pyframe_localsplus_offset = ((size_t)PyFrame_Type.tp_basicsize) -
Py_MEMBER_SIZE(PyFrameObject, f_localsplus)))
01296 #define __Pyx_PyFrame_GetLocalsplus(frame)\
01297 (assert(__pyx_pyframe_localsplus_offset), (PyObject *)(((char *) (frame)) +
__pyx_pyframe_localsplus_offset))
01298 #endif // CYTHON_FAST_PYCALL
01299 #endif
01300
01301 /* PyObjectCall.proto */
01302 #if CYTHON_COMPILING_IN_CPYTHON
01303 static CYTHON_INLINE PyObject* __Pyx_PyObject_Call(PyObject *func, PyObject *arg, PyObject *kw);
01304 #else
01305 #define __Pyx_PyObject_Call(func, arg, kw) PyObject_Call(func, arg, kw)
01306 #endif
01307
01308 /* PyObjectCallMeth0.proto */
01309 #if CYTHON_COMPILING_IN_CPYTHON
01310 static CYTHON_INLINE PyObject* __Pyx_PyObject_CallMeth0(PyObject *func, PyObject *arg);
01311 #endif
01312
01313 /* PyObjectCallOneArg.proto */
01314 static CYTHON_INLINE PyObject* __Pyx_PyObject_CallOneArg(PyObject *func, PyObject *arg);
01315
01316 /* PyThreadStateGet.proto */
01317 #if CYTHON_FAST_THREAD_STATE
01318 #define __Pyx_PyThreadState_declare PyThreadState *__pyx_tstate;
01319 #define __Pyx_PyThreadState_assign __pyx_tstate = __Pyx_PyThreadState_Current;
01320 #define __Pyx_PyErr_Occurred() __pyx_tstate->curexc_type
01321 #else
01322 #define __Pyx_PyThreadState_declare
01323 #define __Pyx_PyThreadState_assign
01324 #define __Pyx_PyErr_Occurred() PyErr_Occurred()
01325 #endif
01326
01327 /* PyErrFetchRestore.proto */
01328 #if CYTHON_FAST_THREAD_STATE
01329 #define __Pyx_PyErr_Clear() __Pyx_ErrRestore(NULL, NULL, NULL)
01330 #define __Pyx_ErrRestoreWithState(type, value, tb) __Pyx_ErrRestoreInState(PyThreadState_GET(), type,
value, tb)
01331 #define __Pyx_ErrFetchWithState(type, value, tb) __Pyx_ErrFetchInState(PyThreadState_GET(), type,
value, tb)
01332 #define __Pyx_ErrRestore(type, value, tb) __Pyx_ErrRestoreInState(__pyx_tstate, type, value, tb)
01333 #define __Pyx_ErrFetch(type, value, tb) __Pyx_ErrFetchInState(__pyx_tstate, type, value, tb)
01334 static CYTHON_INLINE void __Pyx_ErrRestoreInState(PyThreadState *tstate, PyObject *type, PyObject
**value, PyObject **tb);
01335 static CYTHON_INLINE void __Pyx_ErrFetchInState(PyThreadState *tstate, PyObject **type, PyObject
**value, PyObject **tb);
01336 #if CYTHON_COMPILING_IN_CPYTHON
01337 #define __Pyx_PyErr_SetNone(exc) (Py_INCREF(exc), __Pyx_ErrRestore((exc), NULL, NULL))
01338 #else
01339 #define __Pyx_PyErr_SetNone(exc) PyErr_SetNone(exc)
01340 #endif
01341 #else
01342 #define __Pyx_PyErr_Clear() PyErr_Clear()
01343 #define __Pyx_PyErr_SetNone(exc) PyErr_SetNone(exc)
01344 #define __Pyx_ErrRestoreWithState(type, value, tb) PyErr_Restore(type, value, tb)
01345 #define __Pyx_ErrFetchWithState(type, value, tb) PyErr_Fetch(type, value, tb)
01346 #define __Pyx_ErrRestoreInState(tstate, type, value, tb) PyErr_Restore(type, value, tb)
01347 #define __Pyx_ErrFetchInState(tstate, type, value, tb) PyErr_Fetch(type, value, tb)
01348 #define __Pyx_ErrRestore(type, value, tb) PyErr_Restore(type, value, tb)
01349 #define __Pyx_ErrFetch(type, value, tb) PyErr_Fetch(type, value, tb)
01350 #endif
01351
01352 /* WriteUnraisableException.proto */
01353 static void __Pyx_WriteUnraisable(const char *name, int clineno,
01354 int lineno, const char *filename,
01355 int full_traceback, int nogil);
01356
01357 /* PyDictVersioning.proto */
01358 #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_TYPE_SLOTS
01359 #define __PYX_DICT_VERSION_INIT ((PY_UINT64_T) -1)
01360 #define __PYX_GET_DICT_VERSION(dict) (((PyDictObject*)(dict))->ma_version_tag)
01361 #define __PYX_UPDATE_DICT_CACHE(dict, value, cache_var, version_var)\
01362 (version_var) = __PYX_GET_DICT_VERSION(dict);\
01363 (cache_var) = (value);
01364 #define __PYX_PY_DICT_LOOKUP_IF_MODIFIED(VAR, DICT, LOOKUP) {\
01365 static PY_UINT64_T __pyx_dict_version = 0;\
01366 static PyObject *__pyx_dict_cached_value = NULL;\
01367 if (likely(__PYX_GET_DICT_VERSION(DICT) == __pyx_dict_version)) {\
01368 (VAR) = __pyx_dict_cached_value;\
01369 } else {\

```

```

01370 (VAR) = __pyx_dict_cached_value = (LOOKUP);\
01371 __pyx_dict_version = __PYX_GET_DICT_VERSION(DICT);\
01372 }\
01373 }
01374 static CYTHON_INLINE PY_UINT64_T __Pyx_get_tp_dict_version(PyObject *obj);
01375 static CYTHON_INLINE PY_UINT64_T __Pyx_get_object_dict_version(PyObject *obj);
01376 static CYTHON_INLINE int __Pyx_object_dict_version_matches(PyObject* obj, PY_UINT64_T tp_dict_version,
PY_UINT64_T obj_dict_version);
01377 #else
01378 #define __PYX_GET_DICT_VERSION(dict) (0)
01379 #define __PYX_UPDATE_DICT_CACHE(dict, value, cache_var, version_var)
01380 #define __PYX_PY_DICT_LOOKUP_IF_MODIFIED(VAR, DICT, LOOKUP) (VAR) = (LOOKUP);
01381 #endif
01382
01383 /* PyObjectCallNoArg.proto */
01384 #if CYTHON_COMPILING_IN_CPYTHON
01385 static CYTHON_INLINE PyObject* __Pyx_PyObject_CallNoArg(PyObject *func);
01386 #else
01387 #define __Pyx_PyObject_CallNoArg(func) __Pyx_PyObject_Call(func, __pyx_empty_tuple, NULL)
01388 #endif
01389
01390 /* RaiseDoubleKeywords.proto */
01391 static void __Pyx_RaiseDoubleKeywordsError(const char* func_name, PyObject* kw_name);
01392
01393 /* ParseKeywords.proto */
01394 static int __Pyx_ParseOptionalKeywords(PyObject *kwds, PyObject **argnames[],\
PyObject *kwds2, PyObject *values[], Py_ssize_t num_pos_args,\
const char* function_name);
01397
01398 /* RaiseArgTupleInvalid.proto */
01399 static void __Pyx_RaiseArgtupleInvalid(const char* func_name, int exact,
Py_ssize_t num_min, Py_ssize_t num_max, Py_ssize_t num_found);
01401
01402 /* GetModuleGlobalName.proto */
01403 #if CYTHON_USE_DICT_VERSIONS
01404 #define __Pyx_GetModuleGlobalName(var, name) {\
01405 static PY_UINT64_T __pyx_dict_version = 0;\
01406 static PyObject * __pyx_dict_cached_value = NULL;\
01407 (var) = (likely(__pyx_dict_version == __PYX_GET_DICT_VERSION(__pyx_d))) ?\
01408 (likely(__pyx_dict_cached_value) ? __Pyx_NewRef(__pyx_dict_cached_value) :\
__Pyx_GetBuiltinName(name)) :\
__Pyx_GetModuleGlobalName(name, &__pyx_dict_version, &__pyx_dict_cached_value);\
01410 }
01411 #define __Pyx_GetModuleGlobalNameUncached(var, name) {\
01412 PY_UINT64_T __pyx_dict_version;\
01413 PyObject * __pyx_dict_cached_value;\
01414 (var) = __Pyx_GetModuleGlobalName(name, &__pyx_dict_version, &__pyx_dict_cached_value);\
01415 }
01416 static PyObject * __Pyx_GetModuleGlobalName(PyObject *name, PY_UINT64_T *dict_version, PyObject
**dict_cached_value);
01417 #else
01418 #define __Pyx_GetModuleGlobalName(var, name) (var) = __Pyx_GetModuleGlobalName(name)
01419 #define __Pyx_GetModuleGlobalNameUncached(var, name) (var) = __Pyx_GetModuleGlobalName(name)
01420 static CYTHON_INLINE PyObject * __Pyx_GetModuleGlobalName(PyObject *name);
01421 #endif
01422
01423 /* GetTopmostException.proto */
01424 #if CYTHON_USE_EXC_INFO_STACK
01425 static _PyErr_StackItem * __Pyx_PyErr_GetTopmostException(PyThreadState *tstate);
01426 #endif
01427
01428 /* SaveResetException.proto */
01429 #if CYTHON_FAST_THREAD_STATE
01430 #define __Pyx_ExceptionSave(type, value, tb) __Pyx_ExceptionSave(__pyx_tstate, type, value, tb)
01431 static CYTHON_INLINE void __Pyx_ExceptionSave(PyThreadState *tstate, PyObject **type, PyObject
**value, PyObject **tb);
01432 #define __Pyx_ExceptionReset(type, value, tb) __Pyx_ExceptionReset(__pyx_tstate, type, value, tb)
01433 static CYTHON_INLINE void __Pyx_ExceptionReset(PyThreadState *tstate, PyObject *type, PyObject
*value, PyObject *tb);
01434 #else
01435 #define __Pyx_ExceptionSave(type, value, tb) PyErr_GetExcInfo(type, value, tb)
01436 #define __Pyx_ExceptionReset(type, value, tb) PyErr_SetExcInfo(type, value, tb)
01437 #endif
01438
01439 /* PyErrExceptionMatches.proto */
01440 #if CYTHON_FAST_THREAD_STATE
01441 #define __Pyx_PyErr_ExceptionMatches(err) __Pyx_PyErr_ExceptionMatchesInState(__pyx_tstate, err)
01442 static CYTHON_INLINE int __Pyx_PyErr_ExceptionMatches(PyThreadState* tstate, PyObject* err);
01443 #else
01444 #define __Pyx_PyErr_ExceptionMatches(err) PyErr_ExceptionMatches(err)
01445 #endif
01446
01447 /* GetException.proto */
01448 #if CYTHON_FAST_THREAD_STATE
01449 #define __Pyx_GetException(type, value, tb) __Pyx_GetException(__pyx_tstate, type, value, tb)
01450 static int __Pyx_GetException(PyThreadState *tstate, PyObject **type, PyObject **value, PyObject
**tb);

```

```

01451 #else
01452 static int __Pyx_GetException(PyObject **type, PyObject **value, PyObject **tb);
01453 #endif
01454
01455 /* RaiseException.proto */
01456 static void __Pyx_Raise(PyObject *type, PyObject *value, PyObject *tb, PyObject *cause);
01457
01458 /* PyObjectCall2Args.proto */
01459 static CYTHON_UNUSED PyObject* __Pyx_PyObject_Call2Args(PyObject* function, PyObject* arg1, PyObject*
arg2);
01460
01461 /* PyIntBinop.proto */
01462 #if !CYTHON_COMPILING_IN_PYPY
01463 static PyObject* __Pyx_PyInt_AddObjC(PyObject *op1, PyObject *op2, long intval, int inplace, int
zerodivision_check);
01464 #else
01465 #define __Pyx_PyInt_AddObjC(op1, op2, intval, inplace, zerodivision_check)\
01466 (inplace ? PyNumber_InPlaceAdd(op1, op2) : PyNumber_Add(op1, op2))
01467 #endif
01468
01469 /* PySequenceContains.proto */
01470 static CYTHON_INLINE int __Pyx_PySequence_ContainsTF(PyObject* item, PyObject* seq, int eq) {
01471 int result = PySequence_Contains(seq, item);
01472 return unlikely(result < 0) ? result : (result == (eq == Py_EQ));
01473 }
01474
01475 /* IncludeCppStringH.proto */
01476 #include <string>
01477
01478 /* decode_c_string_utf16.proto */
01479 static CYTHON_INLINE PyObject *__Pyx_PyUnicode_DecodeUTF16(const char *s, Py_ssize_t size, const char
*errors) {
01480 int byteorder = 0;
01481 return PyUnicode_DecodeUTF16(s, size, errors, &byteorder);
01482 }
01483 static CYTHON_INLINE PyObject *__Pyx_PyUnicode_DecodeUTF16LE(const char *s, Py_ssize_t size, const
char *errors) {
01484 int byteorder = -1;
01485 return PyUnicode_DecodeUTF16(s, size, errors, &byteorder);
01486 }
01487 static CYTHON_INLINE PyObject *__Pyx_PyUnicode_DecodeUTF16BE(const char *s, Py_ssize_t size, const
char *errors) {
01488 int byteorder = 1;
01489 return PyUnicode_DecodeUTF16(s, size, errors, &byteorder);
01490 }
01491
01492 /* decode_c_bytes.proto */
01493 static CYTHON_INLINE PyObject* __Pyx_decode_c_bytes(
01494 const char* cstring, Py_ssize_t length, Py_ssize_t start, Py_ssize_t stop,
01495 const char* encoding, const char* errors,
01496 PyObject* (*decode_func)(const char *s, Py_ssize_t size, const char *errors));
01497
01498 /* decode_cpp_string.proto */
01499 static CYTHON_INLINE PyObject* __Pyx_decode_cpp_string(
01500 std::string cppstring, Py_ssize_t start, Py_ssize_t stop,
01501 const char* encoding, const char* errors,
01502 PyObject* (*decode_func)(const char *s, Py_ssize_t size, const char *errors)) {
01503 return __Pyx_decode_c_bytes(
01504 cppstring.data(), cppstring.size(), start, stop, encoding, errors, decode_func);
01505 }
01506
01507 /* SwapException.proto */
01508 #if CYTHON_FAST_THREAD_STATE
01509 #define __Pyx_ExceptionSwap(type, value, tb) __Pyx__ExceptionSwap(__pyx_tstate, type, value, tb)
01510 static CYTHON_INLINE void __Pyx__ExceptionSwap(PyThreadState *tstate, PyObject **type, PyObject
**value, PyObject **tb);
01511 #else
01512 static CYTHON_INLINE void __Pyx_ExceptionSwap(PyObject **type, PyObject **value, PyObject **tb);
01513 #endif
01514
01515 /* SetItemInt.proto */
01516 #define __Pyx_SetItemInt(o, i, v, type, is_signed, to_py_func, is_list, wraparound, boundscheck)\
01517 (__Pyx_fits_Py_ssize_t(i, type, is_signed) ?\
01518 __Pyx_SetItemInt_Fast(o, (Py_ssize_t)i, v, is_list, wraparound, boundscheck) :\
01519 (is_list ? (PyErr_SetString(PyExc_IndexError, "list assignment index out of range"), -1) :\
01520 __Pyx_SetItemInt_Generic(o, to_py_func(i), v))
01521 static int __Pyx_SetItemInt_Generic(PyObject *o, PyObject *j, PyObject *v);
01522 static CYTHON_INLINE int __Pyx_SetItemInt_Fast(PyObject *o, Py_ssize_t i, PyObject *v,
01523 int is_list, int wraparound, int boundscheck);
01524
01525 /* ArgTypeTest.proto */
01526 #define __Pyx_ArgTypeTest(obj, type, none_allowed, name, exact)\
01527 ((likely((Py_TYPE(obj) == type) | (none_allowed && (obj == Py_None)))) ? 1 : \
01528 __Pyx_ArgTypeTest(obj, type, name, exact))
01529 static int __Pyx_ArgTypeTest(PyObject *obj, PyTypeObject *type, const char *name, int exact);
01530
01531 /* Import.proto */

```

```

01532 static PyObject * __Pyx_Import(PyObject *name, PyObject *from_list, int level);
01533
01534 /* IncludeStringH.proto */
01535 #include <string.h>
01536
01537 /* PyObject_GenericGetAttrNoDict.proto */
01538 #if CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP && PY_VERSION_HEX < 0x03070000
01539 static CYTHON_INLINE PyObject* __Pyx_PyObject_GenericGetAttrNoDict(PyObject* obj, PyObject*
 attr_name);
01540 #else
01541 #define __Pyx_PyObject_GenericGetAttrNoDict PyObject_GenericGetAttr
01542 #endif
01543
01544 /* PyObject_GenericGetAttr.proto */
01545 #if CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP && PY_VERSION_HEX < 0x03070000
01546 static PyObject* __Pyx_PyObject_GenericGetAttr(PyObject* obj, PyObject* attr_name);
01547 #else
01548 #define __Pyx_PyObject_GenericGetAttr PyObject_GenericGetAttr
01549 #endif
01550
01551 /* SetVTable.proto */
01552 static int __Pyx_SetVtable(PyObject *dict, void *vtable);
01553
01554 /* PyObjectGetAttrStrNoError.proto */
01555 static CYTHON_INLINE PyObject* __Pyx_PyObject_GetAttrStrNoError(PyObject* obj, PyObject* attr_name);
01556
01557 /* SetupReduce.proto */
01558 static int __Pyx_setup_reduce(PyObject* type_obj);
01559
01560 /* BytesEquals.proto */
01561 static CYTHON_INLINE int __Pyx_PyBytes_Equals(PyObject* s1, PyObject* s2, int equals);
01562
01563 /* UnicodeEquals.proto */
01564 static CYTHON_INLINE int __Pyx_PyUnicode_Equals(PyObject* s1, PyObject* s2, int equals);
01565
01566 /* CLineInTraceback.proto */
01567 #ifdef CYTHON_CLINE_IN_TRACEBACK
01568 #define __Pyx_CLineForTraceback(tstate, c_line) (((CYTHON_CLINE_IN_TRACEBACK)) ? c_line : 0)
01569 #else
01570 static int __Pyx_CLineForTraceback(PyThreadState *tstate, int c_line);
01571 #endif
01572
01573 /* CodeObjectCache.proto */
01574 typedef struct {
01575 PyCodeObject* code_object;
01576 int code_line;
01577 } __Pyx_CodeObjectCacheEntry;
01578 struct __Pyx_CodeObjectCache {
01579 int count;
01580 int max_count;
01581 __Pyx_CodeObjectCacheEntry* entries;
01582 };
01583 static struct __Pyx_CodeObjectCache __pyx_code_cache = {0,0,NULL};
01584 static int __pyx_bisect_code_objects(__Pyx_CodeObjectCacheEntry* entries, int count, int code_line);
01585 static PyCodeObject* __pyx_find_code_object(int code_line);
01586 static void __pyx_insert_code_object(int code_line, PyCodeObject* code_object);
01587
01588 /* AddTraceback.proto */
01589 static void __Pyx_AddTraceback(const char *funcname, int c_line,
 int py_line, const char *filename);
01590
01591
01592 /* GCCDiagnostics.proto */
01593 #if defined(__GNUC__) && (__GNUC__ > 4 || (__GNUC__ == 4 && __GNUC_MINOR__ >= 6))
01594 #define __Pyx_HAS_GCC_DIAGNOSTIC
01595 #endif
01596
01597 /* CppExceptionConversion.proto */
01598 #ifndef __Pyx_CppExn2PyErr
01599 #include <new>
01600 #include <typeinfo>
01601 #include <stdexcept>
01602 #include <ios>
01603 static void __Pyx_CppExn2PyErr() {
01604 try {
01605 if (PyErr_Occurred())
01606 ; // let the latest Python exn pass through and ignore the current one
01607 else
01608 throw;
01609 } catch (const std::bad_alloc& exn) {
01610 PyErr_SetString(PyExc_MemoryError, exn.what());
01611 } catch (const std::bad_cast& exn) {
01612 PyErr_SetString(PyExc_TypeError, exn.what());
01613 } catch (const std::bad_typeid& exn) {
01614 PyErr_SetString(PyExc_TypeError, exn.what());
01615 } catch (const std::domain_error& exn) {
01616 PyErr_SetString(PyExc_ValueError, exn.what());
01617 } catch (const std::invalid_argument& exn) {

```



```

01618 PyErr_SetString(PyExc_ValueError, exn.what());
01619 } catch (const std::ios_base::failure& exn) {
01620 PyErr_SetString(PyExc_IOError, exn.what());
01621 } catch (const std::out_of_range& exn) {
01622 PyErr_SetString(PyExc_IndexError, exn.what());
01623 } catch (const std::overflow_error& exn) {
01624 PyErr_SetString(PyExc_OverflowError, exn.what());
01625 } catch (const std::range_error& exn) {
01626 PyErr_SetString(PyExc_ArithmeticError, exn.what());
01627 } catch (const std::underflow_error& exn) {
01628 PyErr_SetString(PyExc_ArithmeticError, exn.what());
01629 } catch (const std::exception& exn) {
01630 PyErr_SetString(PyExc_RuntimeError, exn.what());
01631 }
01632 catch (...)
01633 {
01634 PyErr_SetString(PyExc_RuntimeError, "Unknown exception");
01635 }
01636 }
01637 #endif
01638
01639 /* CIntFromPy.proto */
01640 static CYTHON_INLINE int __Pyx_PyInt_As_int(PyObject *);
01641
01642 /* CIntToPy.proto */
01643 static CYTHON_INLINE PyObject* __Pyx_PyInt_From_int(int value);
01644
01645 /* CIntToPy.proto */
01646 static CYTHON_INLINE PyObject* __Pyx_PyInt_From_long(long value);
01647
01648 /* CIntFromPy.proto */
01649 static CYTHON_INLINE long __Pyx_PyInt_As_long(PyObject *);
01650
01651 /* FastTypeChecks.proto */
01652 #if CYTHON_COMPILING_IN_CPYTHON
01653 #define __Pyx_TypeCheck(obj, type) __Pyx_IsSubtype(Py_TYPE(obj), (PyTypeObject *)type)
01654 static CYTHON_INLINE int __Pyx_IsSubtype(PyTypeObject *a, PyTypeObject *b);
01655 static CYTHON_INLINE int __Pyx_PyErr_GivenExceptionMatches(PyObject *err, PyObject *type);
01656 static CYTHON_INLINE int __Pyx_PyErr_GivenExceptionMatches2(PyObject *err, PyObject *typel, PyObject
 *type2);
01657 #else
01658 #define __Pyx_TypeCheck(obj, type) PyObject_TypeCheck(obj, (PyTypeObject *)type)
01659 #define __Pyx_PyErr_GivenExceptionMatches(err, type) PyErr_GivenExceptionMatches(err, type)
01660 #define __Pyx_PyErr_GivenExceptionMatches2(err, typel, type2) (PyErr_GivenExceptionMatches(err, typel)
 || PyErr_GivenExceptionMatches(err, type2))
01661 #endif
01662 #define __Pyx_PyException_Check(obj) __Pyx_TypeCheck(obj, PyExc_Exception)
01663
01664 /* FetchCommonType.proto */
01665 static PyTypeObject* __Pyx_FetchCommonType(PyTypeObject* type);
01666
01667 /* PyObjectGetMethod.proto */
01668 static int __Pyx_PyObject_GetMethod(PyObject *obj, PyObject *name, PyObject **method);
01669
01670 /* PyObjectCallMethod1.proto */
01671 static PyObject* __Pyx_PyObject_CallMethod1(PyObject* obj, PyObject* method_name, PyObject* arg);
01672
01673 /* CoroutineBase.proto */
01674 typedef PyObject *(*__pyx_coroutine_body_t)(PyObject *, PyThreadState *, PyObject *);
01675 #if CYTHON_USE_EXC_INFO_STACK
01676 #define __Pyx_ExcInfoStruct __PyErr_StackItem
01677 #else
01678 typedef struct {
01679 PyObject *exc_type;
01680 PyObject *exc_value;
01681 PyObject *exc_traceback;
01682 } __Pyx_ExcInfoStruct;
01683 #endif
01684 typedef struct {
01685 PyObject_HEAD
01686 __pyx_coroutine_body_t body;
01687 PyObject *closure;
01688 __Pyx_ExcInfoStruct gi_exc_state;
01689 PyObject *gi_weakreflist;
01690 PyObject *classobj;
01691 PyObject *yieldfrom;
01692 PyObject *gi_name;
01693 PyObject *gi_qualname;
01694 PyObject *gi_modulename;
01695 PyObject *gi_code;
01696 PyObject *gi_frame;
01697 int resume_label;
01698 char is_running;
01699 } __pyx_CoroutineObject;
01700 static __pyx_CoroutineObject * __Pyx__Coroutine_New(
01701 PyTypeObject *type, __pyx_coroutine_body_t body, PyObject *code, PyObject *closure,
01702 PyObject *name, PyObject *qualname, PyObject *module_name);

```



```

01703 static __pyx_CoroutineObject *__Pyx__Coroutine_NewInit (
01704 __pyx_CoroutineObject *gen, __pyx_coroutine_body_t body, PyObject *code, PyObject
 *closure,
01705 PyObject *name, PyObject *qualname, PyObject *module_name);
01706 static CYTHON_INLINE void __Pyx_Coroutine_ExceptionClear(__Pyx_ExcInfoStruct *self);
01707 static int __Pyx_Coroutine_Clear(PyObject *self);
01708 static PyObject *__Pyx_Coroutine_Send(PyObject *self, PyObject *value);
01709 static PyObject *__Pyx_Coroutine_Close(PyObject *self);
01710 static PyObject *__Pyx_Coroutine_Throw(PyObject *gen, PyObject *args);
01711 #if CYTHON_USE_EXC_INFO_STACK
01712 #define __Pyx_Coroutine_SwapException(self)
01713 #define __Pyx_Coroutine_ResetAndClearException(self)
 __Pyx_Coroutine_ExceptionClear(&(self)->gi_exc_state)
01714 #else
01715 #define __Pyx_Coroutine_SwapException(self) {\
01716 __Pyx_ExceptionSwap(&(self)->gi_exc_state.exc_type, &(self)->gi_exc_state.exc_value,
 &(self)->gi_exc_state.exc_traceback);\
01717 __Pyx_Coroutine_ResetFrameBackpointer(&(self)->gi_exc_state);\
01718 }
01719 #define __Pyx_Coroutine_ResetAndClearException(self) {\
01720 __Pyx_ExceptionReset((self)->gi_exc_state.exc_type, (self)->gi_exc_state.exc_value,
 (self)->gi_exc_state.exc_traceback);\
01721 (self)->gi_exc_state.exc_type = (self)->gi_exc_state.exc_value =
 (self)->gi_exc_state.exc_traceback = NULL;\
01722 }
01723 #endif
01724 #if CYTHON_FAST_THREAD_STATE
01725 #define __Pyx_PyGen_FetchStopIterationValue(pvalue)\
01726 __Pyx_PyGen__FetchStopIterationValue(__pyx_tstate, pvalue)
01727 #else
01728 #define __Pyx_PyGen_FetchStopIterationValue(pvalue)\
01729 __Pyx_PyGen__FetchStopIterationValue(__Pyx_PyThreadState_Current, pvalue)
01730 #endif
01731 static int __Pyx_PyGen__FetchStopIterationValue(PyThreadState *tstate, PyObject **pvalue);
01732 static CYTHON_INLINE void __Pyx_Coroutine_ResetFrameBackpointer(__Pyx_ExcInfoStruct *exc_state);
01733
01734 /* PatchModuleWithCoroutine.proto */
01735 static PyObject* __Pyx_Coroutine_patch_module(PyObject* module, const char* py_code);
01736
01737 /* PatchGeneratorABC.proto */
01738 static int __Pyx_patch_abc(void);
01739
01740 /* Generator.proto */
01741 #define __Pyx_Generator_USED
01742 static PyTypeObject *__pyx_GeneratorType = 0;
01743 #define __Pyx_Generator_CheckExact(obj) (Py_TYPE(obj) == __pyx_GeneratorType)
01744 #define __Pyx_Generator_New(body, code, closure, name, qualname, module_name)\
01745 __Pyx__Coroutine_New(__pyx_GeneratorType, body, code, closure, name, qualname, module_name)
01746 static PyObject *__Pyx_Generator_Next(PyObject *self);
01747 static int __pyx_Generator_init(void);
01748
01749 /* CheckBinaryVersion.proto */
01750 static int __Pyx_check_binary_version(void);
01751
01752 /* InitStrings.proto */
01753 static int __Pyx_InitStrings(__Pyx_StringTabEntry *t);
01754
01755 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_9index_set_wrap(struct __pyx_obj_8PyClical_index_set
 __pyx_v_self, IndexSet __pyx_v_other); / proto*/
01756 static CYTHON_INLINE IndexSet __pyx_f_8PyClical_9index_set_unwrap(struct __pyx_obj_8PyClical_index_set
 __pyx_v_self); / proto*/
01757 static PyObject *__pyx_f_8PyClical_9index_set_copy(struct __pyx_obj_8PyClical_index_set *__pyx_v_self,
 int __pyx_skip_dispatch); /* proto*/
01758 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_8clifford_wrap(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self, Clifford __pyx_v_other); / proto*/
01759 static CYTHON_INLINE Clifford __pyx_f_8PyClical_8clifford_unwrap(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto*/
01760 static PyObject *__pyx_f_8PyClical_8clifford_copy(struct __pyx_obj_8PyClical_clifford *__pyx_v_self,
 int __pyx_skip_dispatch); /* proto*/
01761
01762 /* Module declarations from 'libcpp.vector' */
01763
01764 /* Module declarations from 'glucat' */
01765
01766 /* Module declarations from 'libc.string' */
01767
01768 /* Module declarations from 'libcpp.string' */
01769
01770 /* Module declarations from 'PyClical' */
01771 static PyTypeObject *__pyx_ptype_8PyClical_index_set = 0;
01772 static PyTypeObject *__pyx_ptype_8PyClical_clifford = 0;
01773 static PyTypeObject *__pyx_ptype_8PyClical__pyx_scope_struct__iter__ = 0;
01774 static CYTHON_INLINE IndexSet __pyx_f_8PyClical_toIndexSet(PyObject *); /*proto*/
01775 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_compare(PyObject *, PyObject *, int
 __pyx_skip_dispatch); /*proto*/
01776 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_min_neg(PyObject *, int __pyx_skip_dispatch);
 /*proto*/

```

```

01777 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_max_pos(PyObject *, int __pyx_skip_dispatch);
/*proto*/
01778 static CYTHON_INLINE std::vector<scalar_t> __pyx_f_8PyClical_list_to_vector(PyObject *); /*proto*/
01779 static CYTHON_INLINE Clifford __pyx_f_8PyClical_toClifford(PyObject *); /*proto*/
01780 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_error_squared_tol(PyObject *, int
__pyx_skip_dispatch); /*proto*/
01781 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_error_squared(PyObject *, PyObject *, PyObject *, int
__pyx_skip_dispatch); /*proto*/
01782 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_approx_equal(PyObject *, PyObject *, int
__pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_approx_equal * __pyx_optional_args); /*proto*/
01783 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_inv(PyObject *, int __pyx_skip_dispatch); /*proto*/
01784 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_scalar(PyObject *, int __pyx_skip_dispatch);
/*proto*/
01785 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_real(PyObject *, int __pyx_skip_dispatch); /*proto*/
01786 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_imag(PyObject *, int __pyx_skip_dispatch); /*proto*/
01787 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_pure(PyObject *, int __pyx_skip_dispatch); /*proto*/
01788 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_even(PyObject *, int __pyx_skip_dispatch); /*proto*/
01789 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_odd(PyObject *, int __pyx_skip_dispatch); /*proto*/
01790 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_involute(PyObject *, int __pyx_skip_dispatch);
/*proto*/
01791 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_reverse(PyObject *, int __pyx_skip_dispatch);
/*proto*/
01792 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_conj(PyObject *, int __pyx_skip_dispatch); /*proto*/
01793 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_quad(PyObject *, int __pyx_skip_dispatch); /*proto*/
01794 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_norm(PyObject *, int __pyx_skip_dispatch); /*proto*/
01795 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_abs(PyObject *, int __pyx_skip_dispatch); /*proto*/
01796 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_max_abs(PyObject *, int __pyx_skip_dispatch);
/*proto*/
01797 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_pow(PyObject *, PyObject *, int __pyx_skip_dispatch);
/*proto*/
01798 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_outer_pow(PyObject *, PyObject *, int
__pyx_skip_dispatch); /*proto*/
01799 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_complexifier(PyObject *, int __pyx_skip_dispatch);
/*proto*/
01800 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_sqrt(PyObject *, int __pyx_skip_dispatch, struct
__pyx_opt_args_8PyClical_sqrt * __pyx_optional_args); /*proto*/
01801 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_exp(PyObject *, int __pyx_skip_dispatch); /*proto*/
01802 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_log(PyObject *, int __pyx_skip_dispatch, struct
__pyx_opt_args_8PyClical_log * __pyx_optional_args); /*proto*/
01803 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_cos(PyObject *, int __pyx_skip_dispatch, struct
__pyx_opt_args_8PyClical_cos * __pyx_optional_args); /*proto*/
01804 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_acos(PyObject *, int __pyx_skip_dispatch, struct
__pyx_opt_args_8PyClical_acos * __pyx_optional_args); /*proto*/
01805 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_cosh(PyObject *, int __pyx_skip_dispatch); /*proto*/
01806 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_acosh(PyObject *, int __pyx_skip_dispatch, struct
__pyx_opt_args_8PyClical_acosh * __pyx_optional_args); /*proto*/
01807 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_sin(PyObject *, int __pyx_skip_dispatch, struct
__pyx_opt_args_8PyClical_sin * __pyx_optional_args); /*proto*/
01808 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_asin(PyObject *, int __pyx_skip_dispatch, struct
__pyx_opt_args_8PyClical_asin * __pyx_optional_args); /*proto*/
01809 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_sinh(PyObject *, int __pyx_skip_dispatch); /*proto*/
01810 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_asinh(PyObject *, int __pyx_skip_dispatch, struct
__pyx_opt_args_8PyClical_asinh * __pyx_optional_args); /*proto*/
01811 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_tan(PyObject *, int __pyx_skip_dispatch, struct
__pyx_opt_args_8PyClical_tan * __pyx_optional_args); /*proto*/
01812 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_atan(PyObject *, int __pyx_skip_dispatch, struct
__pyx_opt_args_8PyClical_atan * __pyx_optional_args); /*proto*/
01813 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_tanh(PyObject *, int __pyx_skip_dispatch); /*proto*/
01814 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_atanh(PyObject *, int __pyx_skip_dispatch, struct
__pyx_opt_args_8PyClical_atanh * __pyx_optional_args); /*proto*/
01815 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_random_clifford(struct __pyx_obj_8PyClical_index_set
*, int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_random_clifford * __pyx_optional_args);
/*proto*/
01816 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_cga3(PyObject *, int __pyx_skip_dispatch); /*proto*/
01817 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_cga3std(PyObject *, int __pyx_skip_dispatch);
/*proto*/
01818 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_agc3(PyObject *, int __pyx_skip_dispatch); /*proto*/
01819 static CYTHON_INLINE PyObject * __pyx_convert_PyObject_string_to_py_std_in_string(std::string const
&); /*proto*/
01820 static CYTHON_INLINE PyObject * __pyx_convert_PyUnicode_string_to_py_std_in_string(std::string const
&); /*proto*/
01821 static CYTHON_INLINE PyObject * __pyx_convert_PyStr_string_to_py_std_in_string(std::string const &);
/*proto*/
01822 static CYTHON_INLINE PyObject * __pyx_convert_PyBytes_string_to_py_std_in_string(std::string const &);
/*proto*/
01823 static CYTHON_INLINE PyObject * __pyx_convert_PyByteArray_string_to_py_std_in_string(std::string const
&); /*proto*/
01824 #define __Pyx_MODULE_NAME "PyClical"
01825 extern int __pyx_module_is_main_PyClical;
01826 int __pyx_module_is_main_PyClical = 0;
01827
01828 /* Implementation of 'PyClical' */
01829 static PyObject * __pyx_builtin_IndexError;
01830 static PyObject * __pyx_builtin_RuntimeError;
01831 static PyObject * __pyx_builtin_TypeError;
01832 static PyObject * __pyx_builtin_ValueError;
01833 static PyObject * __pyx_builtin_NotImplemented;

```

```

01834 static PyObject *__pyx_builtin_range;
01835 static PyObject *__pyx_builtin_xrange;
01836 static const char __pyx_k_[] = ".";
01837 static const char __pyx_k_e[] = "e";
01838 static const char __pyx_k_i[] = "i";
01839 static const char __pyx_k_m[] = "m";
01840 static const char __pyx_k_p[] = "p";
01841 static const char __pyx_k_q[] = "q";
01842 static const char __pyx_k_2[] = " ";
01843 static const char __pyx_k_5[] = ":";
01844 static const char __pyx_k_6[] = "\n\t";
01845 static const char __pyx_k_7[] = "(";
01846 static const char __pyx_k_8[] = ", ";
01847 static const char __pyx_k_9[] = ").";
01848 static const char __pyx_k_cl[] = "cl";
01849 static const char __pyx_k_pi[] = "pi";
01850 static const char __pyx_k_abc[] = "abc";
01851 static const char __pyx_k_cos[] = "cos";
01852 static const char __pyx_k_exp[] = "exp";
01853 static const char __pyx_k_frm[] = "frm";
01854 static const char __pyx_k_inv[] = "inv";
01855 static const char __pyx_k_ist[] = "ist";
01856 static const char __pyx_k_ixt[] = "ixt";
01857 static const char __pyx_k_lhs[] = "lhs";
01858 static const char __pyx_k_log[] = "log";
01859 static const char __pyx_k_max[] = "max";
01860 static const char __pyx_k_min[] = "min";
01861 static const char __pyx_k_obj[] = "obj";
01862 static const char __pyx_k_odd[] = "odd";
01863 static const char __pyx_k_pow[] = "pow";
01864 static const char __pyx_k_rhs[] = "rhs";
01865 static const char __pyx_k_sin[] = "sin";
01866 static const char __pyx_k_tan[] = "tan";
01867 static const char __pyx_k_tau[] = "tau";
01868 static const char __pyx_k_tol[] = "tol";
01869 static const char __pyx_k_Real[] = "Real";
01870 static const char __pyx_k_acos[] = "acos";
01871 static const char __pyx_k_args[] = "args";
01872 static const char __pyx_k_asin[] = "asin";
01873 static const char __pyx_k_atan[] = "atan";
01874 static const char __pyx_k_conj[] = "conj";
01875 static const char __pyx_k_copy[] = "copy";
01876 static const char __pyx_k_cosh[] = "cosh";
01877 static const char __pyx_k_even[] = "even";
01878 static const char __pyx_k_fill[] = "fill";
01879 static const char __pyx_k_from[] = " from ";
01880 static const char __pyx_k_iter[] = "__iter__";
01881 static const char __pyx_k_main[] = "__main__";
01882 static const char __pyx_k_math[] = "math";
01883 static const char __pyx_k_name[] = "__name__";
01884 static const char __pyx_k_norm[] = "norm";
01885 static const char __pyx_k_pure[] = "pure";
01886 static const char __pyx_k_quad[] = "quad";
01887 static const char __pyx_k_send[] = "send";
01888 static const char __pyx_k_sinh[] = "sinh";
01889 static const char __pyx_k_sqrt[] = "sqrt";
01890 static const char __pyx_k_tanh[] = "tanh";
01891 static const char __pyx_k_test[] = "_test";
01892 static const char __pyx_k_UTF_8[] = "UTF-8";
01893 static const char __pyx_k_acosh[] = "acosh";
01894 static const char __pyx_k_asinh[] = "asinh";
01895 static const char __pyx_k_atanh[] = "atanh";
01896 static const char __pyx_k_close[] = "close";
01897 static const char __pyx_k_grade[] = "grade";
01898 static const char __pyx_k_istpq[] = "istpq";
01899 static const char __pyx_k_nbar3[] = "nbar3";
01900 static const char __pyx_k_ninf3[] = "ninf3";
01901 static const char __pyx_k_other[] = "other";
01902 static const char __pyx_k_range[] = "range";
01903 static const char __pyx_k_throw[] = "throw";
01904 static const char __pyx_k_using[] = " using (";
01905 static const char __pyx_k_utf_8[] = "utf-8";
01906 static const char __pyx_k_value[] = " value ";
01907 static const char __pyx_k_encode[] = "encode";
01908 static const char __pyx_k_import[] = "__import__";
01909 static const char __pyx_k_reduce[] = "__reduce__";
01910 static const char __pyx_k_scalar[] = "scalar";
01911 static const char __pyx_k_test_2[] = "__test__";
01912 static const char __pyx_k_xrange[] = "xrange";
01913 static const char __pyx_k_doctest[] = "doctest";
01914 static const char __pyx_k_invalid[] = " invalid ";
01915 static const char __pyx_k_numbers[] = "numbers";
01916 static const char __pyx_k_reverse[] = "reverse";
01917 static const char __pyx_k_testmod[] = "testmod";
01918 static const char __pyx_k_version[] = "__version__";
01919 static const char __pyx_k_Integral[] = "Integral";
01920 static const char __pyx_k_PyClical[] = "PyClical";

```

```

01921 static const char __pyx_k_Sequence[] = "Sequence";
01922 static const char __pyx_k_as_frame[] = " as frame:\n\t";
01923 static const char __pyx_k_clifford[] = "clifford";
01924 static const char __pyx_k_getstate[] = "__getstate__";
01925 static const char __pyx_k_involute[] = "involute";
01926 static const char __pyx_k_setstate[] = "__setstate__";
01927 static const char __pyx_k_to_frame[] = " to frame ";
01928 static const char __pyx_k_TypeError[] = "TypeError";
01929 static const char __pyx_k_index_set[] = "index_set";
01930 static const char __pyx_k_outer_pow[] = "outer_pow";
01931 static const char __pyx_k_reduce_ex[] = "__reduce_ex__";
01932 static const char __pyx_k_threshold[] = "threshold";
01933 static const char __pyx_k_IndexError[] = "IndexError";
01934 static const char __pyx_k_ValueError[] = "ValueError";
01935 static const char __pyx_k_pyx_vtable[] = "__pyx_vtable__";
01936 static const char __pyx_k_collections[] = "collections";
01937 static const char __pyx_k_e_line_1936[] = "e (line 1936)";
01938 static const char __pyx_k_PyClical_pyx[] = "PyClical.pyx";
01939 static const char __pyx_k_RuntimeError[] = "RuntimeError";
01940 static const char __pyx_k_abs_line_1522[] = "abs (line 1522)";
01941 static const char __pyx_k_cos_line_1651[] = "cos (line 1651)";
01942 static const char __pyx_k_exp_line_1614[] = "exp (line 1614)";
01943 static const char __pyx_k_inv_line_1378[] = "inv (line 1378)";
01944 static const char __pyx_k_log_line_1628[] = "log (line 1628)";
01945 static const char __pyx_k_odd_line_1446[] = "odd (line 1446)";
01946 static const char __pyx_k_pow_line_1543[] = "pow (line 1543)";
01947 static const char __pyx_k_reduce_cython[] = "__reduce_cython__";
01948 static const char __pyx_k_sin_line_1728[] = "sin (line 1728)";
01949 static const char __pyx_k_tan_line_1801[] = "tan (line 1801)";
01950 static const char __pyx_k_using_invalid[] = " using invalid ";
01951 static const char __pyx_k_Cannot_reframe[] = "Cannot reframe";
01952 static const char __pyx_k_NotImplemented[] = "NotImplemented";
01953 static const char __pyx_k_Not_applicable[] = "Not applicable.";
01954 static const char __pyx_k_acos_line_1668[] = "acos (line 1668)";
01955 static const char __pyx_k_agc3_line_1893[] = "agc3 (line 1893)";
01956 static const char __pyx_k_asin_line_1747[] = "asin (line 1747)";
01957 static const char __pyx_k_atan_line_1818[] = "atan (line 1818)";
01958 static const char __pyx_k_cga3_line_1873[] = "cga3 (line 1873)";
01959 static const char __pyx_k_conj_line_1485[] = "conj (line 1485)";
01960 static const char __pyx_k_cosh_line_1689[] = "cosh (line 1689)";
01961 static const char __pyx_k_even_line_1437[] = "even (line 1437)";
01962 static const char __pyx_k_imag_line_1415[] = "imag (line 1415)";
01963 static const char __pyx_k_invalid_string[] = " invalid string ";
01964 static const char __pyx_k_norm_line_1511[] = "norm (line 1511)";
01965 static const char __pyx_k_pure_line_1426[] = "pure (line 1426)";
01966 static const char __pyx_k_quad_line_1500[] = "quad (line 1500)";
01967 static const char __pyx_k_real_line_1404[] = "real (line 1404)";
01968 static const char __pyx_k_scalar_epsilon[] = "scalar_epsilon";
01969 static const char __pyx_k_sinh_line_1768[] = "sinh (line 1768)";
01970 static const char __pyx_k_sqrt_line_1591[] = "sqrt (line 1591)";
01971 static const char __pyx_k_tanh_line_1835[] = "tanh (line 1835)";
01972 static const char __pyx_k_acosh_line_1705[] = "acosh (line 1705)";
01973 static const char __pyx_k_asinh_line_1782[] = "asinh (line 1782)";
01974 static const char __pyx_k_atanh_line_1847[] = "atanh (line 1847)";
01975 static const char __pyx_k_istpq_line_1949[] = "istpq (line 1949)";
01976 static const char __pyx_k_setstate_cython[] = "__setstate_cython__";
01977 static const char __pyx_k_compare_line_492[] = "compare (line 492)";
01978 static const char __pyx_k_index_set__iter[] = "index_set.__iter__";
01979 static const char __pyx_k_max_pos_line_513[] = "max_pos (line 513)";
01980 static const char __pyx_k_min_neg_line_504[] = "min_neg (line 504)";
01981 static const char __pyx_k_scalar_line_1393[] = "scalar (line 1393)";
01982 static const char __pyx_k_cga3std_line_1882[] = "cga3std (line 1882)";
01983 static const char __pyx_k_max_abs_line_1531[] = "max_abs (line 1531)";
01984 static const char __pyx_k_reverse_line_1470[] = "reverse (line 1470)";
01985 static const char __pyx_k_cline_in_traceback[] = "cline_in_traceback";
01986 static const char __pyx_k_involute_line_1455[] = "involute (line 1455)";
01987 static const char __pyx_k_outer_pow_line_1567[] = "outer_pow (line 1567)";
01988 static const char __pyx_k_clifford_inv_line_926[] = "clifford.inv (line 926)";
01989 static const char __pyx_k_clifford_pow_line_980[] = "clifford.pow (line 980)";
01990 static const char __pyx_k_approx_equal_line_1359[] = "approx_equal (line 1359)";
01991 static const char __pyx_k_clifford_abs_line_1175[] = "clifford.abs (line 1175)";
01992 static const char __pyx_k_clifford_copy_line_556[] = "clifford.copy (line 556)";
01993 static const char __pyx_k_clifford_odd_line_1070[] = "clifford.odd (line 1070)";
01994 static const char __pyx_k_complexifier_line_1576[] = "complexifier (line 1576)";
01995 static const char __pyx_k_index_set_copy_line_65[] = "index_set.copy (line 65)";
01996 static const char __pyx_k_index_set_max_line_351[] = "index_set.max (line 351)";
01997 static const char __pyx_k_index_set_min_line_342[] = "index_set.min (line 342)";
01998 static const char __pyx_k_clifford_conj_line_1138[] = "clifford.conj (line 1138)";
01999 static const char __pyx_k_clifford_even_line_1061[] = "clifford.even (line 1061)";
02000 static const char __pyx_k_clifford_norm_line_1164[] = "clifford.norm (line 1164)";
02001 static const char __pyx_k_clifford_pure_line_1050[] = "clifford.pure (line 1050)";
02002 static const char __pyx_k_clifford_quad_line_1153[] = "clifford.quad (line 1153)";
02003 static const char __pyx_k_error_squared_line_1346[] = "error_squared (line 1346)";
02004 static const char __pyx_k_Unary_print_clifford_1_1[] = "\n Unary -. \n\n >>
print(-clifford(\n {1}\n)\n -{1}\n ");
02005 static const char __pyx_k_clifford__or__line_939[] = "clifford.__or__ (line 939)";
02006 static const char __pyx_k_clifford_frame_line_1224[] = "clifford.frame (line 1224)";

```

```

02007 static const char __pyx_k_clifford_hidden_doctests[] = "clifford_hidden_doctests";
02008 static const char __pyx_k_clifford_isinf_line_1206[] = "clifford.isinf (line 1206)";
02009 static const char __pyx_k_clifford_isnan_line_1215[] = "clifford.isnan (line 1215)";
02010 static const char __pyx_k_index_set_count_line_315[] = "index_set.count (line 315)";
02011 static const char __pyx_k_clifford__add_line_740[] = "clifford.__add__ (line 740)";
02012 static const char __pyx_k_clifford__and_line_836[] = "clifford.__and__ (line 836)";
02013 static const char __pyx_k_clifford__ior_line_950[] = "clifford.__ior__ (line 950)";
02014 static const char __pyx_k_clifford__mod_line_806[] = "clifford.__mod__ (line 806)";
02015 static const char __pyx_k_clifford__mul_line_780[] = "clifford.__mul__ (line 780)";
02016 static const char __pyx_k_clifford__neg_line_722[] = "clifford.__neg__ (line 722)";
02017 static const char __pyx_k_clifford__pos_line_731[] = "clifford.__pos__ (line 731)";
02018 static const char __pyx_k_clifford__pow_line_961[] = "clifford.__pow__ (line 961)";
02019 static const char __pyx_k_clifford__sub_line_760[] = "clifford.__sub__ (line 760)";
02020 static const char __pyx_k_clifford__xor_line_866[] = "clifford.__xor__ (line 866)";
02021 static const char __pyx_k_clifford_reframe_line_649[] = "clifford.reframe (line 649)";
02022 static const char __pyx_k_clifford_scalar_line_1039[] = "clifford.scalar (line 1039)";
02023 static const char __pyx_k_index_set__or_line_293[] = "index_set.__or__ (line 293)";
02024 static const char __pyx_k_index_set_hidden_doctests[] = "index_set_hidden_doctests";
02025 static const char __pyx_k_random_clifford_line_1864[] = "random_clifford (line 1864)";
02026 static const char __pyx_k_Cannot_take_vector_part_of[] = "Cannot take vector part of ";
02027 static const char __pyx_k_Unary_print_clifford_1_1_2[] = "\n Unary +.\n\n\n" >>>
 print(+clifford("\{1\}"))\n {1}\n ";
02028 static const char __pyx_k_clifford__iadd_line_751[] = "clifford.__iadd__ (line 751)";
02029 static const char __pyx_k_clifford__iand_line_851[] = "clifford.__iand__ (line 851)";
02030 static const char __pyx_k_clifford__idiv_line_911[] = "clifford.__idiv__ (line 911)";
02031 static const char __pyx_k_clifford__imod_line_821[] = "clifford.__imod__ (line 821)";
02032 static const char __pyx_k_clifford__imul_line_793[] = "clifford.__imul__ (line 793)";
02033 static const char __pyx_k_clifford__isub_line_771[] = "clifford.__isub__ (line 771)";
02034 static const char __pyx_k_clifford__iter_line_638[] = "clifford.__iter__ (line 638)";
02035 static const char __pyx_k_clifford__ixor_line_881[] = "clifford.__ixor__ (line 881)";
02036 static const char __pyx_k_clifford__str_line_1244[] = "clifford.__str__ (line 1244)";
02037 static const char __pyx_k_clifford_max_abs_line_1184[] = "clifford.max_abs (line 1184)";
02038 static const char __pyx_k_clifford_reverse_line_1123[] = "clifford.reverse (line 1123)";
02039 static const char __pyx_k_index_set__and_line_271[] = "index_set.__and__ (line 271)";
02040 static const char __pyx_k_index_set__ior_line_304[] = "index_set.__ior__ (line 304)";
02041 static const char __pyx_k_index_set__str_line_395[] = "index_set.__str__ (line 395)";
02042 static const char __pyx_k_index_set__xor_line_249[] = "index_set.__xor__ (line 249)";
02043 static const char __pyx_k_clifford__call_line_1020[] = "clifford.__call__ (line 1020)";
02044 static const char __pyx_k_clifford__repr_line_1235[] = "clifford.__repr__ (line 1235)";
02045 static const char __pyx_k_clifford_involute_line_1107[] = "clifford.involute (line 1107)";
02046 static const char __pyx_k_error_squared_tol_line_1337[] = "error_squared_tol (line 1337)";
02047 static const char __pyx_k_index_set__iand_line_282[] = "index_set.__iand__ (line 282)";
02048 static const char __pyx_k_index_set__iter_line_229[] = "index_set.__iter__ (line 229)";
02049 static const char __pyx_k_index_set__ixor_line_260[] = "index_set.__ixor__ (line 260)";
02050 static const char __pyx_k_index_set__repr_line_384[] = "index_set.__repr__ (line 384)";
02051 static const char __pyx_k_clifford_outer_pow_line_1004[] = "clifford.outer_pow (line 1004)";
02052 static const char __pyx_k_clifford_truncated_line_1195[] = "clifford.truncated (line 1195)";
02053 static const char __pyx_k_index_set_count_neg_line_324[] = "index_set.count_neg (line 324)";
02054 static const char __pyx_k_index_set_count_pos_line_333[] = "index_set.count_pos (line 333)";
02055 static const char __pyx_k_clifford_getitem_line_707[] = "clifford.__getitem__ (line 707)";
02056 static const char __pyx_k_clifford_truediv_line_896[] = "clifford.__truediv__ (line 896)";
02057 static const char __pyx_k_index_set__invert_line_240[] = "index_set.__invert__ (line 240)";
02058 static const char __pyx_k_Abbreviation_for_index_set_q_p[] = "\n Abbreviation for
index_set((-q, ...p)).\n\n >> print(istpg(2,3))\n {-3,-2,-1,1,2}\n ";
02059 static const char __pyx_k_Conjugation_reverse_o_involute[] = "\n Conjugation, reverse o
involute == involute o reverse.\n\n >> print((clifford("\{1\}")).conj())\n {-1}\n\n >> print((clifford("\{2\}")) * clifford("\{1\}")).conj())\n {1,2}\n\n >>
print((clifford("\{1\}")) * clifford("\{2\}")).conj())\n {-1,2}\n\n >>
print(clifford("\{1+{1}+{1,2}\}").conj())\n 1-{1}-{1,2}\n ";
02060 static const char __pyx_k_Geometric_product_x_clifford_2[] = "\n Geometric product.\n\n
>> x = clifford(2); x *= clifford("\{2\}"); print(x)\n 2{2}\n\n >> x = clifford("\{1\}");
x *= clifford("\{2\}"); print(x)\n {1,2}\n\n >> x = clifford("\{1\}"); x *=
clifford("\{1,2\}"); print(x)\n {2}\n ";
02061 static const char __pyx_k_Geometric_sum_print_clifford_1[] = "\n Geometric sum.\n\n
>> print(clifford(1) + clifford("\{2\}"))\n 1+{2}\n\n >> print(clifford("\{1\}")) +
clifford("\{2\}"))\n {1}+{2}\n ";
02062 static const char __pyx_k_Hyperbolic_sine_of_multivector[] = "\n Hyperbolic sine of
multivector.\n\n >> x=clifford("\{1,2\}")) * pi/2; print(sinh(x))\n {1,2}\n\n >>
x=clifford("\{1,2\}")) * pi/6; print(sinh(x))\n 0.5{1,2}\n ";
02063 static const char __pyx_k_Inner_product_print_clifford_1[] = "\n Inner product.\n\n
>> print(clifford("\{1\}")) & clifford("\{2\}"))\n 0\n\n >> print(clifford(2) &
clifford("\{2\}"))\n 0\n\n >> print(clifford("\{1\}")) & clifford("\{1\}"))\n 1\n
>> print(clifford("\{1\}")) & clifford("\{1,2\}"))\n {2}\n ";
02064 static const char __pyx_k_Inverse_tangent_of_multivector[] = "\n Inverse tangent of multivector
with optional complexifier.\n\n >> s=index_set({1,2,3}); x=clifford("\{1\}");
print(tan(atan(x,s)))\n {1}\n\n >> x=clifford("\{1\}"); print(tan(atan(x)))\n {1}\n ";
02065 static const char __pyx_k_Iterate_over_the_indices_of_an[] = "\n Iterate over the indices of an
index_set.\n\n >> for i in index_set((-3,4,7)):print(i, end=',')\n -3,4,7,\n ";
02066 static const char __pyx_k_Maximum_member_index_set_1_1_2[] = "\n Maximum member.\n\n
>> index_set((-1,1,2)).max()\n 2\n ";
02067 static const char __pyx_k_Maximum_positive_index_or_0_if[] = "\n Maximum positive index, or 0 if
none.\n\n >> max_pos(index_set({1,2}))\n 2\n ";
02068 static const char __pyx_k_Minimum_member_index_set_1_1_2[] = "\n Minimum member.\n\n
>> index_set((-1,1,2)).min()\n -1\n ";
02069 static const char __pyx_k_Minimum_negative_index_or_0_if[] = "\n Minimum negative index, or 0 if
none.\n\n >> min_neg(index_set({1,2}))\n 0\n ";
02070 static const char __pyx_k_Odd_part_of_multivector_sum_of[] = "\n Odd part of multivector, sum

```



```

of odd grade terms.\n\n >> print(clifford("\1+{1}+{1,2}\").odd())\n {1}\n";
02071 static const char __pyx_k_Outer_product_power_x_clifford[] = "\n Outer product power.\n\n
>> x=clifford("\2+{1}\"); print(x.outer_pow(0))\n 1\n >> x=clifford("\2+{1}\");
print(x.outer_pow(1))\n 2+{1}\n >> x=clifford("\2+{1}\"); print(x.outer_pow(2))\n
4+4{1}\n >> print(clifford("\1+{1}+{1,2}\").outer_pow(3))\n 1+3{1}+3{1,2}\n\n ";
02072 static const char __pyx_k_Outer_product_print_clifford_1[] = "\n Outer product.\n\n
print(clifford("\{1}\") ^ clifford("\{2}\"))\n {1,2}\n >> print(clifford(2) ^
clifford("\{2}\"))\n 2{2}\n >> print(clifford("\{1}\") ^ clifford("\{1}\"))\n 0\n
>> print(clifford("\{1}\") ^ clifford("\{1,2}\"))\n 0\n ";
02073 static const char __pyx_k_Power_self_to_the_m_x_clifford[] = "\n Power: self to the m.\n\n
>> x=clifford("\{1}\"); print(x ** 2)\n 1\n >> x=clifford("\2\"); print(x ** 2)\n
4\n >> x=clifford("\2+{1}\"); print(x ** 0)\n 1\n >> x=clifford("\2+{1}\");
print(x ** 1)\n 2+{1}\n >> x=clifford("\2+{1}\"); print(x ** 2)\n 5+4{1}\n
>> i=clifford("\{1,2}\"); print(exp(pi/2) * (i ** i))\n 1\n ";
02074 static const char __pyx_k_Pure_part_print_clifford_1_1[] = "\n Pure part.\n\n >>
print(clifford("\1+{1}+{1,2}\").pure())\n {1}+{1,2}\n >>
print(clifford("\{1,2}\").pure())\n {1,2}\n ";
02075 static const char __pyx_k_Quadratic_form_rev_x_x_0_print[] = "\n Quadratic form ==
(rev(x)*x)(0).\n\n >> print(clifford("\1+{1}+{1,2}\").quad())\n 3.0\n >>
print(clifford("\1+{-1}+{1,2}+{1,2,3}\").quad())\n 2.0\n ";
02076 static const char __pyx_k_Quadratic_norm_error_tolerance[] = "\n Quadratic norm error tolerance
relative to a specific multivector.\n\n >> print(error_squared_tol(clifford("\{1}\")) * 3.0 -
error_squared_tol(clifford("\{1,1}-2{2}+3{3}\"))) \n 0.0\n ";
02077 static const char __pyx_k_Set_complement_not_print_index[] = "\n Set complement: not.\n\n
>>
print(~index_set({-16,-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}))\n
{-32,-31,-30,-29,-28,-27,-26,-25,-24,-23,-22,-21,-20,-19,-18,-17,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32}\n
";
02078 static const char __pyx_k_Set_union_or_print_index_set_1[] = "\n Set union: or.\n\n >>
print(index_set({1}) | index_set({2}))\n {1,2}\n >> print(index_set({1,2}) |
index_set({2}))\n {1,2}\n ";
02079 static const char __pyx_k_Transform_left_hand_side_using[] = "\n Transform left hand side,
using right hand side as a transformation.\n\n >> x=clifford("\{1,2}\") * pi/2;
y=clifford("\{1}\"); print(y|x)\n {-1}\n >> x=clifford("\{1,2}\") * pi/2;
y=clifford("\{1}\"); print(y|exp(x))\n {-1}\n ";
02080 static const char __pyx_k_clifford_vector_part_line_1079[] = "clifford.vector_part (line 1079)";
02081 static const char __pyx_k_index_set__getitem__line_191[] = "index_set.__getitem__ (line 191)";
02082 static const char __pyx_k_index_set__setitem__line_179[] = "index_set.__setitem__ (line 179)";
02083 static const char __pyx_k_lexicographic_compare_eg_3_4_5[] = "\n \"lexicographic compare\" eg.
{3,4,5} is less than {3,7,8};\n -1 if a<b, +1 if a>b, 0 if a==b.\n\n >>
compare(index_set({1,2}),index_set({-1,3}))\n -1\n >>
compare(index_set({-1,4}),index_set({-1,3}))\n 1\n ";
02084 static const char __pyx_k_Abbreviation_for_clifford_index[] = "\n Abbreviation for
clifford(index_set(obj)).\n\n >> print(e(1))\n {1}\n >> print(e(-1))\n {-1}\n >>
print(e(0))\n 1\n ";
02085 static const char __pyx_k_Absolute_value_of_multivector_m[] = "\n Absolute value of multivector:
multivector 2-norm.\n\n >> abs(clifford("\1+{-1}+{1,2}+{1,2,3}\"))\n 2.0\n ";
02086 static const char __pyx_k_Absolute_value_square_root_of_norm[] = "\n Absolute value: square root
of norm.\n\n >> clifford("\1+{-1}+{1,2}+{1,2,3}\").abs()\n 2.0\n ";
02087 static const char __pyx_k_Cardinality_Number_of_indices_i[] = "\n Cardinality: Number of
indices included in set.\n\n >> index_set({-1,1,2}).count()\n 3\n ";
02088 static const char __pyx_k_Check_if_a_multivector_contains[] = "\n Check if a multivector
contains any infinite values.\n\n >> clifford().isinf()\n False\n ";
02089 static const char __pyx_k_Contraction_print_clifford_1_cl[] = "\n Contraction.\n\n >>
print(clifford("\{1}\") % clifford("\{2}\"))\n 0\n >> print(clifford(2) %
clifford("\{2}\"))\n 2{2}\n >> print(clifford("\{1}\") % clifford("\{1}\"))\n 1\n
>> print(clifford("\{1}\") % clifford("\{1,2}\"))\n {2}\n ";
02090 static const char __pyx_k_Contraction_x_clifford_1_x_clif[] = "\n Contraction.\n\n >> x
= clifford("\{1}\"); x %= clifford("\{2}\"); print(x)\n 0\n >> x = clifford(2); x %=
clifford("\{2}\"); print(x)\n 2{2}\n >> x = clifford("\{1}\"); x %= clifford("\{1}\");
print(x)\n 1\n >> x = clifford("\{1}\"); x %= clifford("\{1,2}\"); print(x)\n
{2}\n ";
02091 static const char __pyx_k_Convert_CGA3_null_vector_to_Euc[] = "\n Convert CGA3 null vector to
Euclidean 3D vector using Doran and Lasenby definition.\n\n >> x=clifford("\2{1}+9{2}+{3}\");
print(agc3(cga3(x)))\n 2{1}+9{2}+{3}\n >> x=clifford("\2{1}+9{2}+{3}\");
print(agc3(cga3(x))-x)\n 0\n ";
02092 static const char __pyx_k_Convert_CGA3_null_vector_to_sta[] = "\n Convert CGA3 null vector to
standard conformal null vector using Doran and Lasenby definition.\n\n >>
x=clifford("\2{1}+9{2}+{3}\"); print(cga3std(cga3(x)))\n 87{-1}+4{1}+18{2}+2{3}+85{4}\n >>
x=clifford("\2{1}+9{2}+{3}\"); print(cga3std(cga3(x))-cga3(x))\n 0\n ";
02093 static const char __pyx_k_Convert_Euclidean_3D_multivector[] = "\n Convert Euclidean 3D multivector
to Conformal Geometric Algebra using Doran and Lasenby definition.\n\n >>
x=clifford("\2{1}+9{2}+{3}\"); print(cga3(x))\n 87{-1}+4{1}+18{2}+2{3}+85{4}\n ";
02094 static const char __pyx_k_Copy_this_clifford_object_x_cli[] = "\n Copy this clifford
object.\n\n >> x=clifford("\1{2}\"); y=x.copy(); print(y)\n {2}\n ";
02095 static const char __pyx_k_Copy_this_index_set_object_s_in[] = "\n Copy this index_set
object.\n\n >> s=index_set(1); t=s.copy(); print(t)\n {1}\n ";
02096 static const char __pyx_k_Cosine_of_multivector_with_opti[] = "\n Cosine of multivector with
optional complexifier.\n\n >> x=clifford("\{1,2}\"); print(cos(acos(x), "\{1,2,3}\"))\n {1,2}\n
>> x=clifford("\{1,2}\"); print(cos(acos(x)))\n {1,2}\n ";
02097 static const char __pyx_k_Even_part_of_multivector_sum_of[] = "\n Even part of multivector, sum
of even grade terms.\n\n >> print(clifford("\1+{1}+{1,2}\").even())\n 1+{1,2}\n ";
02098 static const char __pyx_k_Exponential_of_multivector_x_cl[] = "\n Exponential of multivector.\n\n
>> x=clifford("\{1,2}\") * pi/4; print(exp(x))\n 0.7071+0.7071{1,2}\n >> x=clifford("\{1,2}\") *
pi/2; print(exp(x))\n {1,2}\n ";
02099 static const char __pyx_k_Geometric_difference_print_clif[] = "\n Geometric difference.\n\n

```

```

 >> print(clifford(1) - clifford("\{2}\"))\n 1-{2}\n >> print(clifford("\{1}\") -
 clifford("\{2}\"))\n {1}-{2}\n ";
02100 static const char __pyx_k_Geometric_difference_x_clifford[] = "\n Geometric difference.\n\n
 >> x = clifford(1); x -= clifford("\{2}\"); print(x)\n 1-{2}\n ";
02101 static const char __pyx_k_Geometric_multiplicative_invers[] = "\n Geometric multiplicative
 inverse.\n\n >> x = clifford("\{1}\"); print(x.inv())\n {1}\n >> x = clifford(2);
 print(x.inv())\n 0.5\n >> x = clifford("\{1,2}\"); print(x.inv())\n -{1,2}\n
 ";
02102 static const char __pyx_k_Geometric_product_print_cliffor[] = "\n Geometric product.\n\n
 >> print(clifford("\{1}\") * clifford("\{2}\"))\n {1,2}\n >> print(clifford(2) *
 clifford("\{2}\"))\n 2{2}\n >> print(clifford("\{1}\") * clifford("\{1,2}\"))\n
 {2}\n ";
02103 static const char __pyx_k_Geometric_quotient_print_cliffo[] = "\n Geometric quotient.\n\n
 >> print(clifford("\{1}\") / clifford("\{2}\"))\n {1,2}\n >> print(clifford(2) /
 clifford("\{2}\"))\n 2{2}\n >> print(clifford("\{1}\") / clifford("\{1}\"))\n 1\n
 >> print(clifford("\{1}\") / clifford("\{1,2}\"))\n -{2}\n ";
02104 static const char __pyx_k_Geometric_quotient_x_clifford_1[] = "\n Geometric quotient.\n\n
 >> x = clifford("\{1}\"); x /= clifford("\{2}\"); print(x)\n {1,2}\n >> x = clifford(2);
 x /= clifford("\{2}\"); print(x)\n 2{2}\n >> x = clifford("\{1}\"); x /=
 clifford("\{1}\"); print(x)\n 1\n >> x = clifford("\{1}\"); x /= clifford("\{1,2}\");
 print(x)\n -{2}\n ";
02105 static const char __pyx_k_Geometric_sum_x_clifford_1_x_cl[] = "\n Geometric sum.\n\n >>
 x = clifford(1); x += clifford("\{2}\"); print(x)\n 1+{2}\n ";
02106 static const char __pyx_k_Get_the_value_of_an_index_set_o[] = "\n Get the value of an index_set
 object at an index.\n\n >> index_set({1})[1]\n True\n >> index_set({1})[2]\n
 False\n >> index_set({2})[-1]\n False\n >> index_set({2})[1]\n False\n
 >> index_set({2})[2]\n True\n >> index_set({2})[33]\n False\n ";
02107 static const char __pyx_k_Hyperbolic_cosine_of_multivecto[] = "\n Hyperbolic cosine of
 multivector.\n\n >> x=clifford("\{1,2}\") * pi; print(cosh(x))\n -1\n >>
 x=clifford("\{1,2,3}\"); print(cosh(acosh(x)))\n {1,2,3}\n >> x=clifford("\{1,2}\");
 print(cosh(acosh(x)))\n {1,2}\n ";
02108 static const char __pyx_k_Hyperbolic_tangent_of_multivect[] = "\n Hyperbolic tangent of
 multivector.\n\n >> x=clifford("\{1,2}\") * pi/4; print(tanh(x))\n {1,2}\n ";
02109 static const char __pyx_k_Imaginary_part_deprecated_alway[] = "\n Imaginary part: deprecated
 (always 0).\n\n >> imag(clifford("\{1+{1}+{1,2}\"))\n 0.0\n >> imag(clifford("\{1,2}\"))\n
 0.0\n ";
02110 static const char __pyx_k_Inner_product_x_clifford_1_x_cl[] = "\n Inner product.\n\n >>
 x = clifford("\{1}\"); x &= clifford("\{2}\"); print(x)\n 0\n >> x = clifford(2); x &=
 clifford("\{2}\"); print(x)\n 0\n >> x = clifford("\{1}\"); x &= clifford("\{1}\");
 print(x)\n 1\n >> x = clifford("\{1}\"); x &= clifford("\{1,2}\"); print(x)\n
 {2}\n ";
02111 static const char __pyx_k_Integer_power_of_multivector_ob[] = "\n Integer power of multivector: obj
 to the m.\n\n >> x=clifford("\{1}\"); print(pow(x,2))\n 1\n >> x=clifford("\{2}\");
 print(pow(x,2))\n 4\n >> x=clifford("\{2+{1}\"); print(pow(x,0))\n 1\n >>
 x=clifford("\{2+{1}\"); print(pow(x,1))\n 2+{1}\n >> x=clifford("\{2+{1}\"); print(pow(x,2))\n
 5+4{1}\n >> print(pow(clifford("\{1+{1}+{1,2}\"),3))\n 1+3{1}+3{1,2}\n >>
 i=clifford("\{1,2}\"); print(exp(pi/2) * pow(i, i))\n 1\n ";
02112 static const char __pyx_k_Inverse_cosine_of_multivector_w[] = "\n Inverse cosine of multivector
 with optional complexifier.\n\n >> x=clifford("\{1,2}\"); print(cos(acos(x),"\{1,2,3}\"))\n
 {1,2}\n >> x=clifford("\{1,2}\"); print(cos(acos(x),"\{-1,1,2,3,4}\"))\n {1,2}\n >>
 print(acos(0) / pi)\n 0.5\n >> x=clifford("\{1,2}\"); print(cos(acos(x))\n {1,2}\n
 ";
02113 static const char __pyx_k_Inverse_hyperbolic_cosine_of_mu[] = "\n Inverse hyperbolic cosine of
 multivector with optional complexifier.\n\n >> print(acosh(0,"\{-2,-1,1}\"))\n 1.571{-2,-1,1}\n
 >> x=clifford("\{1,2,3}\"); print(cosh(acosh(x),"\{-1,1,2,3,4}\"))\n {1,2,3}\n >>
 print(acosh(0))\n 1.571{-1}\n >> x=clifford("\{1,2,3}\"); print(cosh(acosh(x))\n {1,2,3}\n
 >> x=clifford("\{1,2}\"); print(cosh(acosh(x))\n {1,2}\n ";
02114 static const char __pyx_k_Inverse_hyperbolic_sine_of_multiv[] = "\n Inverse hyperbolic sine of
 multivector with optional complexifier.\n\n >> x=clifford("\{1,2}\"); print(asinh(x,"\{1,2,3}\") *
 2/pi)\n {1,2}\n >> x=clifford("\{1,2}\"); print(asinh(x) * 2/pi)\n {1,2}\n >>
 x=clifford("\{1,2}\") / 2; print(asinh(x) * 6/pi)\n {1,2}\n ";
02115 static const char __pyx_k_Inverse_hyperbolic_tangent_of_m[] = "\n Inverse hyperbolic tangent of
 multivector with optional complexifier.\n\n >> s=index_set({1,2,3}); x=clifford("\{1,2}\");
 print(tanh(atanh(x,s))\n {1,2}\n >> x=clifford("\{1,2}\"); print(tanh(atanh(x))\n {1,2}\n
 ";
02116 static const char __pyx_k_Inverse_sine_of_multivector_wit[] = "\n Inverse sine of multivector with
 optional complexifier.\n\n >> s="\{-1}\"; x=clifford(s); print(asin(sin(x,s),s))\n {-1}\n >>
 s="\{-1}\"; x=clifford(s); print(asin(sin(x,s),"\{-2,-1,1}\"))\n {-1}\n >> print(asin(1) / pi)\n
 0.5\n >> x=clifford("\{1,2,3}\"); print(asin(sin(x))\n {1,2,3}\n ";
02117 static const char __pyx_k_Main_involution_each_i_is_repla[] = "\n Main involution, each {i} is
 replaced by -{i} in each term,\n eg. clifford("\{1}\") -> -clifford("\{1}\").\n\n >>
 print(clifford("\{1}\").involute())\n -{1}\n >> print((clifford("\{2}\") *
 clifford("\{1}\")).involute())\n -{1,2}\n >> print((clifford("\{1}\") *
 clifford("\{2}\")).involute())\n {1,2}\n >>
 print(clifford("\{1+{1}+{1,2}\").involute())\n 1-{1}+{1,2}\n ";
02118 static const char __pyx_k_Maximum_absolute_value_of_coord[] = "\n Maximum absolute value of
 coordinates multivector: multivector infinity-norm.\n\n >>
 max_abs(clifford("\{1+{-1}+{1,2}+{1,2,3}\"))\n 1.0\n >> max_abs(clifford("\{3+2{1}+{1,2}\"))\n
 3.0\n ";
02119 static const char __pyx_k_Maximum_of_absolute_values_of_c[] = "\n Maximum of absolute values of
 components of multivector: multivector infinity norm.\n\n >>
 clifford("\{1+{-1}+{1,2}+{1,2,3}\").max_abs()\n 1.0\n >>
 clifford("\{3+2{1}+{1,2}\").max_abs()\n 3.0\n ";
02120 static const char __pyx_k_Natural_logarithm_of_multivecto[] = "\n Natural logarithm of multivector
 with optional complexifier.\n\n >> x=clifford("\{-1}\"); print((log(x,"\{-1}\") * 2/pi))\n
 {-1}\n >> x=clifford("\{1,2}\"); print((log(x,"\{1,2,3}\") * 2/pi))\n {1,2}\n >>
 x=clifford("\{1,2}\"); print((log(x) * 2/pi))\n {1,2}\n >> x=clifford("\{1,2}\");
 print((log(x,"\{1,2}\") * 2/pi))\n Traceback (most recent call last):\n ...
 RuntimeError:

```

```

check_complex(val, i): i is not a valid complexifier for val\n ";
02121 static const char __pyx_k_Norm_sum_of_squares_of_coordina[] = "\n Norm == sum of squares of\n coordinates.\n\n >> clifford(\"1+{1}+{1,2}\").norm()\n clifford(\"1+{-1}+{1,2}+{1,2,3}\").norm()\n 4.0\n 3.0\n >>";
02122 static const char __pyx_k_Not_applicable_for_a_in_cliffor[] = "\n Not applicable.\n\n >> for a in clifford(index_set({-3,4,7})): print(a, end=\"\\n\\n\")\n last):\n ...\\n TypeError: Not applicable.\n Traceback (most recent call\n last):\n ...\\n 4.0\n 3.0\n >>";
02123 static const char __pyx_k_Number_of_negative_indices_incl[] = "\n Number of negative indices\n included in set.\n\n >> index_set({-1,1,2}).count_neg()\n 1\n 1\n >>";
02124 static const char __pyx_k_Number_of_positive_indices_incl[] = "\n Number of positive indices\n included in set.\n\n >> index_set({-1,1,2}).count_pos()\n 2\n 2\n >>";
02125 static const char __pyx_k_Outer_product_power_of_multivec[] = "\n Outer product power of\n multivector.\n\n >> print(outer_pow(clifford(\"1+{1}+{1,2}\"),3))\n 1+3{1}+3{1,2}\n 1\n 1\n >>";
02126 static const char __pyx_k_Outer_product_x_clifford_1_x_cl[] = "\n Outer product.\n\n >> x = clifford(\"{1}\"); x ^= clifford(\"{2}\"); print(x)\n {1,2}\n >> x = clifford(2); x\n ^= clifford(\"{2}\"); print(x)\n 2{2}\n >> x = clifford(\"{1}\"); x ^=\n clifford(\"{1}\"); print(x)\n 0\n >> x = clifford(\"{1}\"); x ^= clifford(\"{1,2}\");\n print(x)\n 0\n >>";
02127 static const char __pyx_k_Pure_grade_vector_part_print_cl[] = "\n Pure grade-vector part.\n\n >> print(clifford(\"{1}\")(1))\n {1}\n >> print(clifford(\"{1}\")(0))\n 0\n >> print(clifford(\"1+{1}+{1,2}\")(0))\n 1\n >> print(clifford(\"1+{1}+{1,2}\")(1))\n {1}\n >> print(clifford(\"1+{1}+{1,2}\")(2))\n {1,2}\n >> print(clifford(\"1+{1}+{1,2}\")(3))\n 0\n >>";
02128 static const char __pyx_k_Pure_part_print_pure_clifford_1[] = "\n Pure part\n\n >> print(pure(clifford(\"1+{1}+{1,2}\"))) \n {1}+{1,2}\n >> print(pure(clifford(\"{1,2}\"))) \n {1,2}\n >>";
02129 static const char __pyx_k_Put_self_into_a_larger_frame_co[] = "\n Put self into a larger frame,\n containing the union of self.frame() and index set ixt.\n This can be used to make\n multiplication faster, by multiplying within a common frame.\n\n >> clifford(\"2+3{1}\").reframe(index_set({1,2,3}))\n clifford(\"2+3{1}\")\n >> s=index_set({1,2,3}); t=index_set({-3,-2,-1}); x=random_clifford(s); x.reframe(t).frame() == (s|t);\n True\n >>";
02130 static const char __pyx_k_Random_multivector_within_a_fra[] = "\n Random multivector within a\n frame.\n\n >> print(random_clifford(index_set({-3,-1,2})).frame())\n {-3,-1,2}\n >>";
02131 static const char __pyx_k_Real_part_synonym_for_scalar_pa[] = "\n Real part: synonym for scalar\n part.\n\n >> real(clifford(\"1+{1}+{1,2}\"))\n 1.0\n >> real(clifford(\"{1,2}\"))\n 0.0\n >>";
02132 static const char __pyx_k_Relative_or_absolute_error_usin[] = "\n Relative or absolute error using\n the quadratic norm.\n\n >> err2=sqrt(epsilon)*epsilon\n >> print(error_squared(clifford(\"{1}\"), clifford(\"{1}\"), err2))\n 0.0\n >> print(error_squared(clifford(\"{1}\")+3{2}+4{3}), clifford(\"{1}\"), err2))\n 25.0\n >>";
02133 static const char __pyx_k_Remove_all_terms_of_self_with_r[] = "\n Remove all terms of self with\n relative size smaller than limit.\n\n >> clifford(\"1e8+{1}+1e-8{1,2}\").truncated(1.0e-6)\n clifford(\"1000000000\")\n >> clifford(\"1e4+{1}+1e-4{1,2}\").truncated(1.0e-6)\n clifford(\"10000+{1}\")\n >>";
02134 static const char __pyx_k_Reversion_eg_1_2_2_1_print_reve[] = "\n Reversion, eg. {1}*{2} ->\n {2}*{1}\n\n >> print(reverse(clifford(\"{1}\"))) \n {1}\n >> print(reverse(clifford(\"{2}\") * clifford(\"{1}\"))) \n {1,2}\n >> print(reverse(clifford(\"{1}\") * clifford(\"{2}\"))) \n -{1,2}\n >> print(reverse(clifford(\"1+{1}+{1,2}\"))) \n 1+{1}-{1,2}\n >>";
02135 static const char __pyx_k_Reversion_eg_clifford_1_cliffor[] = "\n Reversion, eg.\n clifford(\"{1}\")*clifford(\"{2}\") -> clifford(\"{2}\")*clifford(\"{1}\")\n\n >> print(clifford(\"{1}\").reverse()) \n {1}\n >> print((clifford(\"{2}\") * clifford(\"{1}\")).reverse()) \n {1,2}\n >> print((clifford(\"{1}\") * clifford(\"{2}\")).reverse()) \n -{1,2}\n >> print(clifford(\"1+{1}+{1,2}\").reverse()) \n 1+{1}-{1,2}\n >>";
02136 static const char __pyx_k_Scalar_part_clifford_1_1_2_sc[] = "\n Scalar part.\n\n >> clifford(\"1+{1}+{1,2}\").scalar()\n 1.0\n >> clifford(\"{1,2}\").scalar()\n 0.0\n >>";
02137 static const char __pyx_k_Scalar_part_scalar_clifford_1_1[] = "\n Scalar part.\n\n >> scalar(clifford(\"1+{1}+{1,2}\")) \n 1.0\n >> scalar(clifford(\"{1,2}\")) \n 0.0\n >>";
02138 static const char __pyx_k_Set_intersection_and_print_inde[] = "\n Set intersection: and.\n\n >> print(index_set({1}) & index_set({2})) \n {2}\n >> print(index_set({1,2}) & index_set({2})) \n {2}\n >>";
02139 static const char __pyx_k_Set_intersection_and_x_index_se[] = "\n Set intersection: and.\n\n >> x = index_set({1}); x &= index_set({2}); print(x)\n {2}\n >> x = index_set({1,2}); x\n &= index_set({2}); print(x)\n {2}\n >>";
02140 static const char __pyx_k_Set_the_value_of_an_index_set_o[] = "\n Set the value of an index_set\n object at index idx to value val.\n\n >> s=index_set({1}); s[2] = True; print(s)\n {1,2}\n >> s=index_set({1,2}); s[1] = False; print(s)\n {2}\n >>";
02141 static const char __pyx_k_Set_union_or_x_index_set_1_x_in[] = "\n Set union: or.\n\n >> x = index_set({1}); x |= index_set({2}); print(x)\n {1,2}\n >> x = index_set({1,2}); x\n |= index_set({2}); print(x)\n {1,2}\n >>";
02142 static const char __pyx_k_Sign_of_geometric_product_of_tw[] = "\n Sign of geometric product of\n two Clifford basis elements.\n\n >> s = index_set({1,2}); t=index_set({-1});\n s.sign_of_mult(t)\n 1\n >>";
02143 static const char __pyx_k_Sign_of_geometric_square_of_a_C[] = "\n Sign of geometric square of a\n Clifford basis element.\n\n >> s = index_set({1,2}); s.sign_of_square()\n -1\n >>";
02144 static const char __pyx_k_Sine_of_multivector_with_option[] = "\n Sine of multivector with optional\n complexifier.\n\n >> s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),s))\n {-1}\n >> s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),\"{-2,-1,1}\"))\n {-1}\n >> x=clifford(\"{1,2,3}\"); print(asin(sin(x)))\n {1,2,3}\n >>";
02145 static const char __pyx_k_Square_root_of_1_which_commut[] = "\n Square root of -1 which commutes\n with all members of the frame of the given multivector.\n\n >> print(complexifier(clifford(index_set({1})))\n {1,2,3}\n >> print(complexifier(clifford(index_set({-1})))\n {-1}\n >> print(complexifier(index_set({1})))\n {1,2,3}\n >> print(complexifier(index_set({-1})))\n {-1}\n >>";
02146 static const char __pyx_k_Square_root_of_multivector_with[] = "\n Square root of multivector with

```



Generated by Doxygen

```

(rev(x)*x)(0).\n\n >> print(quad(clifford("\1+{1}+{1,2}\n"))\n 3.0\n >>
print(quad(clifford("\1+{-1}+{1,2}+{1,2,3}\n"))\n 2.0\n ");
02165 static const char __pyx_k_Transform_left_hand_side_using_2[] = "\n Transform left hand
side, using right hand side as a transformation.\n\n >> x=clifford("\1,2\n") * pi/2;
y=clifford("\1\n"); y|=x; print(y)\n -{1}\n >> x=clifford("\1,2\n") * pi/2;
y=clifford("\1\n"); y|=exp(x); print(y)\n -{1}\n ";
02166 static const char __pyx_k_clifford_hidden_doctests_line_12[] = "clifford_hidden_doctests (line
1253)";
02167 static const char __pyx_k_index_set_hidden_doctests_line_4[] = "index_set_hidden_doctests
(line 406)";
02168 static const char __pyx_k_index_set_sign_of_square_line_37[] = "index_set.sign_of_square (line
375)";
02169 static const char __pyx_k_no_default__reduce__due_to_non[] = "no default __reduce__ due to
non-trivial __cinit__";
02170 static const char __pyx_k_Check_if_a_multivector_contains_2[] = "\n Check if a
multivector contains any IEEE NaN values.\n\n >> clifford().isnan()\n False\n ";
02171 static const char __pyx_k_Even_part_of_multivector_sum_of_2[] = "\n Even part of
multivector, sum of even grade terms.\n\n >> print(even(clifford("\1+{1}+{1,2}\n"))\n 1+{1,2}\n
");
02172 static const char __pyx_k_Geometric_multiplicative_invers_2[] = "\n Geometric
multiplicative inverse.\n\n >> print(inv(clifford("\1\n"))\n {1}\n >>
print(inv(clifford("\{-1}\n"))\n -{1}\n >> print(inv(clifford("\{-2,-1}\n"))\n -{-2,-1}\n
>> print(inv(clifford("\{-1}+{1}\n"))\n nan\n ");
02173 static const char __pyx_k_Main_involution_each_i_is_repla_2[] = "\n Main involution, each
{i} is replaced by -{i} in each term, eg. {1}*{2} -> (-{2})*(-{1})\n\n >>
print(involute(clifford("\1\n"))\n -{1}\n >> print(involute(clifford("\2\n") *
clifford("\1\n"))\n -{1,2}\n >> print(involute(clifford("\1\n") * clifford("\2\n"))\n
{1,2}\n >> print(involute(clifford("\1+{1}+{1,2}\n"))\n 1-{1}+{1,2}\n ";
02174 static const char __pyx_k_Symmetric_set_difference_exclus_2[] = "\n Symmetric set
difference: exclusive or.\n\n >> x = index_set({1}); x ^= index_set({2}); print(x)\n
{1,2}\n >> x = index_set({1,2}); x ^= index_set({2}); print(x)\n {1}\n ";
02175 static const char __pyx_k_Tests_for_functions_that_Doctes_2[] = "\n Tests for functions
that Doctest cannot see.\n\n For clifford.__cinit__: Construct an object of type clifford.\n\n
>> print(clifford(2))\n 2\n >> print(clifford(2.0))\n 2\n >> print(clifford(1.0e-1))\n
0.1\n >> print(clifford("\2\n"))\n 2\n >> print(clifford("\2{1,2,3}\n"))\n 2{1,2,3}\n
>> print(clifford(clifford("\2{1,2,3}\n"))\n 2{1,2,3}\n >> print(clifford("\{-1}\n"))\n -{1}\n
>> print(clifford(2,index_set({1,2}))\n 2{1,2}\n >> print(clifford([2,3],index_set({1,2}))\n
2{1}+3{2}\n >> print(clifford([1,2]))\n Traceback (most recent call last):\n ...
TypeError: Cannot initialize clifford object from <class 'list'>.\n >> print(clifford(None))\n
Traceback (most recent call last):\n ...
TypeError: Cannot initialize clifford object from
<class 'NoneType'>.\n >> print(clifford(None,[1,2]))\n Traceback (most recent call last):\n
...
TypeError: Cannot initialize clifford object from (<class 'list'>, <class 'list'>).\n
>> print(clifford([1,2],[1,2]))\n Traceback (most recent call last):\n ...
TypeError: Cannot initialize clifford object from (<class 'list'>, <class 'list'>).\n
>> print(clifford("\n"))\n Traceback (most recent call last):\n ...
ValueError: Cannot initialize clifford object from invalid string "\n"
>> print(clifford("\{ }\n"))\n Traceback (most recent call last):\n ...
ValueError: Cannot initialize clifford object from invalid string "\{ }\n"
>> print(clifford("\{1}\n"))\n Traceback (most recent call last):\n ...
ValueError: Cannot initialize clifford object from invalid string "\{1}\n"
>> print(clifford("\{1}+\n"))\n Traceback (most recent call last):\n ...
ValueError: Cannot initialize clifford object from invalid string "\{1}+\n"
>> print(clifford("\{-\n"))\n Traceback (most recent call last):\n ...
ValueError: Cannot initialize clifford object fro"m invalid string "\{-\n"
>> print(clifford("\{1}+\n"))\n Traceback (most recent call last):\n ...
ValueError: Cannot initialize clifford object from invalid string "\{1}+\n"
For clifford.__richcmp__: Compare objects of type clifford.\n\n >> clifford("\1\n") ==
clifford("\1{1}\n")\n True\n >> clifford("\1\n") != clifford("\1.0{1}\n")\n False\n >>
clifford("\1\n") != clifford("\1.0\n")\n True\n >> clifford("\1,2\n") == None\n False\n
>> clifford("\1,2\n") != None\n True\n >> None == clifford("\1,2\n")\n False\n >> None
!= clifford("\1,2\n")\n True\n ";
02176 static const char __pyx_k_The_informal_string_representat_2[] = "\n The
\342\200\234informal\342\200\235 string representation of self.\n\n >>
clifford("\1+3{-1}+2{1,2}+4{-2,7}\n").__str__()\n '1+3{-1}+2{1,2}+4{-2,7}\n'
";
02177 static const char __pyx_k_The_official_string_representat_2[] = "\n The
\342\200\234official\342\200\235 string representation of self.\n\n >>
clifford("\1+3{-1}+2{1,2}+4{-2,7}\n").__repr__()\n 'clifford("\1+3{-1}+2{1,2}+4{-2,7}\n")\n
";
02178 static PyObject * __pyx_kp_u_;
02179 static PyObject * __pyx_kp_u_Abbreviation_for_clifford_index;
02180 static PyObject * __pyx_kp_u_Abbreviation_for_index_set_q_p;
02181 static PyObject * __pyx_kp_u_Absolute_value_of_multivector_m;
02182 static PyObject * __pyx_kp_u_Absolute_value_square_root_of_n;
02183 static PyObject * __pyx_kp_u_Cannot_initialize_clifford_objec;
02184 static PyObject * __pyx_kp_u_Cannot_initialize_index_set_obje;
02185 static PyObject * __pyx_kp_u_Cannot_reframe;
02186 static PyObject * __pyx_kp_u_Cannot_take_vector_part_of;
02187 static PyObject * __pyx_kp_u_Cardinality_Number_of_indices_i;
02188 static PyObject * __pyx_kp_u_Check_if_a_multivector_contains;
02189 static PyObject * __pyx_kp_u_Check_if_a_multivector_contains_2;
02190 static PyObject * __pyx_kp_u_Conjugation_reverse_o_involute;
02191 static PyObject * __pyx_kp_u_Conjugation_reverse_o_involute_2;
02192 static PyObject * __pyx_kp_u_Contraction_print_clifford_l_cl;
02193 static PyObject * __pyx_kp_u_Contraction_x_clifford_l_x_clif;
02194 static PyObject * __pyx_kp_u_Convert_CGA3_null_vector_to_Euc;
02195 static PyObject * __pyx_kp_u_Convert_CGA3_null_vector_to_sta;
02196 static PyObject * __pyx_kp_u_Convert_Euclidean_3D_multivecto;
02197 static PyObject * __pyx_kp_u_Copy_this_clifford_object_x_cli;
02198 static PyObject * __pyx_kp_u_Copy_this_index_set_object_s_in;

```

```

02199 static PyObject *__pyx_kp_u_Cosine_of_multivector_with_opti;
02200 static PyObject *__pyx_kp_u_Even_part_of_multivector_sum_of;
02201 static PyObject *__pyx_kp_u_Even_part_of_multivector_sum_of_2;
02202 static PyObject *__pyx_kp_u_Exponential_of_multivector_x_cl;
02203 static PyObject *__pyx_kp_u_Geometric_difference_print_clif;
02204 static PyObject *__pyx_kp_u_Geometric_difference_x_clifford;
02205 static PyObject *__pyx_kp_u_Geometric_multiplicative_invers;
02206 static PyObject *__pyx_kp_u_Geometric_multiplicative_invers_2;
02207 static PyObject *__pyx_kp_u_Geometric_product_print_cliffor;
02208 static PyObject *__pyx_kp_u_Geometric_product_x_clifford_2;
02209 static PyObject *__pyx_kp_u_Geometric_quotient_print_cliffo;
02210 static PyObject *__pyx_kp_u_Geometric_quotient_x_clifford_1;
02211 static PyObject *__pyx_kp_u_Geometric_sum_print_clifford_1;
02212 static PyObject *__pyx_kp_u_Geometric_sum_x_clifford_1_x_cl;
02213 static PyObject *__pyx_kp_u_Get_the_value_of_an_index_set_o;
02214 static PyObject *__pyx_kp_u_Hyperbolic_cosine_of_multivecto;
02215 static PyObject *__pyx_kp_u_Hyperbolic_sine_of_multivector;
02216 static PyObject *__pyx_kp_u_Hyperbolic_tangent_of_multivect;
02217 static PyObject *__pyx_kp_u_Imaginary_part_deprecated_alway;
02218 static PyObject *__pyx_n_s_IndexError;
02219 static PyObject *__pyx_kp_u_Inner_product_print_clifford_1;
02220 static PyObject *__pyx_kp_u_Inner_product_x_clifford_1_x_cl;
02221 static PyObject *__pyx_kp_u_Integer_power_of_multivector_ob;
02222 static PyObject *__pyx_n_s_Integral;
02223 static PyObject *__pyx_kp_u_Inverse_cosine_of_multivector_w;
02224 static PyObject *__pyx_kp_u_Inverse_hyperbolic_cosine_of_mu;
02225 static PyObject *__pyx_kp_u_Inverse_hyperbolic_sine_of_mult;
02226 static PyObject *__pyx_kp_u_Inverse_hyperbolic_tangent_of_m;
02227 static PyObject *__pyx_kp_u_Inverse_sine_of_multivector_wit;
02228 static PyObject *__pyx_kp_u_Inverse_tangent_of_multivector;
02229 static PyObject *__pyx_kp_u_Iterate_over_the_indices_of_an;
02230 static PyObject *__pyx_kp_u_Main_involution_each_i_is_repla;
02231 static PyObject *__pyx_kp_u_Main_involution_each_i_is_repla_2;
02232 static PyObject *__pyx_kp_u_Maximum_absolute_value_of_coord;
02233 static PyObject *__pyx_kp_u_Maximum_member_index_set_1_1_2;
02234 static PyObject *__pyx_kp_u_Maximum_of_absolute_values_of_c;
02235 static PyObject *__pyx_kp_u_Maximum_positive_index_or_0_if;
02236 static PyObject *__pyx_kp_u_Minimum_member_index_set_1_1_2;
02237 static PyObject *__pyx_kp_u_Minimum_negative_index_or_0_if;
02238 static PyObject *__pyx_kp_u_Natural_logarithm_of_multivecto;
02239 static PyObject *__pyx_kp_u_Norm_sum_of_squares_of_coordina;
02240 static PyObject *__pyx_n_s_NotImplemented;
02241 static PyObject *__pyx_kp_u_Not_applicable;
02242 static PyObject *__pyx_kp_u_Not_applicable_for_a_in_cliffor;
02243 static PyObject *__pyx_kp_u_Number_of_negative_indices_incl;
02244 static PyObject *__pyx_kp_u_Number_of_positive_indices_incl;
02245 static PyObject *__pyx_kp_u_Odd_part_of_multivector_sum_of;
02246 static PyObject *__pyx_kp_u_Odd_part_of_multivector_sum_of_2;
02247 static PyObject *__pyx_kp_u_Outer_product_power_of_multivec;
02248 static PyObject *__pyx_kp_u_Outer_product_power_x_clifford;
02249 static PyObject *__pyx_kp_u_Outer_product_print_clifford_1;
02250 static PyObject *__pyx_kp_u_Outer_product_x_clifford_1_x_cl;
02251 static PyObject *__pyx_kp_u_Power_self_to_the_m_x_clifford;
02252 static PyObject *__pyx_kp_u_Power_self_to_the_m_x_clifford_2;
02253 static PyObject *__pyx_kp_u_Pure_grade_vector_part_print_cl;
02254 static PyObject *__pyx_kp_u_Pure_part_print_clifford_1_1_1;
02255 static PyObject *__pyx_kp_u_Pure_part_print_pure_clifford_1;
02256 static PyObject *__pyx_kp_u_Put_self_into_a_larger_frame_co;
02257 static PyObject *__pyx_n_s_PyClical;
02258 static PyObject *__pyx_kp_s_PyClical_pyx;
02259 static PyObject *__pyx_kp_u_Quadratic_form_rev_x_x_0_print;
02260 static PyObject *__pyx_kp_u_Quadratic_form_rev_x_x_0_print_2;
02261 static PyObject *__pyx_kp_u_Quadratic_norm_error_tolerance;
02262 static PyObject *__pyx_kp_u_Random_multivector_within_a_fra;
02263 static PyObject *__pyx_n_s_Real;
02264 static PyObject *__pyx_kp_u_Real_part_synonym_for_scalar_pa;
02265 static PyObject *__pyx_kp_u_Relative_or_absolute_error_usin;
02266 static PyObject *__pyx_kp_u_Remove_all_terms_of_self_with_r;
02267 static PyObject *__pyx_kp_u_Reversion_eg_1_2_2_1_print_reve;
02268 static PyObject *__pyx_kp_u_Reversion_eg_clifford_1_cliffor;
02269 static PyObject *__pyx_n_s_RuntimeError;
02270 static PyObject *__pyx_kp_u_Scalar_part_clifford_1_1_1_2_sc;
02271 static PyObject *__pyx_kp_u_Scalar_part_scalar_clifford_1_1;
02272 static PyObject *__pyx_n_s_Sequence;
02273 static PyObject *__pyx_kp_u_Set_complement_not_print_index;
02274 static PyObject *__pyx_kp_u_Set_intersection_and_print_inde;
02275 static PyObject *__pyx_kp_u_Set_intersection_and_x_index_se;
02276 static PyObject *__pyx_kp_u_Set_the_value_of_an_index_set_o;
02277 static PyObject *__pyx_kp_u_Set_union_or_print_index_set_1;
02278 static PyObject *__pyx_kp_u_Set_union_or_x_index_set_1_x_in;
02279 static PyObject *__pyx_kp_u_Sign_of_geometric_product_of_tw;
02280 static PyObject *__pyx_kp_u_Sign_of_geometric_square_of_a_C;
02281 static PyObject *__pyx_kp_u_Sine_of_multivector_with_option;
02282 static PyObject *__pyx_kp_u_Square_root_of_1_which_commutates;
02283 static PyObject *__pyx_kp_u_Square_root_of_multivector_with;
02284 static PyObject *__pyx_kp_u_Subalgebra_generated_by_all_gen;
02285 static PyObject *__pyx_kp_u_Subscripting_map_from_index_set;

```

```

02286 static PyObject *__pyx_kp_u_Symmetric_set_difference_exclus;
02287 static PyObject *__pyx_kp_u_Symmetric_set_difference_exclus_2;
02288 static PyObject *__pyx_kp_u_Tangent_of_multivector_with_opt;
02289 static PyObject *__pyx_kp_u_Test_for_approximate_equality_o;
02290 static PyObject *__pyx_kp_u_Tests_for_functions_that_Doctes;
02291 static PyObject *__pyx_kp_u_Tests_for_functions_that_Doctes_2;
02292 static PyObject *__pyx_kp_u_The_informal_string_representat;
02293 static PyObject *__pyx_kp_u_The_informal_string_representat_2;
02294 static PyObject *__pyx_kp_u_The_official_string_representat;
02295 static PyObject *__pyx_kp_u_The_official_string_representat_2;
02296 static PyObject *__pyx_kp_u_This_comparison_operator_is_not;
02297 static PyObject *__pyx_kp_u_Transform_left_hand_side_using;
02298 static PyObject *__pyx_kp_u_Transform_left_hand_side_using_2;
02299 static PyObject *__pyx_n_s_TypeError;
02300 static PyObject *__pyx_kp_u_UTF_8;
02301 static PyObject *__pyx_kp_u_Unary_print_clifford_l_1;
02302 static PyObject *__pyx_kp_u_Unary_print_clifford_l_1_2;
02303 static PyObject *__pyx_n_s_ValueError;
02304 static PyObject *__pyx_kp_u_Vector_part_of_multivector_as_a;
02305 static PyObject *__pyx_kp_u__2;
02306 static PyObject *__pyx_kp_u__5;
02307 static PyObject *__pyx_kp_u__6;
02308 static PyObject *__pyx_kp_u__7;
02309 static PyObject *__pyx_kp_u__8;
02310 static PyObject *__pyx_kp_u__9;
02311 static PyObject *__pyx_n_s_abc;
02312 static PyObject *__pyx_kp_u_abs_line_1522;
02313 static PyObject *__pyx_n_s_acos;
02314 static PyObject *__pyx_kp_u_acos_line_1668;
02315 static PyObject *__pyx_n_s_acosh;
02316 static PyObject *__pyx_kp_u_acosh_line_1705;
02317 static PyObject *__pyx_kp_u_agc3_line_1893;
02318 static PyObject *__pyx_kp_u_approx_equal_line_1359;
02319 static PyObject *__pyx_n_s_args;
02320 static PyObject *__pyx_kp_u_as_frame;
02321 static PyObject *__pyx_n_s_asin;
02322 static PyObject *__pyx_kp_u_asin_line_1747;
02323 static PyObject *__pyx_n_s_asinh;
02324 static PyObject *__pyx_kp_u_asinh_line_1782;
02325 static PyObject *__pyx_n_s_atan;
02326 static PyObject *__pyx_kp_u_atan_line_1818;
02327 static PyObject *__pyx_n_s_atanh;
02328 static PyObject *__pyx_kp_u_atanh_line_1847;
02329 static PyObject *__pyx_kp_u_cga3_line_1873;
02330 static PyObject *__pyx_kp_u_cga3std_line_1882;
02331 static PyObject *__pyx_n_s_cl;
02332 static PyObject *__pyx_n_s_clifford;
02333 static PyObject *__pyx_kp_u_clifford__add__line_740;
02334 static PyObject *__pyx_kp_u_clifford__and__line_836;
02335 static PyObject *__pyx_kp_u_clifford__call__line_1020;
02336 static PyObject *__pyx_kp_u_clifford__getitem__line_707;
02337 static PyObject *__pyx_kp_u_clifford__iadd__line_751;
02338 static PyObject *__pyx_kp_u_clifford__iand__line_851;
02339 static PyObject *__pyx_kp_u_clifford__idiv__line_911;
02340 static PyObject *__pyx_kp_u_clifford__imod__line_821;
02341 static PyObject *__pyx_kp_u_clifford__imul__line_793;
02342 static PyObject *__pyx_kp_u_clifford__ior__line_950;
02343 static PyObject *__pyx_kp_u_clifford__isub__line_771;
02344 static PyObject *__pyx_kp_u_clifford__iter__line_638;
02345 static PyObject *__pyx_kp_u_clifford__ixor__line_881;
02346 static PyObject *__pyx_kp_u_clifford__mod__line_806;
02347 static PyObject *__pyx_kp_u_clifford__mul__line_780;
02348 static PyObject *__pyx_kp_u_clifford__neg__line_722;
02349 static PyObject *__pyx_kp_u_clifford__or__line_939;
02350 static PyObject *__pyx_kp_u_clifford__pos__line_731;
02351 static PyObject *__pyx_kp_u_clifford__pow__line_961;
02352 static PyObject *__pyx_kp_u_clifford__repr__line_1235;
02353 static PyObject *__pyx_kp_u_clifford__str__line_1244;
02354 static PyObject *__pyx_kp_u_clifford__sub__line_760;
02355 static PyObject *__pyx_kp_u_clifford__truediv__line_896;
02356 static PyObject *__pyx_kp_u_clifford__xor__line_866;
02357 static PyObject *__pyx_kp_u_clifford_abs_line_1175;
02358 static PyObject *__pyx_kp_u_clifford_conj_line_1138;
02359 static PyObject *__pyx_kp_u_clifford_copy_line_556;
02360 static PyObject *__pyx_kp_u_clifford_even_line_1061;
02361 static PyObject *__pyx_kp_u_clifford_frame_line_1224;
02362 static PyObject *__pyx_n_s_clifford_hidden_doctests;
02363 static PyObject *__pyx_kp_u_clifford_hidden_doctests_line_12;
02364 static PyObject *__pyx_kp_u_clifford_inv_line_926;
02365 static PyObject *__pyx_kp_u_clifford_involute_line_1107;
02366 static PyObject *__pyx_kp_u_clifford_isinf_line_1206;
02367 static PyObject *__pyx_kp_u_clifford_isnan_line_1215;
02368 static PyObject *__pyx_kp_u_clifford_max_abs_line_1184;
02369 static PyObject *__pyx_kp_u_clifford_norm_line_1164;
02370 static PyObject *__pyx_kp_u_clifford_odd_line_1070;
02371 static PyObject *__pyx_kp_u_clifford_outer_pow_line_1004;
02372 static PyObject *__pyx_kp_u_clifford_pow_line_980;

```

```

02373 static PyObject *__pyx_kp_u_clifford_pure_line_1050;
02374 static PyObject *__pyx_kp_u_clifford_quad_line_1153;
02375 static PyObject *__pyx_kp_u_clifford_reframe_line_649;
02376 static PyObject *__pyx_kp_u_clifford_reverse_line_1123;
02377 static PyObject *__pyx_kp_u_clifford_scalar_line_1039;
02378 static PyObject *__pyx_kp_u_clifford_truncated_line_1195;
02379 static PyObject *__pyx_kp_u_clifford_vector_part_line_1079;
02380 static PyObject *__pyx_n_s_cline_in_traceback;
02381 static PyObject *__pyx_n_s_close;
02382 static PyObject *__pyx_n_s_collections;
02383 static PyObject *__pyx_kp_u_compare_line_492;
02384 static PyObject *__pyx_kp_u_complexifier_line_1576;
02385 static PyObject *__pyx_n_s_conj;
02386 static PyObject *__pyx_kp_u_conj_line_1485;
02387 static PyObject *__pyx_n_s_copy;
02388 static PyObject *__pyx_n_s_cos;
02389 static PyObject *__pyx_kp_u_cos_line_1651;
02390 static PyObject *__pyx_n_s_cosh;
02391 static PyObject *__pyx_kp_u_cosh_line_1689;
02392 static PyObject *__pyx_n_s_doctest;
02393 static PyObject *__pyx_n_s_e;
02394 static PyObject *__pyx_kp_u_e_line_1936;
02395 static PyObject *__pyx_n_s_encode;
02396 static PyObject *__pyx_kp_u_error_squared_line_1346;
02397 static PyObject *__pyx_kp_u_error_squared_tol_line_1337;
02398 static PyObject *__pyx_n_s_even;
02399 static PyObject *__pyx_kp_u_even_line_1437;
02400 static PyObject *__pyx_n_s_exp;
02401 static PyObject *__pyx_kp_u_exp_line_1614;
02402 static PyObject *__pyx_n_s_fill;
02403 static PyObject *__pyx_n_s_frm;
02404 static PyObject *__pyx_kp_u_from;
02405 static PyObject *__pyx_n_s_getstate;
02406 static PyObject *__pyx_n_s_grade;
02407 static PyObject *__pyx_n_s_i;
02408 static PyObject *__pyx_kp_u_imag_line_1415;
02409 static PyObject *__pyx_n_s_import;
02410 static PyObject *__pyx_n_s_index_set;
02411 static PyObject *__pyx_kp_u_index_set__and__line_271;
02412 static PyObject *__pyx_kp_u_index_set__getitem__line_191;
02413 static PyObject *__pyx_kp_u_index_set__iand__line_282;
02414 static PyObject *__pyx_kp_u_index_set__invert__line_240;
02415 static PyObject *__pyx_kp_u_index_set__ior__line_304;
02416 static PyObject *__pyx_n_s_index_set__iter;
02417 static PyObject *__pyx_kp_u_index_set__iter__line_229;
02418 static PyObject *__pyx_kp_u_index_set__ixor__line_260;
02419 static PyObject *__pyx_kp_u_index_set__or__line_293;
02420 static PyObject *__pyx_kp_u_index_set__repr__line_384;
02421 static PyObject *__pyx_kp_u_index_set__setitem__line_179;
02422 static PyObject *__pyx_kp_u_index_set__str__line_395;
02423 static PyObject *__pyx_kp_u_index_set__xor__line_249;
02424 static PyObject *__pyx_kp_u_index_set_copy_line_65;
02425 static PyObject *__pyx_kp_u_index_set_count_line_315;
02426 static PyObject *__pyx_kp_u_index_set_count_neg_line_324;
02427 static PyObject *__pyx_kp_u_index_set_count_pos_line_333;
02428 static PyObject *__pyx_n_s_index_set_hidden_doctests;
02429 static PyObject *__pyx_kp_u_index_set_hidden_doctests_line_4;
02430 static PyObject *__pyx_kp_u_index_set_max_line_351;
02431 static PyObject *__pyx_kp_u_index_set_min_line_342;
02432 static PyObject *__pyx_kp_u_index_set_sign_of_mult_line_366;
02433 static PyObject *__pyx_kp_u_index_set_sign_of_square_line_37;
02434 static PyObject *__pyx_n_s_inv;
02435 static PyObject *__pyx_kp_u_inv_line_1378;
02436 static PyObject *__pyx_kp_u_invalid;
02437 static PyObject *__pyx_kp_u_invalid_string;
02438 static PyObject *__pyx_n_s_involute;
02439 static PyObject *__pyx_kp_u_involute_line_1455;
02440 static PyObject *__pyx_n_s_ist;
02441 static PyObject *__pyx_n_s_istpq;
02442 static PyObject *__pyx_kp_u_istpq_line_1949;
02443 static PyObject *__pyx_n_s_iter;
02444 static PyObject *__pyx_n_s_ixt;
02445 static PyObject *__pyx_kp_u_lexicographic_compare_eg_3_4_5;
02446 static PyObject *__pyx_n_s_lhs;
02447 static PyObject *__pyx_n_s_log;
02448 static PyObject *__pyx_kp_u_log_line_1628;
02449 static PyObject *__pyx_n_s_m;
02450 static PyObject *__pyx_n_s_main;
02451 static PyObject *__pyx_n_u_main;
02452 static PyObject *__pyx_n_s_math;
02453 static PyObject *__pyx_n_s_max;
02454 static PyObject *__pyx_kp_u_max_abs_line_1531;
02455 static PyObject *__pyx_kp_u_max_pos_line_513;
02456 static PyObject *__pyx_n_s_min;
02457 static PyObject *__pyx_kp_u_min_neg_line_504;
02458 static PyObject *__pyx_n_s_name;
02459 static PyObject *__pyx_n_s_nbar3;

```



```

02460 static PyObject *__pyx_n_s_ninf3;
02461 static PyObject *__pyx_kp_s_no_default_reduce_due_to_non;
02462 static PyObject *__pyx_n_s_norm;
02463 static PyObject *__pyx_kp_u_norm_line_1511;
02464 static PyObject *__pyx_kp_u_norm_sum_of_squares_of_coordina;
02465 static PyObject *__pyx_n_s_numbers;
02466 static PyObject *__pyx_n_s_obj;
02467 static PyObject *__pyx_n_s_odd;
02468 static PyObject *__pyx_kp_u_odd_line_1446;
02469 static PyObject *__pyx_n_s_other;
02470 static PyObject *__pyx_n_s_outer_pow;
02471 static PyObject *__pyx_kp_u_outer_pow_line_1567;
02472 static PyObject *__pyx_n_s_p;
02473 static PyObject *__pyx_n_s_pi;
02474 static PyObject *__pyx_n_s_pow;
02475 static PyObject *__pyx_kp_u_pow_line_1543;
02476 static PyObject *__pyx_n_s_pure;
02477 static PyObject *__pyx_kp_u_pure_line_1426;
02478 static PyObject *__pyx_n_s_pyx_vtable;
02479 static PyObject *__pyx_n_s_q;
02480 static PyObject *__pyx_n_s_quad;
02481 static PyObject *__pyx_kp_u_quad_line_1500;
02482 static PyObject *__pyx_kp_u_random_clifford_line_1864;
02483 static PyObject *__pyx_n_s_range;
02484 static PyObject *__pyx_kp_u_real_line_1404;
02485 static PyObject *__pyx_n_s_reduce;
02486 static PyObject *__pyx_n_s_reduce_cython;
02487 static PyObject *__pyx_n_s_reduce_ex;
02488 static PyObject *__pyx_n_s_reverse;
02489 static PyObject *__pyx_kp_u_reverse_line_1470;
02490 static PyObject *__pyx_n_s_rhs;
02491 static PyObject *__pyx_n_s_scalar;
02492 static PyObject *__pyx_n_s_scalar_epsilon;
02493 static PyObject *__pyx_kp_u_scalar_line_1393;
02494 static PyObject *__pyx_n_s_send;
02495 static PyObject *__pyx_n_s_setstate;
02496 static PyObject *__pyx_n_s_setstate_cython;
02497 static PyObject *__pyx_n_s_sin;
02498 static PyObject *__pyx_kp_u_sin_line_1728;
02499 static PyObject *__pyx_n_s_sinh;
02500 static PyObject *__pyx_kp_u_sinh_line_1768;
02501 static PyObject *__pyx_n_s_sqrt;
02502 static PyObject *__pyx_kp_u_sqrt_line_1591;
02503 static PyObject *__pyx_n_s_tan;
02504 static PyObject *__pyx_kp_u_tan_line_1801;
02505 static PyObject *__pyx_n_s_tanh;
02506 static PyObject *__pyx_kp_u_tanh_line_1835;
02507 static PyObject *__pyx_n_s_tau;
02508 static PyObject *__pyx_n_s_test;
02509 static PyObject *__pyx_n_s_test_2;
02510 static PyObject *__pyx_n_s_testmod;
02511 static PyObject *__pyx_n_s_threshold;
02512 static PyObject *__pyx_n_s_throw;
02513 static PyObject *__pyx_kp_u_to_frame;
02514 static PyObject *__pyx_n_s_tol;
02515 static PyObject *__pyx_kp_u_using;
02516 static PyObject *__pyx_kp_u_using_invalid;
02517 static PyObject *__pyx_kp_u_utf8;
02518 static PyObject *__pyx_kp_u_value;
02519 static PyObject *__pyx_n_s_version;
02520 static PyObject *__pyx_n_s_xrange;
02521 static PyObject *__pyx_pf_8PyClical_9index_set_copy(struct __pyx_obj_8PyClical_index_set
02522 *__pyx_v_self); /* proto */
02522 static int __pyx_pf_8PyClical_9index_set_2_cinit__(struct __pyx_obj_8PyClical_index_set
02523 *__pyx_v_self, PyObject *__pyx_v_other); /* proto */
02523 static void __pyx_pf_8PyClical_9index_set_4_dealloc(struct __pyx_obj_8PyClical_index_set
02524 *__pyx_v_self); /* proto */
02524 static PyObject *__pyx_pf_8PyClical_9index_set_6_richcmp__(struct __pyx_obj_8PyClical_index_set
02525 *__pyx_v_lhs, PyObject *__pyx_v_rhs, int __pyx_v_op); /* proto */
02525 static int __pyx_pf_8PyClical_9index_set_8_setitem__(struct __pyx_obj_8PyClical_index_set
02526 *__pyx_v_self, PyObject *__pyx_v_idx, PyObject *__pyx_v_val); /* proto */
02526 static PyObject *__pyx_pf_8PyClical_9index_set_10_getitem__(struct __pyx_obj_8PyClical_index_set
02527 *__pyx_v_self, PyObject *__pyx_v_idx); /* proto */
02527 static int __pyx_pf_8PyClical_9index_set_12_contains__(struct __pyx_obj_8PyClical_index_set
02528 *__pyx_v_self, PyObject *__pyx_v_idx); /* proto */
02528 static PyObject *__pyx_pf_8PyClical_9index_set_14_iter__(struct __pyx_obj_8PyClical_index_set
02529 *__pyx_v_self); /* proto */
02529 static PyObject *__pyx_pf_8PyClical_9index_set_17_invert__(struct __pyx_obj_8PyClical_index_set
02530 *__pyx_v_self); /* proto */
02530 static PyObject *__pyx_pf_8PyClical_9index_set_19_xor__(PyObject *__pyx_v_lhs, PyObject
02531 *__pyx_v_rhs); /* proto */
02531 static PyObject *__pyx_pf_8PyClical_9index_set_21_ixor__(struct __pyx_obj_8PyClical_index_set
02532 *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
02532 static PyObject *__pyx_pf_8PyClical_9index_set_23_and__(PyObject *__pyx_v_lhs, PyObject
02533 *__pyx_v_rhs); /* proto */
02533 static PyObject *__pyx_pf_8PyClical_9index_set_25_iand__(struct __pyx_obj_8PyClical_index_set
02534 *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */

```

```

02534 static PyObject *__pyx_pf_8PyClical_9index_set_27_or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs);
/* proto */
02535 static PyObject *__pyx_pf_8PyClical_9index_set_29_iior__(struct __pyx_obj_8PyClical_index_set
__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
02536 static PyObject *__pyx_pf_8PyClical_9index_set_31count(struct __pyx_obj_8PyClical_index_set
__pyx_v_self); /* proto */
02537 static PyObject *__pyx_pf_8PyClical_9index_set_33count_neg(struct __pyx_obj_8PyClical_index_set
__pyx_v_self); /* proto */
02538 static PyObject *__pyx_pf_8PyClical_9index_set_35count_pos(struct __pyx_obj_8PyClical_index_set
__pyx_v_self); /* proto */
02539 static PyObject *__pyx_pf_8PyClical_9index_set_37min(struct __pyx_obj_8PyClical_index_set
__pyx_v_self); /* proto */
02540 static PyObject *__pyx_pf_8PyClical_9index_set_39max(struct __pyx_obj_8PyClical_index_set
__pyx_v_self); /* proto */
02541 static PyObject *__pyx_pf_8PyClical_9index_set_41hash_fn(struct __pyx_obj_8PyClical_index_set
__pyx_v_self); /* proto */
02542 static PyObject *__pyx_pf_8PyClical_9index_set_43sign_of_mult(struct __pyx_obj_8PyClical_index_set
__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
02543 static PyObject *__pyx_pf_8PyClical_9index_set_45sign_of_square(struct __pyx_obj_8PyClical_index_set
__pyx_v_self); /* proto */
02544 static PyObject *__pyx_pf_8PyClical_9index_set_47_repr__(struct __pyx_obj_8PyClical_index_set
__pyx_v_self); /* proto */
02545 static PyObject *__pyx_pf_8PyClical_9index_set_49_str__(struct __pyx_obj_8PyClical_index_set
__pyx_v_self); /* proto */
02546 static PyObject *__pyx_pf_8PyClical_9index_set_51_reduce_cython__(CYTHON_UNUSED struct
__pyx_obj_8PyClical_index_set *__pyx_v_self); /* proto */
02547 static PyObject *__pyx_pf_8PyClical_9index_set_53_setstate_cython__(CYTHON_UNUSED struct
__pyx_obj_8PyClical_index_set *__pyx_v_self, CYTHON_UNUSED PyObject *__pyx_v__pyx_state); /* proto */
02548 static PyObject *__pyx_pf_8PyClical_index_set_hidden_doctests(CYTHON_UNUSED PyObject *__pyx_self); /*
proto */
02549 static PyObject *__pyx_pf_8PyClical_2compare(CYTHON_UNUSED PyObject *__pyx_self, PyObject
__pyx_v_lhs, PyObject *__pyx_v_rhs); /* proto */
02550 static PyObject *__pyx_pf_8PyClical_4min_neg(CYTHON_UNUSED PyObject *__pyx_self, PyObject
__pyx_v_obj); /* proto */
02551 static PyObject *__pyx_pf_8PyClical_6max_pos(CYTHON_UNUSED PyObject *__pyx_self, PyObject
__pyx_v_obj); /* proto */
02552 static PyObject *__pyx_pf_8PyClical_8clifford_copy(struct __pyx_obj_8PyClical_clifford *__pyx_v_self);
/* proto */
02553 static int __pyx_pf_8PyClical_8clifford_2_cinit__(struct __pyx_obj_8PyClical_clifford *__pyx_v_self,
PyObject *__pyx_v_other, PyObject *__pyx_v_ixt); /* proto */
02554 static void __pyx_pf_8PyClical_8clifford_4_dealloc__(struct __pyx_obj_8PyClical_clifford
__pyx_v_self); /* proto */
02555 static int __pyx_pf_8PyClical_8clifford_6_contains__(CYTHON_UNUSED struct
__pyx_obj_8PyClical_clifford *__pyx_v_self, CYTHON_UNUSED PyObject *__pyx_v_x); /* proto */
02556 static PyObject *__pyx_pf_8PyClical_8clifford_8_iter__(CYTHON_UNUSED struct
__pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
02557 static PyObject *__pyx_pf_8PyClical_8clifford_10reframe(struct __pyx_obj_8PyClical_clifford
__pyx_v_self, PyObject *__pyx_v_ixt); /* proto */
02558 static PyObject *__pyx_pf_8PyClical_8clifford_12_richcmp__(struct __pyx_obj_8PyClical_clifford
__pyx_v_lhs, PyObject *__pyx_v_rhs, int __pyx_v_op); /* proto */
02559 static PyObject *__pyx_pf_8PyClical_8clifford_14_getitem__(struct __pyx_obj_8PyClical_clifford
__pyx_v_self, PyObject *__pyx_v_ixt); /* proto */
02560 static PyObject *__pyx_pf_8PyClical_8clifford_16_neg__(struct __pyx_obj_8PyClical_clifford
__pyx_v_self); /* proto */
02561 static PyObject *__pyx_pf_8PyClical_8clifford_18_pos__(struct __pyx_obj_8PyClical_clifford
__pyx_v_self); /* proto */
02562 static PyObject *__pyx_pf_8PyClical_8clifford_20_add__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs);
/* proto */
02563 static PyObject *__pyx_pf_8PyClical_8clifford_22_iadd__(struct __pyx_obj_8PyClical_clifford
__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
02564 static PyObject *__pyx_pf_8PyClical_8clifford_24_sub__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs);
/* proto */
02565 static PyObject *__pyx_pf_8PyClical_8clifford_26_isub__(struct __pyx_obj_8PyClical_clifford
__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
02566 static PyObject *__pyx_pf_8PyClical_8clifford_28_mul__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs);
/* proto */
02567 static PyObject *__pyx_pf_8PyClical_8clifford_30_imul__(struct __pyx_obj_8PyClical_clifford
__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
02568 static PyObject *__pyx_pf_8PyClical_8clifford_32_mod__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs);
/* proto */
02569 static PyObject *__pyx_pf_8PyClical_8clifford_34_imod__(struct __pyx_obj_8PyClical_clifford
__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
02570 static PyObject *__pyx_pf_8PyClical_8clifford_36_and__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs);
/* proto */
02571 static PyObject *__pyx_pf_8PyClical_8clifford_38_iand__(struct __pyx_obj_8PyClical_clifford
__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
02572 static PyObject *__pyx_pf_8PyClical_8clifford_40_xor__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs);
/* proto */
02573 static PyObject *__pyx_pf_8PyClical_8clifford_42_ixor__(struct __pyx_obj_8PyClical_clifford
__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
02574 static PyObject *__pyx_pf_8PyClical_8clifford_44_truediv__(PyObject *__pyx_v_lhs, PyObject
__pyx_v_rhs); /* proto */
02575 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
02576 static PyObject *__pyx_pf_8PyClical_8clifford_46_idiv__(struct __pyx_obj_8PyClical_clifford
__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
02577 #endif
02578 static PyObject *__pyx_pf_8PyClical_8clifford_48inv(struct __pyx_obj_8PyClical_clifford

```

```

 __pyx_v_self); /* proto */
02579 static PyObject *__pyx_pf_8PyClical_8clifford_50_or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs);
 /* proto */
02580 static PyObject *__pyx_pf_8PyClical_8clifford_52_iior__(struct __pyx_obj_8PyClical_clifford
 *__pyx_v_self, PyObject *__pyx_v_rhs); /* proto */
02581 static PyObject *__pyx_pf_8PyClical_8clifford_54_pow__(PyObject *__pyx_v_self, PyObject *__pyx_v_m,
 CYTHON_UNUSED PyObject *__pyx_v_dummy); /* proto */
02582 static PyObject *__pyx_pf_8PyClical_8clifford_56pow(struct __pyx_obj_8PyClical_clifford *__pyx_v_self,
 PyObject *__pyx_v_m); /* proto */
02583 static PyObject *__pyx_pf_8PyClical_8clifford_58outer_pow(struct __pyx_obj_8PyClical_clifford
 *__pyx_v_self, PyObject *__pyx_v_m); /* proto */
02584 static PyObject *__pyx_pf_8PyClical_8clifford_60_call__(struct __pyx_obj_8PyClical_clifford
 *__pyx_v_self, PyObject *__pyx_v_grade); /* proto */
02585 static PyObject *__pyx_pf_8PyClical_8clifford_62scalar(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02586 static PyObject *__pyx_pf_8PyClical_8clifford_64pure(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02587 static PyObject *__pyx_pf_8PyClical_8clifford_66even(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02588 static PyObject *__pyx_pf_8PyClical_8clifford_68odd(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02589 static PyObject *__pyx_pf_8PyClical_8clifford_70vector_part(struct __pyx_obj_8PyClical_clifford
 *__pyx_v_self, PyObject *__pyx_v_frm); /* proto */
02590 static PyObject *__pyx_pf_8PyClical_8clifford_72involute(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02591 static PyObject *__pyx_pf_8PyClical_8clifford_74reverse(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02592 static PyObject *__pyx_pf_8PyClical_8clifford_76conj(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02593 static PyObject *__pyx_pf_8PyClical_8clifford_78quad(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02594 static PyObject *__pyx_pf_8PyClical_8clifford_80norm(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02595 static PyObject *__pyx_pf_8PyClical_8clifford_82abs(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02596 static PyObject *__pyx_pf_8PyClical_8clifford_84max_abs(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02597 static PyObject *__pyx_pf_8PyClical_8clifford_86truncated(struct __pyx_obj_8PyClical_clifford
 *__pyx_v_self, PyObject *__pyx_v_limit); /* proto */
02598 static PyObject *__pyx_pf_8PyClical_8clifford_88isinf(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02599 static PyObject *__pyx_pf_8PyClical_8clifford_90isnan(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02600 static PyObject *__pyx_pf_8PyClical_8clifford_92frame(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02601 static PyObject *__pyx_pf_8PyClical_8clifford_94_repr__(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02602 static PyObject *__pyx_pf_8PyClical_8clifford_96__str__(struct __pyx_obj_8PyClical_clifford
 __pyx_v_self); / proto */
02603 static PyObject *__pyx_pf_8PyClical_8clifford_98__reduce_cython__(CYTHON_UNUSED struct
 __pyx_obj_8PyClical_clifford *__pyx_v_self); /* proto */
02604 static PyObject *__pyx_pf_8PyClical_8clifford_100__setstate_cython__(CYTHON_UNUSED struct
 __pyx_obj_8PyClical_clifford *__pyx_v_self, CYTHON_UNUSED PyObject *__pyx_v__pyx_state); /* proto */
02605 static PyObject *__pyx_pf_8PyClical_8clifford_hidden_doctests(CYTHON_UNUSED PyObject *__pyx_self); /*
 proto */
02606 static PyObject *__pyx_pf_8PyClical_10error_squared_tol(CYTHON_UNUSED PyObject *__pyx_self, PyObject
 __pyx_v_obj); / proto */
02607 static PyObject *__pyx_pf_8PyClical_12error_squared(CYTHON_UNUSED PyObject *__pyx_self, PyObject
 *__pyx_v_lhs, PyObject *__pyx_v_rhs, PyObject *__pyx_v_threshold); /* proto */
02608 static PyObject *__pyx_pf_8PyClical_14approx_equal(CYTHON_UNUSED PyObject *__pyx_self, PyObject
 *__pyx_v_lhs, PyObject *__pyx_v_rhs, PyObject *__pyx_v_threshold, PyObject *__pyx_v_tol); /* proto */
02609 static PyObject *__pyx_pf_8PyClical_16inv(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02610 static PyObject *__pyx_pf_8PyClical_18scalar(CYTHON_UNUSED PyObject *__pyx_self, PyObject
 __pyx_v_obj); / proto */
02611 static PyObject *__pyx_pf_8PyClical_20real(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02612 static PyObject *__pyx_pf_8PyClical_22imag(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02613 static PyObject *__pyx_pf_8PyClical_24pure(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02614 static PyObject *__pyx_pf_8PyClical_26even(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02615 static PyObject *__pyx_pf_8PyClical_28odd(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02616 static PyObject *__pyx_pf_8PyClical_30involute(CYTHON_UNUSED PyObject *__pyx_self, PyObject
 __pyx_v_obj); / proto */
02617 static PyObject *__pyx_pf_8PyClical_32reverse(CYTHON_UNUSED PyObject *__pyx_self, PyObject
 __pyx_v_obj); / proto */
02618 static PyObject *__pyx_pf_8PyClical_34conj(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02619 static PyObject *__pyx_pf_8PyClical_36quad(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02620 static PyObject *__pyx_pf_8PyClical_38norm(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02621 static PyObject *__pyx_pf_8PyClical_40abs(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */

```



```

02622 static PyObject *__pyx_pf_8PyClical_42max_abs(CYTHON_UNUSED PyObject *__pyx_self, PyObject
 __pyx_v_obj); / proto */
02623 static PyObject *__pyx_pf_8PyClical_44pow(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj,
 PyObject *__pyx_v_m); /* proto */
02624 static PyObject *__pyx_pf_8PyClical_46outer_pow(CYTHON_UNUSED PyObject *__pyx_self, PyObject
 *__pyx_v_obj, PyObject *__pyx_v_m); /* proto */
02625 static PyObject *__pyx_pf_8PyClical_48complexifier(CYTHON_UNUSED PyObject *__pyx_self, PyObject
 __pyx_v_obj); / proto */
02626 static PyObject *__pyx_pf_8PyClical_50sqrt(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj,
 PyObject *__pyx_v_i); /* proto */
02627 static PyObject *__pyx_pf_8PyClical_52exp(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02628 static PyObject *__pyx_pf_8PyClical_54log(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj,
 PyObject *__pyx_v_i); /* proto */
02629 static PyObject *__pyx_pf_8PyClical_56cos(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj,
 PyObject *__pyx_v_i); /* proto */
02630 static PyObject *__pyx_pf_8PyClical_58acos(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj,
 PyObject *__pyx_v_i); /* proto */
02631 static PyObject *__pyx_pf_8PyClical_60cosh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02632 static PyObject *__pyx_pf_8PyClical_62acosh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj,
 PyObject *__pyx_v_i); /* proto */
02633 static PyObject *__pyx_pf_8PyClical_64sin(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj,
 PyObject *__pyx_v_i); /* proto */
02634 static PyObject *__pyx_pf_8PyClical_66asin(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj,
 PyObject *__pyx_v_i); /* proto */
02635 static PyObject *__pyx_pf_8PyClical_68sinh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02636 static PyObject *__pyx_pf_8PyClical_70asinh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj,
 PyObject *__pyx_v_i); /* proto */
02637 static PyObject *__pyx_pf_8PyClical_72tan(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj,
 PyObject *__pyx_v_i); /* proto */
02638 static PyObject *__pyx_pf_8PyClical_74atan(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj,
 PyObject *__pyx_v_i); /* proto */
02639 static PyObject *__pyx_pf_8PyClical_76tanh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02640 static PyObject *__pyx_pf_8PyClical_78atanh(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj,
 PyObject *__pyx_v_i); /* proto */
02641 static PyObject *__pyx_pf_8PyClical_80random_clifford(CYTHON_UNUSED PyObject *__pyx_self, struct
 __pyx_obj_8PyClical_index_set *__pyx_v_ixt, PyObject *__pyx_v_fill); /* proto */
02642 static PyObject *__pyx_pf_8PyClical_82cga3(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02643 static PyObject *__pyx_pf_8PyClical_84cga3std(CYTHON_UNUSED PyObject *__pyx_self, PyObject
 __pyx_v_obj); / proto */
02644 static PyObject *__pyx_pf_8PyClical_86agc3(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj);
 /* proto */
02645 static PyObject *__pyx_pf_8PyClical_88e(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_obj); /*
 proto */
02646 static PyObject *__pyx_pf_8PyClical_90istpq(CYTHON_UNUSED PyObject *__pyx_self, PyObject *__pyx_v_p,
 PyObject *__pyx_v_q); /* proto */
02647 static PyObject *__pyx_pf_8PyClical_92_test(CYTHON_UNUSED PyObject *__pyx_self); /* proto */
02648 static PyObject *__pyx_tp_new_8PyClical_index_set(PyTypeObject *t, PyObject *a, PyObject *k);
 /*proto*/
02649 static PyObject *__pyx_tp_new_8PyClical_clifford(PyTypeObject *t, PyObject *a, PyObject *k); /*proto*/
02650 static PyObject *__pyx_tp_new_8PyClical__pyx_scope_struct____iter__(PyTypeObject *t, PyObject *a,
 PyObject *k); /*proto*/
02651 static PyObject *__pyx_float_0_0;
02652 static PyObject *__pyx_float_1_0;
02653 static PyObject *__pyx_float_4_0;
02654 static PyObject *__pyx_float_8_0;
02655 static PyObject *__pyx_int_0;
02656 static PyObject *__pyx_int_1;
02657 static PyObject *__pyx_int_4;
02658 static PyObject *__pyx_int_neg_1;
02659 static PyObject *__pyx_tuple_3;
02660 static PyObject *__pyx_tuple_4;
02661 static PyObject *__pyx_tuple_10;
02662 static PyObject *__pyx_tuple_11;
02663 static PyObject *__pyx_tuple_12;
02664 static PyObject *__pyx_tuple_15;
02665 static PyObject *__pyx_tuple_16;
02666 static PyObject *__pyx_tuple_18;
02667 static PyObject *__pyx_tuple_20;
02668 static PyObject *__pyx_tuple_21;
02669 static PyObject *__pyx_tuple_22;
02670 static PyObject *__pyx_codeobj__13;
02671 static PyObject *__pyx_codeobj__14;
02672 static PyObject *__pyx_codeobj__17;
02673 static PyObject *__pyx_codeobj__19;
02674 static PyObject *__pyx_codeobj__23;
02675 /* Late includes */
02676
02677 /* "PyClical.pyx":40
02678 * cdef class index_set
02679 *
02680 * cdef inline IndexSet toIndexSet(obj): # ««««««
02681 * """

```

```

02682 * Return the C++ IndexSet instance wrapped by index_set(obj).
02683 */
02684
02685 static CYTHON_INLINE IndexSet __pyx_f_8PyClical_toIndexSet(PyObject *__pyx_v_obj) {
02686 IndexSet __pyx_r;
02687 __Pyx_RefNannyDeclarations
02688 PyObject *__pyx_t_1 = NULL;
02689 int __pyx_lineno = 0;
02690 const char *__pyx_filename = NULL;
02691 int __pyx_clineno = 0;
02692 __Pyx_RefNannySetupContext("toIndexSet", 0);
02693
02694 /* "PyClical.pyx":44
02695 * Return the C++ IndexSet instance wrapped by index_set(obj).
02696 * """
02697 * return index_set(obj).instance[0] # ««««««««
02698 *
02699 * cdef class index_set:
02700 */
02701 __pyx_t_1 = __Pyx_PyObject_CallOneArg((PyObject *)__pyx_ptype_8PyClical_index_set, __pyx_v_obj);
02702 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 44, __pyx_L1_error)
02703 __Pyx_GOTREF(__pyx_t_1);
02704 __pyx_r = ((struct __pyx_obj_8PyClical_index_set *)__pyx_t_1->instance[0]);
02705 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
02706 goto __pyx_L0;
02707
02708 /* "PyClical.pyx":40
02709 * cdef class index_set
02710 *
02711 * cdef inline IndexSet toIndexSet(obj): # ««««««««
02712 * """
02713 * Return the C++ IndexSet instance wrapped by index_set(obj).
02714 */
02715 /* function exit code */
02716 __pyx_L1_error:;
02717 __Pyx_XDECREF(__pyx_t_1);
02718 __Pyx_WriteUnraisable("PyClical.toIndexSet", __pyx_clineno, __pyx_lineno, __pyx_filename, 1, 0);
02719 __Pyx_prepend_to_initialize(&__pyx_r);
02720 __pyx_L0:;
02721 __Pyx_RefNannyFinishContext();
02722 return __pyx_r;
02723 }
02724
02725 /* "PyClical.pyx":52
02726 * cdef IndexSet *instance # Wrapped instance of C++ class IndexSet.
02727 *
02728 * cdef inline wrap(index_set self, IndexSet other): # ««««««««
02729 * """
02730 * Wrap an instance of the C++ class IndexSet.
02731 */
02732
02733 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_9index_set_wrap(struct __pyx_obj_8PyClical_index_set
__pyx_v_self, IndexSet __pyx_v_other) {
02734 PyObject *__pyx_r = NULL;
02735 __Pyx_RefNannyDeclarations
02736 __Pyx_RefNannySetupContext("wrap", 0);
02737
02738 /* "PyClical.pyx":56
02739 * Wrap an instance of the C++ class IndexSet.
02740 * """
02741 * self.instance[0] = other # ««««««««
02742 * return self
02743 */
02744 (__pyx_v_self->instance[0]) = __pyx_v_other;
02745
02746 /* "PyClical.pyx":57
02747 * """
02748 * self.instance[0] = other
02749 * return self # ««««««««
02750 *
02751 * cdef inline IndexSet unwrap(index_set self):
02752 */
02753 __Pyx_XDECREF(__pyx_r);
02754 __Pyx_INCREF((PyObject *)__pyx_v_self);
02755 __pyx_r = (PyObject *)__pyx_v_self;
02756 goto __pyx_L0;
02757
02758 /* "PyClical.pyx":52
02759 * cdef IndexSet *instance # Wrapped instance of C++ class IndexSet.
02760 *
02761 * cdef inline wrap(index_set self, IndexSet other): # ««««««««
02762 * """
02763 * Wrap an instance of the C++ class IndexSet.
02764 */
02765
02766

```

```

02767 /* function exit code */
02768 __pyx_L0:;
02769 __Pyx_XGIVEREF(__pyx_r);
02770 __Pyx_RefNannyFinishContext();
02771 return __pyx_r;
02772 }
02773
02774 /* "PyClical.pyx":59
02775 * return self
02776 *
02777 * cdef inline IndexSet unwrap(index_set self): # ««««««««
02778 * """
02779 * Return the wrapped C++ IndexSet instance.
02780 */
02781
02782 static CYTHON_INLINE IndexSet __pyx_f_8PyClical_9index_set_unwrap(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self) {
02783 IndexSet __pyx_r;
02784 __Pyx_RefNannyDeclarations
02785 __Pyx_RefNannySetupContext("unwrap", 0);
02786
02787 /* "PyClical.pyx":63
02788 * Return the wrapped C++ IndexSet instance.
02789 * """
02790 * return self.instance[0] # ««««««««
02791 *
02792 * cpdef copy(index_set self):
02793 */
02794 __pyx_r = (__pyx_v_self->instance[0]);
02795 goto __pyx_L0;
02796
02797 /* "PyClical.pyx":59
02798 * return self
02799 *
02800 * cdef inline IndexSet unwrap(index_set self): # ««««««««
02801 * """
02802 * Return the wrapped C++ IndexSet instance.
02803 */
02804
02805 /* function exit code */
02806 __pyx_L0:;
02807 __Pyx_RefNannyFinishContext();
02808 return __pyx_r;
02809 }
02810
02811 /* "PyClical.pyx":65
02812 * return self.instance[0]
02813 *
02814 * cpdef copy(index_set self): # ««««««««
02815 * """
02816 * Copy this index_set object.
02817 */
02818
02819 static PyObject *__pyx_pw_8PyClical_9index_set_1copy(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject
*unused); /*proto*/
02820 static PyObject *__pyx_f_8PyClical_9index_set_copy(struct __pyx_obj_8PyClical_index_set *__pyx_v_self,
int __pyx_skip_dispatch) {
02821 PyObject *__pyx_r = NULL;
02822 __Pyx_RefNannyDeclarations
02823 PyObject *__pyx_t_1 = NULL;
02824 PyObject *__pyx_t_2 = NULL;
02825 PyObject *__pyx_t_3 = NULL;
02826 PyObject *__pyx_t_4 = NULL;
02827 int __pyx_lineno = 0;
02828 const char *__pyx_filename = NULL;
02829 int __pyx_clineno = 0;
02830 __Pyx_RefNannySetupContext("copy", 0);
02831 /* Check if called by wrapper */
02832 if (unlikely(__pyx_skip_dispatch)) ;
02833 /* Check if overridden in Python */
02834 else if (unlikely((Py_TYPE(((PyObject *)__pyx_v_self))->tp_dictoffset != 0) || (Py_TYPE(((PyObject
*)__pyx_v_self))->tp_flags & (Py_TPFLAGS_IS_ABSTRACT | Py_TPFLAGS_HEAPTYPE)))) {
02835 #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_PYTYPE_LOOKUP && CYTHON_USE_TYPE_SLOTS
02836 static PY_UINT64_T __pyx_tp_dict_version = __PYX_DICT_VERSION_INIT, __pyx_obj_dict_version =
__PYX_DICT_VERSION_INIT;
02837 if (unlikely(!__Pyx_object_dict_version_matches(((PyObject *)__pyx_v_self), __pyx_tp_dict_version,
__pyx_obj_dict_version))) {
02838 PY_UINT64_T __pyx_type_dict_guard = __Pyx_get_tp_dict_version(((PyObject *)__pyx_v_self));
02839 #endif
02840 __pyx_t_1 = __Pyx_PyObject_GetAttrStr(((PyObject *)__pyx_v_self), __pyx_n_s_copy); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 65, __pyx_L1_error)
02841 __Pyx_GOTREF(__pyx_t_1);
02842 if (!PyCFunction_Check(__pyx_t_1) || (PyCFunction_GET_FUNCTION(__pyx_t_1) !=
(PyCFunction)(void*)__pyx_pw_8PyClical_9index_set_1copy)) {
02843 __Pyx_XDECREF(__pyx_r);
02844 __Pyx_INCREF(__pyx_t_1);
02845 __pyx_t_3 = __pyx_t_1; __pyx_t_4 = NULL;

```

```

02846 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_3))) {
02847 __pyx_t_4 = PyMethod_GET_SELF(__pyx_t_3);
02848 if (likely(__pyx_t_4)) {
02849 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
02850 __Pyx_INCREF(__pyx_t_4);
02851 __Pyx_INCREF(function);
02852 __Pyx_DECREF_SET(__pyx_t_3, function);
02853 }
02854 }
02855 __pyx_t_2 = (__pyx_t_4) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_4) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
02856 __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
02857 if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 65, __pyx_L1_error)
02858 __Pyx_GOTREF(__pyx_t_2);
02859 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
02860 __pyx_r = __pyx_t_2;
02861 __pyx_t_2 = 0;
02862 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
02863 goto __pyx_L0;
02864 }
02865 #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_PYTYPE_LOOKUP && CYTHON_USE_TYPE_SLOTS
02866 __pyx_tp_dict_version = __Pyx_get_tp_dict_version(((PyObject *)__pyx_v_self));
02867 __pyx_obj_dict_version = __Pyx_get_object_dict_version(((PyObject *)__pyx_v_self));
02868 if (unlikely(__pyx_type_dict_guard != __pyx_tp_dict_version)) {
02869 __pyx_tp_dict_version = __pyx_obj_dict_version = __PYX_DICT_VERSION_INIT;
02870 }
02871 #endif
02872 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
02873 #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_PYTYPE_LOOKUP && CYTHON_USE_TYPE_SLOTS
02874 }
02875 #endif
02876 }
02877
02878 /* "PyClical.pyx":72
02879 * {1}
02880 * """
02881 * return index_set(self) # ««««««««
02882 *
02883 * def __cinit__(self, other = 0):
02884 */
02885 __Pyx_XDECREF(__pyx_r);
02886 __pyx_t_1 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_index_set), ((PyObject
*) __pyx_v_self)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 72, __pyx_L1_error)
02887 __Pyx_GOTREF(__pyx_t_1);
02888 __pyx_r = __pyx_t_1;
02889 __pyx_t_1 = 0;
02890 goto __pyx_L0;
02891
02892 /* "PyClical.pyx":65
02893 * return self.instance[0]
02894 *
02895 * cpdef copy(index_set self): # ««««««««
02896 * """
02897 * Copy this index_set object.
02898 */
02899
02900 /* function exit code */
02901 __pyx_L1_error:;
02902 __Pyx_XDECREF(__pyx_t_1);
02903 __Pyx_XDECREF(__pyx_t_2);
02904 __Pyx_XDECREF(__pyx_t_3);
02905 __Pyx_XDECREF(__pyx_t_4);
02906 __Pyx_AddTraceback("PyClical.index_set.copy", __pyx_clineno, __pyx_lineno, __pyx_filename);
02907 __pyx_r = 0;
02908 __pyx_L0:;
02909 __Pyx_XGIVEREF(__pyx_r);
02910 __Pyx_RefNannyFinishContext();
02911 return __pyx_r;
02912 }
02913
02914 /* Python wrapper */
02915 static PyObject *__pyx_pw_8PyClical_9index_set_1copy(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject
*unused); /*proto*/
02916 static char __pyx_doc_8PyClical_9index_set_copy[] = "\n Copy this index_set object.\n\n
>> s=index_set(1); t=s.copy(); print(t)\n {1}\n ";
02917 static PyObject *__pyx_pw_8PyClical_9index_set_1copy(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject
*unused) {
02918 PyObject *__pyx_r = 0;
02919 __Pyx_RefNannyDeclarations
02920 __Pyx_RefNannySetupContext("copy (wrapper)", 0);
02921 __pyx_r = __pyx_pf_8PyClical_9index_set_copy(((struct __pyx_obj_8PyClical_index_set
*) __pyx_v_self));
02922
02923 /* function exit code */
02924 __Pyx_RefNannyFinishContext();
02925 return __pyx_r;
02926 }

```

```

02927
02928 static PyObject *__pyx_pf_8PyClical_9index_set_copy(struct __pyx_obj_8PyClical_index_set
 *__pyx_v_self) {
02929 PyObject *__pyx_r = NULL;
02930 __Pyx_RefNannyDeclarations
02931 PyObject *__pyx_t_1 = NULL;
02932 int __pyx_lineno = 0;
02933 const char *__pyx_filename = NULL;
02934 int __pyx_clineno = 0;
02935 __Pyx_RefNannySetupContext("copy", 0);
02936 __Pyx_XDECREF(__pyx_r);
02937 __pyx_t_1 = __pyx_f_8PyClical_9index_set_copy(__pyx_v_self, 1); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 65, __pyx_L1_error)
02938 __Pyx_GOTREF(__pyx_t_1);
02939 __pyx_r = __pyx_t_1;
02940 __pyx_t_1 = 0;
02941 goto __pyx_L0;
02942
02943 /* function exit code */
02944 __pyx_L1_error:;
02945 __Pyx_XDECREF(__pyx_t_1);
02946 __Pyx_AddTraceback("PyClical.index_set.copy", __pyx_clineno, __pyx_lineno, __pyx_filename);
02947 __pyx_r = NULL;
02948 __pyx_L0:;
02949 __Pyx_XGIVEREF(__pyx_r);
02950 __Pyx_RefNannyFinishContext();
02951 return __pyx_r;
02952 }
02953
02954 /* "PyClical.pyx":74
02955 * return index_set(self)
02956 *
02957 * def __cinit__(self, other = 0): # ««««««««
02958 * """
02959 * Construct an object of type index_set.
02960 */
02961
02962 /* Python wrapper */
02963 static int __pyx_pw_8PyClical_9index_set_3__cinit__(PyObject *__pyx_v_self, PyObject *__pyx_args,
 PyObject *__pyx_kwds); /*proto*/
02964 static int __pyx_pw_8PyClical_9index_set_3__cinit__(PyObject *__pyx_v_self, PyObject *__pyx_args,
 PyObject *__pyx_kwds) {
02965 PyObject *__pyx_v_other = 0;
02966 int __pyx_lineno = 0;
02967 const char *__pyx_filename = NULL;
02968 int __pyx_clineno = 0;
02969 int __pyx_r;
02970 __Pyx_RefNannyDeclarations
02971 __Pyx_RefNannySetupContext("__cinit__ (wrapper)", 0);
02972 {
02973 static PyObject *__pyx_pyargnames[] = {&__pyx_n_s_other,0};
02974 PyObject* values[1] = {0};
02975 values[0] = ((PyObject *)__pyx_int_0);
02976 if (unlikely(__pyx_kwds)) {
02977 Py_ssize_t kw_args;
02978 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
02979 switch (pos_args) {
02980 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
02981 CYTHON_FALLTHROUGH;
02982 case 0: break;
02983 default: goto __pyx_L5_argtuple_error;
02984 }
02985 kw_args = PyDict_Size(__pyx_kwds);
02986 switch (pos_args) {
02987 case 0:
02988 if (kw_args > 0) {
02989 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_other);
02990 if (value) { values[0] = value; kw_args--; }
02991 }
02992 if (unlikely(kw_args > 0)) {
02993 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values, pos_args,
02994 "__cinit__") < 0)) __PYX_ERR(0, 74, __pyx_L3_error)
02995 }
02996 } else {
02997 switch (PyTuple_GET_SIZE(__pyx_args)) {
02998 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
02999 CYTHON_FALLTHROUGH;
03000 case 0: break;
03001 default: goto __pyx_L5_argtuple_error;
03002 }
03003 }
03004 __pyx_v_other = values[0];
03005 }
03006 goto __pyx_L4_argument_unpacking_done;
03007 __pyx_L5_argtuple_error:;
03008 __Pyx_RaiseArgtupleInvalid("__cinit__", 0, 0, 1, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 74,

```

```

__pyx_L3_error)
03009 __pyx_L3_error;;
03010 __Pyx_AddTraceback("PyCliclal.index_set.__cinit__", __pyx_clineno, __pyx_lineno, __pyx_filename);
03011 __Pyx_RefNannyFinishContext();
03012 return -1;
03013 __pyx_L4_argument_unpacking_done;;
03014 __pyx_r = __pyx_pf_8PyCliclal_9index_set_2__cinit__((struct __pyx_obj_8PyCliclal_index_set
*)__pyx_v_self), __pyx_v_other);
03015
03016 /* function exit code */
03017 __Pyx_RefNannyFinishContext();
03018 return __pyx_r;
03019 }
03020
03021 static int __pyx_pf_8PyCliclal_9index_set_2__cinit__(struct __pyx_obj_8PyCliclal_index_set
*__pyx_v_self, PyObject *__pyx_v_other) {
03022 PyObject *__pyx_v_error_msg_prefix = NULL;
03023 PyObject *__pyx_v_idx = NULL;
03024 PyObject *__pyx_v_bother = NULL;
03025 int __pyx_r;
03026 __Pyx_RefNannyDeclarations
03027 int __pyx_t_1;
03028 int __pyx_t_2;
03029 IndexSet *__pyx_t_3;
03030 PyObject *__pyx_t_4 = NULL;
03031 PyObject *__pyx_t_5 = NULL;
03032 int __pyx_t_6;
03033 int __pyx_t_7;
03034 PyObject *__pyx_t_8 = NULL;
03035 PyObject *__pyx_t_9 = NULL;
03036 PyObject *__pyx_t_10 = NULL;
03037 Py_ssize_t __pyx_t_11;
03038 PyObject *(*__pyx_t_12)(PyObject *);
03039 PyObject *__pyx_t_13 = NULL;
03040 PyObject *__pyx_t_14 = NULL;
03041 PyObject *__pyx_t_15 = NULL;
03042 PyObject *__pyx_t_16 = NULL;
03043 char *__pyx_t_17;
03044 int __pyx_lineno = 0;
03045 const char *__pyx_filename = NULL;
03046 int __pyx_clineno = 0;
03047 __Pyx_RefNannySetupContext("__cinit__", 0);
03048
03049 /* "PyCliclal.pyx":93
03050 * {}
03051 * """
03052 * error_msg_prefix = "Cannot initialize index_set object from"
03053 * if isinstance(other, index_set):
03054 * self.instance = new IndexSet((<index_set>other).unwrap())
03055 */
03056 __Pyx_INCREF(__pyx_kp_u_Cannot_initialize_index_set_obje);
03057 __pyx_v_error_msg_prefix = __pyx_kp_u_Cannot_initialize_index_set_obje;
03058
03059 /* "PyCliclal.pyx":94
03060 * """
03061 * error_msg_prefix = "Cannot initialize index_set object from"
03062 * if isinstance(other, index_set):
03063 * self.instance = new IndexSet((<index_set>other).unwrap())
03064 * elif isinstance(other, numbers.Integral):
03065 */
03066 __pyx_t_1 = __Pyx_TypeCheck(__pyx_v_other, __pyx_ptype_8PyCliclal_index_set);
03067 __pyx_t_2 = (__pyx_t_1 != 0);
03068 if (__pyx_t_2) {
03069
03070 /* "PyCliclal.pyx":95
03071 * error_msg_prefix = "Cannot initialize index_set object from"
03072 * if isinstance(other, index_set):
03073 * self.instance = new IndexSet((<index_set>other).unwrap())
03074 * elif isinstance(other, numbers.Integral):
03075 * self.instance = new IndexSet(<int>other)
03076 */
03077 try {
03078 __pyx_t_3 = new IndexSet(__pyx_f_8PyCliclal_9index_set_unwrap(((struct
__pyx_obj_8PyCliclal_index_set *)__pyx_v_other)));
03079 } catch (...) {
03080 __Pyx_CppExn2PyErr();
03081 __PYX_ERR(0, 95, __pyx_L1_error)
03082 }
03083 __pyx_v_self->instance = __pyx_t_3;
03084
03085 /* "PyCliclal.pyx":94
03086 * """
03087 * error_msg_prefix = "Cannot initialize index_set object from"
03088 * if isinstance(other, index_set):
03089 * self.instance = new IndexSet((<index_set>other).unwrap())
03090 * elif isinstance(other, numbers.Integral):
03091 */

```

```

03092 goto __pyx_L3;
03093 }
03094
03095 /* "PyClical.pyx":96
03096 * if isinstance(other, index_set):
03097 * self.instance = new IndexSet((<index_set>other).unwrap())
03098 * elif isinstance(other, numbers.Integral):
03099 * self.instance = new IndexSet(<int>other)
03100 * elif isinstance(other, (set, frozenset)):
03101 */
03102 __Pyx_GetModuleGlobalName(__pyx_t_4, __pyx_n_s_numbers); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 96,
__pyx_L1_error)
03103 __Pyx_GOTREF(__pyx_t_4);
03104 __pyx_t_5 = __Pyx_PyObject_GetAttrStr(__pyx_t_4, __pyx_n_s_Integral); if (unlikely(!__pyx_t_5))
__PYX_ERR(0, 96, __pyx_L1_error)
03105 __Pyx_GOTREF(__pyx_t_5);
03106 __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
03107 __pyx_t_2 = PyObject_IsInstance(__pyx_v_other, __pyx_t_5); if (unlikely(__pyx_t_2 == ((int)-1)))
__PYX_ERR(0, 96, __pyx_L1_error)
03108 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
03109 __pyx_t_1 = (__pyx_t_2 != 0);
03110 if (__pyx_t_1) {
03111
03112 /* "PyClical.pyx":97
03113 * self.instance = new IndexSet((<index_set>other).unwrap())
03114 * elif isinstance(other, numbers.Integral):
03115 * self.instance = new IndexSet(<int>other)
03116 * elif isinstance(other, (set, frozenset)):
03117 * try:
03118 */
03119 __pyx_t_6 = __Pyx_PyInt_As_int(__pyx_v_other); if (unlikely((__pyx_t_6 == (int)-1) &&
PyErr_Occurred())) __PYX_ERR(0, 97, __pyx_L1_error)
03120 try {
03121 __pyx_t_3 = new IndexSet(((int) __pyx_t_6));
03122 } catch (...) {
03123 __Pyx_CppExn2PyErr();
03124 __PYX_ERR(0, 97, __pyx_L1_error)
03125 }
03126 __pyx_v_self->instance = __pyx_t_3;
03127
03128 /* "PyClical.pyx":96
03129 * if isinstance(other, index_set):
03130 * self.instance = new IndexSet((<index_set>other).unwrap())
03131 * elif isinstance(other, numbers.Integral):
03132 * self.instance = new IndexSet(<int>other)
03133 * elif isinstance(other, (set, frozenset)):
03134 */
03135 goto __pyx_L3;
03136 }
03137
03138 /* "PyClical.pyx":98
03139 * elif isinstance(other, numbers.Integral):
03140 * self.instance = new IndexSet(<int>other)
03141 * elif isinstance(other, (set, frozenset)):
03142 * try:
03143 * self.instance = new IndexSet()
03144 */
03145 __pyx_t_2 = PySet_Check(__pyx_v_other);
03146 __pyx_t_7 = (__pyx_t_2 != 0);
03147 if (!__pyx_t_7) {
03148 } else {
03149 __pyx_t_1 = __pyx_t_7;
03150 goto __pyx_L4_bool_binop_done;
03151 }
03152 __pyx_t_7 = PyFrozenSet_Check(__pyx_v_other);
03153 __pyx_t_2 = (__pyx_t_7 != 0);
03154 __pyx_t_1 = __pyx_t_2;
03155 __pyx_L4_bool_binop_done;
03156 __pyx_t_2 = (__pyx_t_1 != 0);
03157 if (__pyx_t_2) {
03158
03159 /* "PyClical.pyx":99
03160 * self.instance = new IndexSet(<int>other)
03161 * elif isinstance(other, (set, frozenset)):
03162 * try:
03163 * self.instance = new IndexSet()
03164 * for idx in other:
03165 */
03166 {
03167 __Pyx_PyThreadState_declare
03168 __Pyx_PyThreadState_assign
03169 __Pyx_ExceptionSave(&__pyx_t_8, &__pyx_t_9, &__pyx_t_10);
03170 __Pyx_XGOTREF(__pyx_t_8);
03171 __Pyx_XGOTREF(__pyx_t_9);
03172 __Pyx_XGOTREF(__pyx_t_10);
03173 /*try:*/ {
03174

```

```

03175 /* "PyClical.pyx":100
03176 * elif isinstance(other, (set, frozenset)):
03177 * try:
03178 * self.instance = new IndexSet() # ««««««««
03179 * for idx in other:
03180 * self[idx] = True
03181 */
03182 __pyx_t_3 = new IndexSet();
03183 __pyx_v_self->instance = __pyx_t_3;
03184
03185 /* "PyClical.pyx":101
03186 * try:
03187 * self.instance = new IndexSet()
03188 * for idx in other: # ««««««««
03189 * self[idx] = True
03190 * except IndexError:
03191 */
03192 if (likely(PyList_CheckExact(__pyx_v_other) || PyTuple_CheckExact(__pyx_v_other)) {
03193 __pyx_t_5 = __pyx_v_other; __Pyx_INCREF(__pyx_t_5); __pyx_t_11 = 0;
03194 __pyx_t_12 = NULL;
03195 } else {
03196 __pyx_t_11 = -1; __pyx_t_5 = PyObject_GetIter(__pyx_v_other); if (unlikely(!__pyx_t_5))
03197 __PYX_ERR(0, 101, __pyx_L6_error)
03198 __Pyx_GOTREF(__pyx_t_5);
03199 __pyx_t_12 = Py_TYPE(__pyx_t_5)->tp_iternext; if (unlikely(!__pyx_t_12)) __PYX_ERR(0, 101,
03200 __pyx_L6_error)
03201 }
03202 for (;;) {
03203 if (likely(!__pyx_t_12)) {
03204 if (likely(PyList_CheckExact(__pyx_t_5)) {
03205 if (__pyx_t_11 >= PyList_GET_SIZE(__pyx_t_5)) break;
03206 #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
03207 __pyx_t_4 = PyList_GET_ITEM(__pyx_t_5, __pyx_t_11); __Pyx_INCREF(__pyx_t_4);
03208 __pyx_t_11++; if (unlikely(0 < 0)) __PYX_ERR(0, 101, __pyx_L6_error)
03209 #else
03210 __pyx_t_4 = PySequence_ITEM(__pyx_t_5, __pyx_t_11); __pyx_t_11++; if
03211 (unlikely(!__pyx_t_4)) __PYX_ERR(0, 101, __pyx_L6_error)
03212 __Pyx_GOTREF(__pyx_t_4);
03213 #endif
03214 } else {
03215 if (__pyx_t_11 >= PyTuple_GET_SIZE(__pyx_t_5)) break;
03216 #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
03217 __pyx_t_4 = PyTuple_GET_ITEM(__pyx_t_5, __pyx_t_11); __Pyx_INCREF(__pyx_t_4);
03218 __pyx_t_11++; if (unlikely(0 < 0)) __PYX_ERR(0, 101, __pyx_L6_error)
03219 #else
03220 __pyx_t_4 = PySequence_ITEM(__pyx_t_5, __pyx_t_11); __pyx_t_11++; if
03221 (unlikely(!__pyx_t_4)) __PYX_ERR(0, 101, __pyx_L6_error)
03222 __Pyx_GOTREF(__pyx_t_4);
03223 #endif
03224 }
03225 } else {
03226 __pyx_t_4 = __pyx_t_12(__pyx_t_5);
03227 if (unlikely(!__pyx_t_4)) {
03228 PyObject* exc_type = PyErr_Occurred();
03229 if (exc_type) {
03230 if (likely(__Pyx_PyErr_GivenExceptionMatches(exc_type, PyExc_StopIteration)))
03231 PyErr_Clear();
03232 else __PYX_ERR(0, 101, __pyx_L6_error)
03233 }
03234 break;
03235 }
03236 __Pyx_GOTREF(__pyx_t_4);
03237 }
03238 __Pyx_XDECREF_SET(__pyx_v_idx, __pyx_t_4);
03239 __pyx_t_4 = 0;
03240
03241 /* "PyClical.pyx":102
03242 * self.instance = new IndexSet()
03243 * for idx in other:
03244 * self[idx] = True # ««««««««
03245 * except IndexError:
03246 * raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")
03247 */
03248 if (unlikely(PyObject_SetItem((PyObject *)__pyx_v_self, __pyx_v_idx, Py_True) < 0))
03249 __PYX_ERR(0, 102, __pyx_L6_error)
03250
03251 /* "PyClical.pyx":101
03252 * try:
03253 * self.instance = new IndexSet()
03254 * for idx in other: # ««««««««
03255 * self[idx] = True
03256 * except IndexError:
03257 */
03258 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
03259
03260 /* "PyClical.pyx":99

```



```

03254 * self.instance = new IndexSet(<int>other)
03255 * elif isinstance(other, (set, frozenset)):
03256 * try:
03257 * self.instance = new IndexSet()
03258 * for idx in other:
03259 */
03260 }
03261 __Pyx_XDECREF(__pyx_t_8); __pyx_t_8 = 0;
03262 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
03263 __Pyx_XDECREF(__pyx_t_10); __pyx_t_10 = 0;
03264 goto __pyx_L11_try_end;
03265 __pyx_L6_error:;
03266 __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
03267 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
03268
03269 /* "PyClical.pyx":103
03270 * for idx in other:
03271 * self[idx] = True
03272 * except IndexError:
03273 * # ««««««««
03274 * raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")
03275 * except (RuntimeError, TypeError):
03276 */
03276 __pyx_t_6 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_IndexError);
03277 if (__pyx_t_6) {
03278 __Pyx_AddTraceback("PyClical.index_set.__cinit__", __pyx_clineno, __pyx_lineno,
03279 __pyx_filename);
03279 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_4, &__pyx_t_13) < 0) __PYX_ERR(0, 103,
03280 __pyx_L8_except_error)
03281 __Pyx_GOTREF(__pyx_t_5);
03282 __Pyx_GOTREF(__pyx_t_4);
03283 __Pyx_GOTREF(__pyx_t_13);
03284
03285 /* "PyClical.pyx":104
03286 * self[idx] = True
03287 * except IndexError:
03288 * raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")
03289 * except (RuntimeError, TypeError):
03290 * raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
03291 */
03291 __pyx_t_14 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_invalid); if
03292 (unlikely(!__pyx_t_14)) __PYX_ERR(0, 104, __pyx_L8_except_error)
03293 __Pyx_GOTREF(__pyx_t_14);
03294 __pyx_t_15 = PyObject_Repr(__pyx_v_other); if (unlikely(!__pyx_t_15)) __PYX_ERR(0, 104,
03295 __pyx_L8_except_error)
03296 __Pyx_GOTREF(__pyx_t_15);
03297 __pyx_t_16 = PyNumber_Add(__pyx_t_14, __pyx_t_15); if (unlikely(!__pyx_t_16)) __PYX_ERR(0,
03298 104, __pyx_L8_except_error)
03299 __Pyx_GOTREF(__pyx_t_16);
03300 __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
03301 __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
03302 __pyx_t_15 = PyNumber_Add(__pyx_t_16, __pyx_kp_u_); if (unlikely(!__pyx_t_15)) __PYX_ERR(0,
03303 104, __pyx_L8_except_error)
03304 __Pyx_GOTREF(__pyx_t_15);
03305 __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
03306 __pyx_t_16 = __Pyx_PyObject_CallOneArg(__pyx_builtin_IndexError, __pyx_t_15); if
03307 (unlikely(!__pyx_t_16)) __PYX_ERR(0, 104, __pyx_L8_except_error)
03308 __Pyx_GOTREF(__pyx_t_16);
03309 __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
03310 __Pyx_Raise(__pyx_t_16, 0, 0, 0);
03311 __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
03312 __PYX_ERR(0, 104, __pyx_L8_except_error)
03313 }
03314
03315 /* "PyClical.pyx":105
03316 * except IndexError:
03317 * raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")
03318 * except (RuntimeError, TypeError):
03319 * # ««««««««
03320 * raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
03321 * elif isinstance(other, str):
03322 */
03322 __pyx_t_6 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_RuntimeError) ||
03323 __Pyx_PyErr_ExceptionMatches(__pyx_builtin_TypeError);
03324 if (__pyx_t_6) {
03325 __Pyx_AddTraceback("PyClical.index_set.__cinit__", __pyx_clineno, __pyx_lineno,
03326 __pyx_filename);
03327 if (__Pyx_GetException(&__pyx_t_13, &__pyx_t_4, &__pyx_t_5) < 0) __PYX_ERR(0, 105,
03328 __pyx_L8_except_error)
03329 __Pyx_GOTREF(__pyx_t_13);
03330 __Pyx_GOTREF(__pyx_t_4);
03331 __Pyx_GOTREF(__pyx_t_5);
03332
03333 /* "PyClical.pyx":106
03334 * raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")
03335 * except (RuntimeError, TypeError):
03336 * raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
03337 */
03337 raise IndexError(error_msg_prefix + " invalid " + repr(other) + ".")
03338 }
03339 except (RuntimeError, TypeError):
03340 raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
03341 #
03342 ««««««««

```

```

03329 * elif isinstance(other, str):
03330 * try:
03331 */
03332 __pyx_t_16 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_invalid); if
(unlikely(!__pyx_t_16)) __PYX_ERR(0, 106, __pyx_L8_except_error)
03333 __Pyx_GOTREF(__pyx_t_16);
03334 __pyx_t_15 = PyObject_Repr(__pyx_v_other); if (unlikely(!__pyx_t_15)) __PYX_ERR(0, 106,
__pyx_L8_except_error)
03335 __Pyx_GOTREF(__pyx_t_15);
03336 __pyx_t_14 = PyNumber_Add(__pyx_t_16, __pyx_t_15); if (unlikely(!__pyx_t_14)) __PYX_ERR(0,
106, __pyx_L8_except_error)
03337 __Pyx_GOTREF(__pyx_t_14);
03338 __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
03339 __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
03340 __pyx_t_15 = PyNumber_Add(__pyx_t_14, __pyx_kp_u_); if (unlikely(!__pyx_t_15)) __PYX_ERR(0,
106, __pyx_L8_except_error)
03341 __Pyx_GOTREF(__pyx_t_15);
03342 __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
03343 __pyx_t_14 = __Pyx_PyObject_CallOneArg(__pyx_builtin_ValueError, __pyx_t_15); if
(unlikely(!__pyx_t_14)) __PYX_ERR(0, 106, __pyx_L8_except_error)
03344 __Pyx_GOTREF(__pyx_t_14);
03345 __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
03346 __Pyx_Raise(__pyx_t_14, 0, 0, 0);
03347 __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
03348 __PYX_ERR(0, 106, __pyx_L8_except_error)
03349 }
03350 goto __pyx_L8_except_error;
03351 __pyx_L8_except_error:;
03352
03353 /* "PyClical.pyx":99
03354 * self.instance = new IndexSet(<int>other)
03355 * elif isinstance(other, (set, frozenset)):
03356 * try:
03357 * # ««««««««
03358 * self.instance = new IndexSet()
03359 * for idx in other:
03359 */
03360 __Pyx_XGIVREF(__pyx_t_8);
03361 __Pyx_XGIVREF(__pyx_t_9);
03362 __Pyx_XGIVREF(__pyx_t_10);
03363 __Pyx_ExceptionReset(__pyx_t_8, __pyx_t_9, __pyx_t_10);
03364 goto __pyx_L1_error;
03365 __pyx_L11_try_end:;
03366 }
03367
03368 /* "PyClical.pyx":98
03369 * elif isinstance(other, numbers.Integral):
03370 * self.instance = new IndexSet(<int>other)
03371 * elif isinstance(other, (set, frozenset)):
03372 * try:
03373 * # ««««««««
03374 * self.instance = new IndexSet()
03374 */
03375 goto __pyx_L3;
03376 }
03377
03378 /* "PyClical.pyx":107
03379 * except (RuntimeError, TypeError):
03380 * raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
03381 * elif isinstance(other, str):
03382 * try:
03383 * # ««««««««
03384 * bother = other.encode("UTF-8")
03384 */
03385 __pyx_t_2 = PyUnicode_Check(__pyx_v_other);
03386 __pyx_t_1 = (__pyx_t_2 != 0);
03387 if (likely(__pyx_t_1)) {
03388
03389 /* "PyClical.pyx":108
03390 * raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
03391 * elif isinstance(other, str):
03392 * try:
03393 * # ««««««««
03394 * bother = other.encode("UTF-8")
03395 * self.instance = new IndexSet(<char *>bother)
03395 */
03396 {
03397 __Pyx_PyThreadState_declare
03398 __Pyx_PyThreadState_assign
03399 __Pyx_ExceptionSave(&__pyx_t_10, &__pyx_t_9, &__pyx_t_8);
03400 __Pyx_XGOTREF(__pyx_t_10);
03401 __Pyx_XGOTREF(__pyx_t_9);
03402 __Pyx_XGOTREF(__pyx_t_8);
03403 /*try:*/ {
03404
03405 /* "PyClical.pyx":109
03406 * elif isinstance(other, str):
03407 * try:
03408 * # ««««««««
03409 * bother = other.encode("UTF-8")
03410 * self.instance = new IndexSet(<char *>bother)
03410 * except RuntimeError:

```

```

03411 */
03412 __pyx_t_4 = __Pyx_PyObject_GetAttrStr(__pyx_v_other, __pyx_n_s_encode); if
(unlikely(!__pyx_t_4)) __PYX_ERR(0, 109, __pyx_L18_error)
03413 __Pyx_GOTREF(__pyx_t_4);
03414 __pyx_t_13 = NULL;
03415 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_4))) {
03416 __pyx_t_13 = PyMethod_GET_SELF(__pyx_t_4);
03417 if (likely(__pyx_t_13)) {
03418 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_4);
03419 __Pyx_INCREF(__pyx_t_13);
03420 __Pyx_INCREF(function);
03421 __Pyx_DECREF_SET(__pyx_t_4, function);
03422 }
03423 }
03424 __pyx_t_5 = (__pyx_t_13) ? __Pyx_PyObject_Call2Args(__pyx_t_4, __pyx_t_13, __pyx_kp_u_UTF_8) :
__Pyx_PyObject_CallOneArg(__pyx_t_4, __pyx_kp_u_UTF_8);
03425 __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
03426 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 109, __pyx_L18_error)
03427 __Pyx_GOTREF(__pyx_t_5);
03428 __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
03429 __pyx_v_bother = __pyx_t_5;
03430 __pyx_t_5 = 0;
03431
03432 /* "PyClical.pyx":110
03433 * try:
03434 * bother = other.encode("UTF-8")
03435 * self.instance = new IndexSet(<char *>bother) # ««««««««
03436 * except RuntimeError:
03437 * raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
03438 */
03439 __pyx_t_17 = __Pyx_PyObject_AsWritableString(__pyx_v_bother); if (unlikely(!__pyx_t_17) &&
PyErr_Occurred()) __PYX_ERR(0, 110, __pyx_L18_error)
03440 try {
03441 __pyx_t_3 = new IndexSet(((char *)__pyx_t_17));
03442 } catch (...) {
03443 __Pyx_CppExn2PyErr();
03444 __PYX_ERR(0, 110, __pyx_L18_error)
03445 }
03446 __pyx_v_self->instance = __pyx_t_3;
03447
03448 /* "PyClical.pyx":108
03449 * raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
03450 * elif isinstance(other, str):
03451 * try: # ««««««««
03452 * bother = other.encode("UTF-8")
03453 * self.instance = new IndexSet(<char *>bother)
03454 */
03455 }
03456 __Pyx_XDECREF(__pyx_t_10); __pyx_t_10 = 0;
03457 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
03458 __Pyx_XDECREF(__pyx_t_8); __pyx_t_8 = 0;
03459 goto __pyx_L23_try_end;
03460 __pyx_L18_error;;
03461 __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
03462 __Pyx_XDECREF(__pyx_t_14); __pyx_t_14 = 0;
03463 __Pyx_XDECREF(__pyx_t_15); __pyx_t_15 = 0;
03464 __Pyx_XDECREF(__pyx_t_16); __pyx_t_16 = 0;
03465 __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
03466 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
03467
03468 /* "PyClical.pyx":111
03469 * bother = other.encode("UTF-8")
03470 * self.instance = new IndexSet(<char *>bother)
03471 * except RuntimeError: # ««««««««
03472 * raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
03473 * else:
03474 */
03475 __pyx_t_6 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_RuntimeError);
03476 if (__pyx_t_6) {
03477 __Pyx_AddTraceback("PyClical.index_set.__cinit__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
03478 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_4, &__pyx_t_13) < 0) __PYX_ERR(0, 111,
__pyx_L20_except_error)
03479 __Pyx_GOTREF(__pyx_t_5);
03480 __Pyx_GOTREF(__pyx_t_4);
03481 __Pyx_GOTREF(__pyx_t_13);
03482
03483 /* "PyClical.pyx":112
03484 * self.instance = new IndexSet(<char *>bother)
03485 * except RuntimeError:
03486 * raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
03487 * else:
03488 * raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
03489 */
03490 __pyx_t_14 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_invalid_string); if
(unlikely(!__pyx_t_14)) __PYX_ERR(0, 112, __pyx_L20_except_error)

```

```

03491 __Pyx_GOTREF(__pyx_t_14);
03492 __pyx_t_15 = PyObject_Repr(__pyx_v_other); if (unlikely(!__pyx_t_15)) __PYX_ERR(0, 112,
__pyx_L20_except_error)
03493 __Pyx_GOTREF(__pyx_t_15);
03494 __pyx_t_16 = PyNumber_Add(__pyx_t_14, __pyx_t_15); if (unlikely(!__pyx_t_16)) __PYX_ERR(0,
112, __pyx_L20_except_error)
03495 __Pyx_GOTREF(__pyx_t_16);
03496 __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
03497 __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
03498 __pyx_t_15 = PyNumber_Add(__pyx_t_16, __pyx_kp_u); if (unlikely(!__pyx_t_15)) __PYX_ERR(0,
112, __pyx_L20_except_error)
03499 __Pyx_GOTREF(__pyx_t_15);
03500 __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
03501 __pyx_t_16 = __Pyx_PyObject_CallOneArg(__pyx_builtin_ValueError, __pyx_t_15); if
(unlikely(!__pyx_t_16)) __PYX_ERR(0, 112, __pyx_L20_except_error)
03502 __Pyx_GOTREF(__pyx_t_16);
03503 __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
03504 __Pyx_Raise(__pyx_t_16, 0, 0, 0);
03505 __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
03506 __PYX_ERR(0, 112, __pyx_L20_except_error)
03507 }
03508 goto __pyx_L20_except_error;
03509 __pyx_L20_except_error:;
03510
03511 /* "PyClical.pyx":108
03512 * raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
03513 * elif isinstance(other, str):
03514 * try:
03515 * # ««««««««
03516 * bother = other.encode("UTF-8")
03517 * self.instance = new IndexSet(<char *>bother)
03518 */
03518 __Pyx_XGIVEREF(__pyx_t_10);
03519 __Pyx_XGIVEREF(__pyx_t_9);
03520 __Pyx_XGIVEREF(__pyx_t_8);
03521 __Pyx_ExceptionReset(__pyx_t_10, __pyx_t_9, __pyx_t_8);
03522 goto __pyx_L1_error;
03523 __pyx_L23_try_end:;
03524 }
03525
03526 /* "PyClical.pyx":107
03527 * except (RuntimeError, TypeError):
03528 * raise ValueError(error_msg_prefix + " invalid " + repr(other) + ".")
03529 * elif isinstance(other, str):
03530 * # ««««««««
03531 * try:
03532 * bother = other.encode("UTF-8")
03533 */
03533 goto __pyx_L3;
03534 }
03535
03536 /* "PyClical.pyx":114
03537 * raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
03538 * else:
03539 * raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
03540 * # ««««««««
03541 * def __dealloc__(self):
03542 */
03542 /*else*/ {
03543 __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_2); if
(unlikely(!__pyx_t_13)) __PYX_ERR(0, 114, __pyx_L1_error)
03544 __Pyx_GOTREF(__pyx_t_13);
03545 __pyx_t_4 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject
*)Py_TYPE(__pyx_v_other))); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 114, __pyx_L1_error)
03546 __Pyx_GOTREF(__pyx_t_4);
03547 __pyx_t_5 = __Pyx_PyUnicode_Concat(__pyx_t_13, __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0,
114, __pyx_L1_error)
03548 __Pyx_GOTREF(__pyx_t_5);
03549 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
03550 __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
03551 __pyx_t_4 = __Pyx_PyUnicode_Concat(__pyx_t_5, __pyx_kp_u); if (unlikely(!__pyx_t_4)) __PYX_ERR(0,
114, __pyx_L1_error)
03552 __Pyx_GOTREF(__pyx_t_4);
03553 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
03554 __pyx_t_5 = __Pyx_PyObject_CallOneArg(__pyx_builtin_TypeError, __pyx_t_4); if
(unlikely(!__pyx_t_5)) __PYX_ERR(0, 114, __pyx_L1_error)
03555 __Pyx_GOTREF(__pyx_t_5);
03556 __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
03557 __Pyx_Raise(__pyx_t_5, 0, 0, 0);
03558 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
03559 __PYX_ERR(0, 114, __pyx_L1_error)
03560 }
03561 __pyx_L3:;
03562
03563 /* "PyClical.pyx":74
03564 * return index_set(self)
03565 *
03566 *
03567 * def __cinit__(self, other = 0):
03568 * """

```

```

03569 * Construct an object of type index_set.
03570 */
03571
03572 /* function exit code */
03573 __pyx_r = 0;
03574 goto __pyx_L0;
03575 __pyx_L1_error:;
03576 __Pyx_XDECREF(__pyx_t_4);
03577 __Pyx_XDECREF(__pyx_t_5);
03578 __Pyx_XDECREF(__pyx_t_13);
03579 __Pyx_XDECREF(__pyx_t_14);
03580 __Pyx_XDECREF(__pyx_t_15);
03581 __Pyx_XDECREF(__pyx_t_16);
03582 __Pyx_AddTraceback("PyClical.index_set.__cinit__", __pyx_clineno, __pyx_lineno, __pyx_filename);
03583 __pyx_r = -1;
03584 __pyx_L0:;
03585 __Pyx_XDECREF(__pyx_v_error_msg_prefix);
03586 __Pyx_XDECREF(__pyx_v_idx);
03587 __Pyx_XDECREF(__pyx_v_bother);
03588 __Pyx_RefNannyFinishContext();
03589 return __pyx_r;
03590 }
03591
03592 /* "PyClical.pyx":116
03593 * raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
03594 *
03595 * def __dealloc__(self):
03596 * """
03597 * Clean up by deallocating the instance of C++ class IndexSet.
03598 */
03599
03600 /* Python wrapper */
03601 static void __pyx_pw_8PyClical_9index_set_5__dealloc__(PyObject *__pyx_v_self); /*proto*/
03602 static void __pyx_pw_8PyClical_9index_set_5__dealloc__(PyObject *__pyx_v_self) {
03603 __Pyx_RefNannyDeclarations
03604 __Pyx_RefNannySetupContext("__dealloc__ (wrapper)", 0);
03605 __pyx_pf_8PyClical_9index_set_4__dealloc__(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_self));
03606
03607 /* function exit code */
03608 __Pyx_RefNannyFinishContext();
03609 }
03610
03611 static void __pyx_pf_8PyClical_9index_set_4__dealloc__(struct __pyx_obj_8PyClical_index_set
03612 *__pyx_v_self) {
03613 __Pyx_RefNannyDeclarations
03614 __Pyx_RefNannySetupContext("__dealloc__", 0);
03615
03616 /* "PyClical.pyx":120
03617 * Clean up by deallocating the instance of C++ class IndexSet.
03618 * """
03619 * del self.instance
03620 * # <<<<<<<<
03621 *
03622 * def __richcmp__(lhs, rhs, int op):
03623 * """
03624 * Clean up by deallocating the instance of C++ class IndexSet.
03625 */
03626
03627 delete __pyx_v_self->instance;
03628
03629 /* "PyClical.pyx":116
03630 * raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
03631 *
03632 * def __dealloc__(self):
03633 * """
03634 * Clean up by deallocating the instance of C++ class IndexSet.
03635 */
03636
03637 __Pyx_RefNannyFinishContext();
03638 }
03639
03640 /* "PyClical.pyx":122
03641 * del self.instance
03642 *
03643 * def __richcmp__(lhs, rhs, int op):
03644 * """
03645 * Compare two objects of class index_set.
03646 */
03647
03648 /* Python wrapper */
03649 static PyObject *__pyx_pw_8PyClical_9index_set_7__richcmp__(PyObject *__pyx_v_lhs, PyObject
03650 *__pyx_v_rhs, int __pyx_v_op); /*proto*/
03651 static PyObject *__pyx_pw_8PyClical_9index_set_7__richcmp__(PyObject *__pyx_v_lhs, PyObject
03652 *__pyx_v_rhs, int __pyx_v_op) {
03653 PyObject *__pyx_r = 0;
03654 __Pyx_RefNannyDeclarations
03655 __Pyx_RefNannySetupContext("__richcmp__ (wrapper)", 0);
03656 __pyx_r = __pyx_pf_8PyClical_9index_set_6__richcmp__(((struct __pyx_obj_8PyClical_index_set
03657 *)__pyx_v_lhs), ((PyObject *)__pyx_v_rhs), ((int)__pyx_v_op));
03658 }

```

```

03652 /* function exit code */
03653 __Pyx_RefNannyFinishContext();
03654 return __pyx_r;
03655 }
03656
03657 static PyObject *__pyx_pf_8PyClical_9index_set_6__richcmp__(struct __pyx_obj_8PyClical_index_set
*__pyx_v_lhs, PyObject *__pyx_v_rhs, int __pyx_v_op) {
03658 PyObject *__pyx_v_eq = NULL;
03659 PyObject *__pyx_v_lt = NULL;
03660 PyObject *__pyx_r = NULL;
03661 __Pyx_RefNannyDeclarations
03662 int __pyx_t_1;
03663 int __pyx_t_2;
03664 int __pyx_t_3;
03665 PyObject *__pyx_t_4 = NULL;
03666 int __pyx_lineno = 0;
03667 const char *__pyx_filename = NULL;
03668 int __pyx_clineno = 0;
03669 __Pyx_RefNannySetupContext("__richcmp__", 0);
03670
03671 /* "PyClical.pyx":143
03672 * False
03673 * """
03674 * if (lhs is None) or (rhs is None): # ««««««««
03675 * eq = bool(lhs is rhs)
03676 * if op == 2: # ==
03677 */
03678 __pyx_t_2 = (((PyObject *)__pyx_v_lhs) == Py_None);
03679 __pyx_t_3 = (__pyx_t_2 != 0);
03680 if (!__pyx_t_3) {
03681 } else {
03682 __pyx_t_1 = __pyx_t_3;
03683 goto __pyx_L4_bool_binop_done;
03684 }
03685 __pyx_t_3 = (__pyx_v_rhs == Py_None);
03686 __pyx_t_2 = (__pyx_t_3 != 0);
03687 __pyx_t_1 = __pyx_t_2;
03688 __pyx_L4_bool_binop_done;
03689 if (__pyx_t_1) {
03690
03691 /* "PyClical.pyx":144
03692 * """
03693 * if (lhs is None) or (rhs is None):
03694 * eq = bool(lhs is rhs) # ««««««««
03695 * if op == 2: # ==
03696 * return eq
03697 */
03698 __pyx_t_1 = (((PyObject *)__pyx_v_lhs) == __pyx_v_rhs);
03699 __pyx_t_4 = __Pyx_PyBool_FromLong(!(!__pyx_t_1)); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 144,
__pyx_L1_error)
03700 __Pyx_GOTREF(__pyx_t_4);
03701 __pyx_v_eq = __pyx_t_4;
03702 __pyx_t_4 = 0;
03703
03704 /* "PyClical.pyx":145
03705 * if (lhs is None) or (rhs is None):
03706 * eq = bool(lhs is rhs)
03707 * if op == 2: # == # ««««««««
03708 * return eq
03709 * elif op == 3: # !=
03710 */
03711 switch (__pyx_v_op) {
03712 case 2:
03713
03714 /* "PyClical.pyx":146
03715 * eq = bool(lhs is rhs)
03716 * if op == 2: # ==
03717 * return eq # ««««««««
03718 * elif op == 3: # !=
03719 * return not eq
03720 */
03721 __Pyx_XDECREF(__pyx_r);
03722 __Pyx_INCREF(__pyx_v_eq);
03723 __pyx_r = __pyx_v_eq;
03724 goto __pyx_L0;
03725
03726 /* "PyClical.pyx":145
03727 * if (lhs is None) or (rhs is None):
03728 * eq = bool(lhs is rhs)
03729 * if op == 2: # == # ««««««««
03730 * return eq
03731 * elif op == 3: # !=
03732 */
03733 break;
03734 case 3:
03735
03736 /* "PyClical.pyx":148

```

```

03737 * return eq
03738 * elif op == 3: # !=
03739 * return not eq # ««««««««
03740 * else:
03741 * if op == 0: # <
03742 */
03743 __Pyx_XDECREF(__pyx_r);
03744 __pyx_t_1 = __Pyx_PyObject_IsTrue(__pyx_v_eq); if (unlikely(__pyx_t_1 < 0)) __PYX_ERR(0, 148,
__pyx_L1_error)
03745 __pyx_t_4 = __Pyx_PyBool_FromLong(!__pyx_t_1); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 148,
__pyx_L1_error)
03746 __Pyx_GOTREF(__pyx_t_4);
03747 __pyx_r = __pyx_t_4;
03748 __pyx_t_4 = 0;
03749 goto __pyx_L0;
03750
03751 /* "PyClical.pyx":147
03752 * if op == 2: # ==
03753 * return eq
03754 * elif op == 3: # != # ««««««««
03755 * return not eq
03756 * else:
03757 */
03758 break;
03759 default:
03760
03761 /* "PyClical.pyx":150
03762 * return not eq
03763 * else:
03764 * if op == 0: # < # ««««««««
03765 * return False
03766 * elif op == 1: # <=
03767 */
03768 switch (__pyx_v_op) {
03769 case 0:
03770
03771 /* "PyClical.pyx":151
03772 * else:
03773 * if op == 0: # < # ««««««««
03774 * return False
03775 * elif op == 1: # <=
03776 * return eq
03777 */
03778 __Pyx_XDECREF(__pyx_r);
03779 __Pyx_INCREF(Py_False);
03780 __pyx_r = Py_False;
03781 goto __pyx_L0;
03782
03783 /* "PyClical.pyx":150
03784 * return not eq
03785 * else:
03786 * if op == 0: # < # ««««««««
03787 * return False
03788 * elif op == 1: # <=
03789 */
03790 break;
03791 case 1:
03792
03793 /* "PyClical.pyx":153
03794 * return False
03795 * elif op == 1: # <=
03796 * return eq # ««««««««
03797 * elif op == 4: # >
03798 * return False
03799 */
03800 __Pyx_XDECREF(__pyx_r);
03801 __Pyx_INCREF(__pyx_v_eq);
03802 __pyx_r = __pyx_v_eq;
03803 goto __pyx_L0;
03804
03805 /* "PyClical.pyx":152
03806 * if op == 0: # <
03807 * return False
03808 * elif op == 1: # <= # ««««««««
03809 * return eq
03810 * elif op == 4: # >
03811 */
03812 break;
03813 case 4:
03814
03815 /* "PyClical.pyx":155
03816 * return eq
03817 * elif op == 4: # >
03818 * return False # ««««««««
03819 * elif op == 5: # >=
03820 * return eq
03821 */

```

```

03822 __Pyx_XDECREF(__pyx_r);
03823 __Pyx_INCREF(Py_False);
03824 __pyx_r = Py_False;
03825 goto __pyx_L0;
03826
03827 /* "PyClical.pyx":154
03828 * elif op == 1: # <=
03829 * return eq
03830 * elif op == 4: # > # ««««««««
03831 * return False
03832 * elif op == 5: # >=
03833 */
03834 break;
03835 case 5:
03836
03837 /* "PyClical.pyx":157
03838 * return False
03839 * elif op == 5: # >=
03840 * return eq # ««««««««
03841 * else:
03842 * return NotImplemented
03843 */
03844 __Pyx_XDECREF(__pyx_r);
03845 __Pyx_INCREF(__pyx_v_eq);
03846 __pyx_r = __pyx_v_eq;
03847 goto __pyx_L0;
03848
03849 /* "PyClical.pyx":156
03850 * elif op == 4: # >
03851 * return False
03852 * elif op == 5: # >= # ««««««««
03853 * return eq
03854 * else:
03855 */
03856 break;
03857 default:
03858
03859 /* "PyClical.pyx":159
03860 * return eq
03861 * else:
03862 * return NotImplemented # ««««««««
03863 * else:
03864 * eq = bool(toIndexSet(lhs) == toIndexSet(rhs))
03865 */
03866 __Pyx_XDECREF(__pyx_r);
03867 __Pyx_INCREF(__pyx_builtin_NotImplemented);
03868 __pyx_r = __pyx_builtin_NotImplemented;
03869 goto __pyx_L0;
03870 break;
03871 }
03872 break;
03873 }
03874
03875 /* "PyClical.pyx":143
03876 * False
03877 * """
03878 * if (lhs is None) or (rhs is None): # ««««««««
03879 * eq = bool(lhs is rhs)
03880 * if op == 2: # ==
03881 */
03882 }
03883
03884 /* "PyClical.pyx":161
03885 * return NotImplemented
03886 * else:
03887 * eq = bool(toIndexSet(lhs) == toIndexSet(rhs)) # ««««««««
03888 * if op == 2: # ==
03889 * return eq
03890 */
03891 /*else*/ {
03892 __pyx_t_1 = (__pyx_f_8PyClical_toIndexSet(((PyObject *)__pyx_v_lhs)) ==
__pyx_f_8PyClical_toIndexSet(__pyx_v_rhs));
03893 __pyx_t_4 = __Pyx_PyBool_FromLong(!(!__pyx_t_1)); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 161,
__pyx_L1_error)
03894 __Pyx_GOTREF(__pyx_t_4);
03895 __pyx_v_eq = __pyx_t_4;
03896 __pyx_t_4 = 0;
03897
03898 /* "PyClical.pyx":162
03899 * else:
03900 * eq = bool(toIndexSet(lhs) == toIndexSet(rhs))
03901 * if op == 2: # == # ««««««««
03902 * return eq
03903 * elif op == 3: # !=
03904 */
03905 switch (__pyx_v_op) {
03906 case 2:

```



```

03907
03908 /* "PyClical.pyx":163
03909 * eq = bool(toIndexSet(lhs) == toIndexSet(rhs))
03910 * if op == 2: # ==
03911 * return eq # ««««««««
03912 * elif op == 3: # !=
03913 * return not eq
03914 */
03915 __Pyx_XDECREF(__pyx_r);
03916 __Pyx_INCREF(__pyx_v_eq);
03917 __pyx_r = __pyx_v_eq;
03918 goto __pyx_L0;
03919
03920 /* "PyClical.pyx":162
03921 * else:
03922 * eq = bool(toIndexSet(lhs) == toIndexSet(rhs))
03923 * if op == 2: # ==
03924 * return eq # ««««««««
03925 * elif op == 3: # !=
03926 */
03927 break;
03928 case 3:
03929
03930 /* "PyClical.pyx":165
03931 * return eq
03932 * elif op == 3: # !=
03933 * return not eq # ««««««««
03934 * else:
03935 * lt = bool(toIndexSet(lhs) < toIndexSet(rhs))
03936 */
03937 __Pyx_XDECREF(__pyx_r);
03938 __pyx_t_1 = __Pyx_PyObject_IsTrue(__pyx_v_eq); if (unlikely(__pyx_t_1 < 0)) __PYX_ERR(0, 165,
__pyx_L1_error)
03939 __pyx_t_4 = __Pyx_PyBool_FromLong(!__pyx_t_1); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 165,
__pyx_L1_error)
03940 __Pyx_GOTREF(__pyx_t_4);
03941 __pyx_r = __pyx_t_4;
03942 __pyx_t_4 = 0;
03943 goto __pyx_L0;
03944
03945 /* "PyClical.pyx":164
03946 * if op == 2: # ==
03947 * return eq
03948 * elif op == 3: # !=
03949 * return not eq # ««««««««
03950 * else:
03951 */
03952 break;
03953 default:
03954
03955 /* "PyClical.pyx":167
03956 * return not eq
03957 * else:
03958 * lt = bool(toIndexSet(lhs) < toIndexSet(rhs)) # ««««««««
03959 * if op == 0: # <
03960 * return lt
03961 */
03962 __pyx_t_1 = (__pyx_f_8PyClical_toIndexSet((PyObject *)__pyx_v_lhs)) <
__pyx_f_8PyClical_toIndexSet(__pyx_v_rhs));
03963 __pyx_t_4 = __Pyx_PyBool_FromLong(!(!__pyx_t_1)); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 167,
__pyx_L1_error)
03964 __Pyx_GOTREF(__pyx_t_4);
03965 __pyx_v_lt = __pyx_t_4;
03966 __pyx_t_4 = 0;
03967
03968 /* "PyClical.pyx":168
03969 * else:
03970 * lt = bool(toIndexSet(lhs) < toIndexSet(rhs))
03971 * if op == 0: # <
03972 * return lt # ««««««««
03973 * elif op == 1: # <=
03974 */
03975 switch (__pyx_v_op) {
03976 case 0:
03977
03978 /* "PyClical.pyx":169
03979 * lt = bool(toIndexSet(lhs) < toIndexSet(rhs))
03980 * if op == 0: # <
03981 * return lt # ««««««««
03982 * elif op == 1: # <=
03983 * return lt or eq
03984 */
03985 __Pyx_XDECREF(__pyx_r);
03986 __Pyx_INCREF(__pyx_v_lt);
03987 __pyx_r = __pyx_v_lt;
03988 goto __pyx_L0;
03989

```

```

03990 /* "PyClical.pyx":168
03991 * else:
03992 * lt = bool(toIndexSet(lhs) < toIndexSet(rhs))
03993 * if op == 0: # < # ««««««««
03994 * return lt
03995 * elif op == 1: # <=
03996 */
03997 break;
03998 case 1:
03999
04000 /* "PyClical.pyx":171
04001 * return lt
04002 * elif op == 1: # <=
04003 * return lt or eq # ««««««««
04004 * elif op == 4: # >
04005 * return not (lt or eq)
04006 */
04007 __Pyx_XDECREF(__pyx_r);
04008 __pyx_t_1 = __Pyx_PyObject_IsTrue(__pyx_v_lt); if (unlikely(__pyx_t_1 < 0)) __PYX_ERR(0, 171,
__pyx_L1_error)
04009 if (!__pyx_t_1) {
04010 } else {
04011 __Pyx_INCREF(__pyx_v_lt);
04012 __pyx_t_4 = __pyx_v_lt;
04013 goto __pyx_L6_bool_binop_done;
04014 }
04015 __Pyx_INCREF(__pyx_v_eq);
04016 __pyx_t_4 = __pyx_v_eq;
04017 __pyx_L6_bool_binop_done;
04018 __pyx_r = __pyx_t_4;
04019 __pyx_t_4 = 0;
04020 goto __pyx_L0;
04021
04022 /* "PyClical.pyx":170
04023 * if op == 0: # <
04024 * return lt
04025 * elif op == 1: # <= # ««««««««
04026 * return lt or eq
04027 * elif op == 4: # >
04028 */
04029 break;
04030 case 4:
04031
04032 /* "PyClical.pyx":173
04033 * return lt or eq
04034 * elif op == 4: # >
04035 * return not (lt or eq) # ««««««««
04036 * elif op == 5: # >=
04037 * return not lt
04038 */
04039 __Pyx_XDECREF(__pyx_r);
04040 __pyx_t_2 = __Pyx_PyObject_IsTrue(__pyx_v_lt); if (unlikely(__pyx_t_2 < 0)) __PYX_ERR(0, 173,
__pyx_L1_error)
04041 if (!__pyx_t_2) {
04042 } else {
04043 __pyx_t_1 = __pyx_t_2;
04044 goto __pyx_L8_bool_binop_done;
04045 }
04046 __pyx_t_2 = __Pyx_PyObject_IsTrue(__pyx_v_eq); if (unlikely(__pyx_t_2 < 0)) __PYX_ERR(0, 173,
__pyx_L1_error)
04047 __pyx_t_1 = __pyx_t_2;
04048 __pyx_L8_bool_binop_done;
04049 __pyx_t_4 = __Pyx_PyBool_FromLong(!__pyx_t_1); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 173,
__pyx_L1_error)
04050 __Pyx_GOTREF(__pyx_t_4);
04051 __pyx_r = __pyx_t_4;
04052 __pyx_t_4 = 0;
04053 goto __pyx_L0;
04054
04055 /* "PyClical.pyx":172
04056 * elif op == 1: # <=
04057 * return lt or eq
04058 * elif op == 4: # > # ««««««««
04059 * return not (lt or eq)
04060 * elif op == 5: # >=
04061 */
04062 break;
04063 case 5:
04064
04065 /* "PyClical.pyx":175
04066 * return not (lt or eq)
04067 * elif op == 5: # >=
04068 * return not lt # ««««««««
04069 * else:
04070 * return NotImplemented
04071 */
04072 __Pyx_XDECREF(__pyx_r);

```

```

04073 __pyx_t_1 = __Pyx_PyObject_IsTrue(__pyx_v_lt); if (unlikely(__pyx_t_1 < 0)) __PYX_ERR(0, 175,
__pyx_L1_error)
04074 __pyx_t_4 = __Pyx_PyBool_FromLong(!__pyx_t_1); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 175,
__pyx_L1_error)
04075 __Pyx_GOTREF(__pyx_t_4);
04076 __pyx_r = __pyx_t_4;
04077 __pyx_t_4 = 0;
04078 goto __pyx_L0;
04079
04080 /* "PyClical.pyx":174
04081 * elif op == 4: # >
04082 * return not (lt or eq)
04083 * elif op == 5: # >= # ««««««««
04084 * return not lt
04085 * else:
04086 */
04087 break;
04088 default:
04089
04090 /* "PyClical.pyx":177
04091 * return not lt
04092 * else:
04093 * return NotImplemented # ««««««««
04094 *
04095 * def __setitem__(self, idx, val):
04096 */
04097 __Pyx_XDECREF(__pyx_r);
04098 __Pyx_INCREF(__pyx_builtin_NotImplemented);
04099 __pyx_r = __pyx_builtin_NotImplemented;
04100 goto __pyx_L0;
04101 break;
04102 }
04103 break;
04104 }
04105 }
04106
04107 /* "PyClical.pyx":122
04108 * del self.instance
04109 *
04110 * def __richcmp__(lhs, rhs, int op): # ««««««««
04111 * """
04112 * Compare two objects of class index_set.
04113 */
04114
04115 /* function exit code */
04116 __pyx_L1_error:;
04117 __Pyx_XDECREF(__pyx_t_4);
04118 __Pyx_AddTraceback("PyClical.index_set.__richcmp__", __pyx_clineno, __pyx_lineno, __pyx_filename);
04119 __pyx_r = NULL;
04120 __pyx_L0:;
04121 __Pyx_XDECREF(__pyx_v_eq);
04122 __Pyx_XDECREF(__pyx_v_lt);
04123 __Pyx_XGIVEREF(__pyx_r);
04124 __Pyx_RefNannyFinishContext();
04125 return __pyx_r;
04126 }
04127
04128 /* "PyClical.pyx":179
04129 * return NotImplemented
04130 *
04131 * def __setitem__(self, idx, val): # ««««««««
04132 * """
04133 * Set the value of an index_set object at index idx to value val.
04134 */
04135
04136 /* Python wrapper */
04137 static int __pyx_pw_8PyClical_9index_set_9__setitem__(PyObject *__pyx_v_self, PyObject *__pyx_v_idx,
PyObject *__pyx_v_val); /*proto*/
04138 static char __pyx_doc_8PyClical_9index_set_8__setitem__[] = "\n Set the value of an index_set
object at index idx to value val.\n\n >> s=index_set({1}); s[2] = True; print(s)\n
{1,2}\n >> s=index_set({1,2}); s[1] = False; print(s)\n {2}\n ";
04139 #if CYTHON_COMPILING_IN_CPYTHON
04140 struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_8__setitem__;
04141 #endif
04142 static int __pyx_pw_8PyClical_9index_set_9__setitem__(PyObject *__pyx_v_self, PyObject *__pyx_v_idx,
PyObject *__pyx_v_val) {
04143 int __pyx_r;
04144 __Pyx_RefNannyDeclarations
04145 __Pyx_RefNannySetupContext("__setitem__ (wrapper)", 0);
04146 __pyx_r = __pyx_pf_8PyClical_9index_set_8__setitem__(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self), ((PyObject *)__pyx_v_idx), ((PyObject *)__pyx_v_val));
04147
04148 /* function exit code */
04149 __Pyx_RefNannyFinishContext();
04150 return __pyx_r;
04151 }
04152

```

```

04153 static int __pyx_pf_8PyClical_9index_set_8_setitem__(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self, PyObject *__pyx_v_idx, PyObject *__pyx_v_val) {
04154 int __pyx_r;
04155 __Pyx_RefNannyDeclarations
04156 int __pyx_t_1;
04157 int __pyx_t_2;
04158 int __pyx_lineno = 0;
04159 const char *__pyx_filename = NULL;
04160 int __pyx_clineno = 0;
04161 __Pyx_RefNannySetupContext("__setitem__", 0);
04162
04163 /* "PyClical.pyx":188
04164 * {2}
04165 * """
04166 * self.instance.set(idx, val) # ««««««««
04167 * return
04168 *
04169 */
04170 __pyx_t_1 = __Pyx_PyInt_As_int(__pyx_v_idx); if (unlikely((__pyx_t_1 == (int)-1) &&
PyErr_Occurred())) __PYX_ERR(0, 188, __pyx_L1_error)
04171 __pyx_t_2 = __Pyx_PyInt_As_int(__pyx_v_val); if (unlikely((__pyx_t_2 == (int)-1) &&
PyErr_Occurred())) __PYX_ERR(0, 188, __pyx_L1_error)
04172 try {
04173 __pyx_v_self->instance->set(__pyx_t_1, __pyx_t_2);
04174 } catch (...) {
04175 __Pyx_CppExn2PyErr();
04176 __PYX_ERR(0, 188, __pyx_L1_error)
04177 }
04178
04179 /* "PyClical.pyx":189
04180 * """
04181 * self.instance.set(idx, val)
04182 * return # ««««««««
04183 *
04184 * def __getitem__(self, idx):
04185 */
04186 __pyx_r = 0;
04187 goto __pyx_L0;
04188
04189 /* "PyClical.pyx":179
04190 * return NotImplemented
04191 *
04192 * def __setitem__(self, idx, val): # ««««««««
04193 * """
04194 * Set the value of an index_set object at index idx to value val.
04195 */
04196
04197 /* function exit code */
04198 __pyx_L1_error:;
04199 __Pyx_AddTraceback("PyClical.index_set.__setitem__", __pyx_clineno, __pyx_lineno, __pyx_filename);
04200 __pyx_r = -1;
04201 __pyx_L0:;
04202 __Pyx_RefNannyFinishContext();
04203 return __pyx_r;
04204 }
04205
04206 /* "PyClical.pyx":191
04207 * return
04208 *
04209 * def __getitem__(self, idx): # ««««««««
04210 * """
04211 * Get the value of an index_set object at an index.
04212 */
04213
04214 /* Python wrapper */
04215 static PyObject *__pyx_pw_8PyClical_9index_set_11_getitem__(PyObject *__pyx_v_self, PyObject
*__pyx_v_idx); /*proto*/
04216 static char __pyx_doc_8PyClical_9index_set_10_getitem__[] = "\n Get the value of an index_set
object at an index.\n\n >> index_set({1})[1]\n True\n >> index_set({1})[2]\n False\n >> index_set({2})[-1]\n False\n >> index_set({2})[1]\n False\n >> index_set({2})[2]\n True\n >> index_set({2})[33]\n False\n ";
04217 #if CYTHON_COMPILING_IN_CPYTHON
04218 struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_10_getitem__;
04219 #endif
04220 static PyObject *__pyx_pw_8PyClical_9index_set_11_getitem__(PyObject *__pyx_v_self, PyObject
*__pyx_v_idx) {
04221 PyObject *__pyx_r = 0;
04222 __Pyx_RefNannyDeclarations
04223 __Pyx_RefNannySetupContext("__getitem__ (wrapper)", 0);
04224 __pyx_r = __pyx_pf_8PyClical_9index_set_10_getitem__(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self), ((PyObject *)__pyx_v_idx));
04225
04226 /* function exit code */
04227 __Pyx_RefNannyFinishContext();
04228 return __pyx_r;
04229 }
04230

```

```

04231 static PyObject *__pyx_pf_8PyClical_9index_set_10__getitem__(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self, PyObject *__pyx_v_idx) {
04232 PyObject *__pyx_r = NULL;
04233 __Pyx_RefNannyDeclarations
04234 int __pyx_t_1;
04235 PyObject *__pyx_t_2 = NULL;
04236 int __pyx_lineno = 0;
04237 const char *__pyx_filename = NULL;
04238 int __pyx_clineno = 0;
04239 __Pyx_RefNannySetupContext("__getitem__", 0);
04240
04241 /* "PyClical.pyx":208
04242 * False
04243 * """
04244 * return self.instance.getitem(idx) # ««««««««
04245 *
04246 * def __contains__(self, idx):
04247 */
04248 __Pyx_XDECREF(__pyx_r);
04249 __pyx_t_1 = __Pyx_PyInt_As_int(__pyx_v_idx); if (unlikely((__pyx_t_1 == (int)-1) &&
PyErr_Occurred())) __PYX_ERR(0, 208, __pyx_L1_error)
04250 __pyx_t_2 = __Pyx_PyBool_FromLong(__pyx_v_self->instance->operator[](__pyx_t_1)); if
(unlikely(!__pyx_t_2)) __PYX_ERR(0, 208, __pyx_L1_error)
04251 __Pyx_GOTREF(__pyx_t_2);
04252 __pyx_r = __pyx_t_2;
04253 __pyx_t_2 = 0;
04254 goto __pyx_L0;
04255
04256 /* "PyClical.pyx":191
04257 * return
04258 *
04259 * def __getitem__(self, idx): # ««««««««
04260 * """
04261 * Get the value of an index_set object at an index.
04262 */
04263
04264 /* function exit code */
04265 __pyx_L1_error:;
04266 __Pyx_XDECREF(__pyx_t_2);
04267 __Pyx_AddTraceback("PyClical.index_set.__getitem__", __pyx_clineno, __pyx_lineno, __pyx_filename);
04268 __pyx_r = NULL;
04269 __pyx_L0:;
04270 __Pyx_XGIVEREF(__pyx_r);
04271 __Pyx_RefNannyFinishContext();
04272 return __pyx_r;
04273 }
04274
04275 /* "PyClical.pyx":210
04276 * return self.instance.getitem(idx)
04277 *
04278 * def __contains__(self, idx): # ««««««««
04279 * """
04280 * Check that an index_set object contains the index idx: idx in self.
04281 */
04282
04283 /* Python wrapper */
04284 static int __pyx_pw_8PyClical_9index_set_13__contains__(PyObject *__pyx_v_self, PyObject
*__pyx_v_idx); /*proto*/
04285 static char __pyx_doc_8PyClical_9index_set_12__contains__[] = "\n Check that an index_set
object contains the index idx: idx in self.\n\n >> 1 in index_set({1})\n True\n
>> 2 in index_set({1})\n False\n >> -1 in index_set({2})\n False\n >> 1 in
index_set({2})\n False\n >> 2 in index_set({2})\n True\n >> 33 in
index_set({2})\n False\n ";
04286 #if CYTHON_COMPILING_IN_CPYTHON
04287 struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_12__contains__;
04288 #endif
04289 static int __pyx_pw_8PyClical_9index_set_13__contains__(PyObject *__pyx_v_self, PyObject *__pyx_v_idx)
{
04290 int __pyx_r;
04291 __Pyx_RefNannyDeclarations
04292 __Pyx_RefNannySetupContext("__contains__ (wrapper)", 0);
04293 __pyx_r = __pyx_pf_8PyClical_9index_set_12__contains__(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self), ((PyObject *)__pyx_v_idx));
04294
04295 /* function exit code */
04296 __Pyx_RefNannyFinishContext();
04297 return __pyx_r;
04298 }
04299
04300 static int __pyx_pf_8PyClical_9index_set_12__contains__(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self, PyObject *__pyx_v_idx) {
04301 int __pyx_r;
04302 __Pyx_RefNannyDeclarations
04303 int __pyx_t_1;
04304 int __pyx_lineno = 0;
04305 const char *__pyx_filename = NULL;
04306 int __pyx_clineno = 0;

```

```

04307 __Pyx_RefNannySetupContext("__contains__", 0);
04308
04309 /* "PyClicl.pyx":227
04310 * False
04311 * """
04312 * return self.instance.getitem(idx) # ««««««««
04313 *
04314 * def __iter__(self):
04315 */
04316 __pyx_t_1 = __Pyx_PyInt_As_int(__pyx_v_idx); if (unlikely((__pyx_t_1 == (int)-1) &&
PyErr_Occurred())) __PYX_ERR(0, 227, __pyx_L1_error)
04317 __pyx_r = __pyx_v_self->instance->operator[](__pyx_t_1);
04318 goto __pyx_L0;
04319
04320 /* "PyClicl.pyx":210
04321 * return self.instance.getitem(idx)
04322 *
04323 * def __contains__(self, idx): # ««««««««
04324 * """
04325 * Check that an index_set object contains the index idx: idx in self.
04326 */
04327
04328 /* function exit code */
04329 __pyx_L1_error:;
04330 __Pyx_AddTraceback("PyClicl.index_set.__contains__", __pyx_clineno, __pyx_lineno, __pyx_filename);
04331 __pyx_r = -1;
04332 __pyx_L0:;
04333 __Pyx_RefNannyFinishContext();
04334 return __pyx_r;
04335 }
04336 static PyObject * __pyx_gb_8PyClicl_9index_set_16generator(__pyx_CoroutineObject * __pyx_generator,
CYTHON_UNUSED PyThreadState * __pyx_tstate, PyObject * __pyx_sent_value); /* proto */
04337
04338 /* "PyClicl.pyx":229
04339 * return self.instance.getitem(idx)
04340 *
04341 * def __iter__(self): # ««««««««
04342 * """
04343 * Iterate over the indices of an index_set.
04344 */
04345
04346 /* Python wrapper */
04347 static PyObject * __pyx_pw_8PyClicl_9index_set_15__iter__(PyObject * __pyx_v_self); /*proto*/
04348 static char __pyx_doc_8PyClicl_9index_set_14__iter__[] = "\n Iterate over the indices of an
index_set.\n\n >> for i in index_set({-3,4,7}):print(i, end=\",\\n\")\n -3,4,7,\n ";
04349 #if CYTHON_COMPILING_IN_CPYTHON
04350 struct wrapperbase __pyx_wrapperbase_8PyClicl_9index_set_14__iter__;
04351 #endif
04352 static PyObject * __pyx_pw_8PyClicl_9index_set_15__iter__(PyObject * __pyx_v_self) {
04353 PyObject * __pyx_r = 0;
04354 __Pyx_RefNannyDeclarations
04355 __Pyx_RefNannySetupContext("__iter__ (wrapper)", 0);
04356 __pyx_r = __pyx_pf_8PyClicl_9index_set_14__iter__(((struct __pyx_obj_8PyClicl_index_set
*) __pyx_v_self));
04357
04358 /* function exit code */
04359 __Pyx_RefNannyFinishContext();
04360 return __pyx_r;
04361 }
04362
04363 static PyObject * __pyx_pf_8PyClicl_9index_set_14__iter__(struct __pyx_obj_8PyClicl_index_set
* __pyx_v_self) {
04364 struct __pyx_obj_8PyClicl__pyx_scope_struct__iter__ * __pyx_cur_scope;
04365 PyObject * __pyx_r = NULL;
04366 __Pyx_RefNannyDeclarations
04367 int __pyx_lineno = 0;
04368 const char * __pyx_filename = NULL;
04369 int __pyx_clineno = 0;
04370 __Pyx_RefNannySetupContext("__iter__", 0);
04371 __pyx_cur_scope = (struct __pyx_obj_8PyClicl__pyx_scope_struct__iter__
*) __pyx_tp_new_8PyClicl__pyx_scope_struct__iter__((__pyx_ptype_8PyClicl__pyx_scope_struct__iter__,
__pyx_empty_tuple, NULL);
04372 if (unlikely(!__pyx_cur_scope)) {
04373 __pyx_cur_scope = ((struct __pyx_obj_8PyClicl__pyx_scope_struct__iter__ *)Py_None);
04374 __Pyx_INCREF(Py_None);
04375 __PYX_ERR(0, 229, __pyx_L1_error)
04376 } else {
04377 __Pyx_GOTREF(__pyx_cur_scope);
04378 }
04379 __pyx_cur_scope->__pyx_v_self = __pyx_v_self;
04380 __Pyx_INCREF((PyObject *) __pyx_cur_scope->__pyx_v_self);
04381 __Pyx_GIVEREF((PyObject *) __pyx_cur_scope->__pyx_v_self);
04382 {
04383 __pyx_CoroutineObject *gen = __Pyx_Generator_New((__pyx_coroutine_body_t)
__pyx_gb_8PyClicl_9index_set_16generator, NULL, (PyObject *) __pyx_cur_scope, __pyx_n_s_iter,
__pyx_n_s_index_set__iter__, __pyx_n_s_PyClicl); if (unlikely(!gen)) __PYX_ERR(0, 229, __pyx_L1_error)
04384 __Pyx_DECREF(__pyx_cur_scope);

```

```

04385 __Pyx_RefNannyFinishContext();
04386 return (PyObject *) gen;
04387 }
04388
04389 /* function exit code */
04390 __pyx_l1_error:;
04391 __Pyx_AddTraceback("PyClical.index_set.__iter__", __pyx_clineno, __pyx_lineno, __pyx_filename);
04392 __pyx_r = NULL;
04393 __Pyx_DECREF(((PyObject *) __pyx_cur_scope));
04394 __Pyx_XGIVEREF(__pyx_r);
04395 __Pyx_RefNannyFinishContext();
04396 return __pyx_r;
04397 }
04398
04399 static PyObject * __pyx_gb_8PyClical_9index_set_16generator(__pyx_CoroutineObject * __pyx_generator,
CYTHON_UNUSED PyThreadState * __pyx_tstate, PyObject * __pyx_sent_value) /* generator body */
04400 {
04401 struct __pyx_obj_8PyClical__pyx_scope_struct__iter__ * __pyx_cur_scope = ((struct
__pyx_obj_8PyClical__pyx_scope_struct__iter__ *) __pyx_generator->closure);
04402 PyObject * __pyx_r = NULL;
04403 PyObject * __pyx_t_1 = NULL;
04404 PyObject * __pyx_t_2 = NULL;
04405 PyObject * __pyx_t_3 = NULL;
04406 PyObject * __pyx_t_4 = NULL;
04407 Py_ssize_t __pyx_t_5;
04408 PyObject * (*__pyx_t_6)(PyObject *);
04409 int __pyx_t_7;
04410 int __pyx_t_8;
04411 int __pyx_lineno = 0;
04412 const char * __pyx_filename = NULL;
04413 int __pyx_clineno = 0;
04414 __Pyx_RefNannyDeclarations
04415 __Pyx_RefNannySetupContext("__iter__", 0);
04416 switch (__pyx_generator->resume_label) {
04417 case 0: goto __pyx_L3_first_run;
04418 case 1: goto __pyx_L7_resume_from_yield;
04419 default: /* CPython raises the right error here */
04420 __Pyx_RefNannyFinishContext();
04421 return NULL;
04422 }
04423 __pyx_L3_first_run:;
04424 if (unlikely(!__pyx_sent_value)) __PYX_ERR(0, 229, __pyx_l1_error)
04425
04426 /* "PyClical.pyx":236
04427 * -3,4,7,
04428 * """
04429 * for idx in range(self.min(), self.max()+1):
04430 * if idx in self:
04431 * yield idx
04432 */
04433 __pyx_t_2 = __Pyx_PyObject_GetAttrStr(((PyObject *) __pyx_cur_scope->__pyx_v_self), __pyx_n_s_min);
04434 if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 236, __pyx_l1_error)
04435 __Pyx_GOTREF(__pyx_t_2);
04436 __pyx_t_3 = NULL;
04437 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_2))) {
04438 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_2);
04439 if (likely(__pyx_t_3)) {
04440 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_2);
04441 __Pyx_INCREF(__pyx_t_3);
04442 __Pyx_INCREF(function);
04443 __Pyx_DECREF_SET(__pyx_t_2, function);
04444 }
04445 }
04446 __pyx_t_1 = (__pyx_t_3) ? __Pyx_PyObject_CallOneArg(__pyx_t_2, __pyx_t_3) :
__Pyx_PyObject_CallNoArg(__pyx_t_2);
04447 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
04448 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 236, __pyx_l1_error)
04449 __Pyx_GOTREF(__pyx_t_1);
04450 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
04451 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(((PyObject *) __pyx_cur_scope->__pyx_v_self), __pyx_n_s_max);
04452 if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 236, __pyx_l1_error)
04453 __Pyx_GOTREF(__pyx_t_3);
04454 __pyx_t_4 = NULL;
04455 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
04456 __pyx_t_4 = PyMethod_GET_SELF(__pyx_t_3);
04457 if (likely(__pyx_t_4)) {
04458 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
04459 __Pyx_INCREF(__pyx_t_4);
04460 __Pyx_INCREF(function);
04461 __Pyx_DECREF_SET(__pyx_t_3, function);
04462 }
04463 }
04464 __pyx_t_2 = (__pyx_t_4) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_4) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
04465 __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
04466 if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 236, __pyx_l1_error)
04467 __Pyx_GOTREF(__pyx_t_2);

```

```

04466 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
04467 __pyx_t_3 = __Pyx_PyInt_AddObjC(__pyx_t_2, __pyx_int_1, 1, 0, 0); if (unlikely(!__pyx_t_3))
__PYX_ERR(0, 236, __pyx_L1_error)
04468 __Pyx_GOTREF(__pyx_t_3);
04469 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
04470 __pyx_t_2 = PyTuple_New(2); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 236, __pyx_L1_error)
04471 __Pyx_GOTREF(__pyx_t_2);
04472 __Pyx_GIVEREF(__pyx_t_1);
04473 PyTuple_SET_ITEM(__pyx_t_2, 0, __pyx_t_1);
04474 __Pyx_GIVEREF(__pyx_t_3);
04475 PyTuple_SET_ITEM(__pyx_t_2, 1, __pyx_t_3);
04476 __pyx_t_1 = 0;
04477 __pyx_t_3 = 0;
04478 __pyx_t_3 = __Pyx_PyObject_Call(__pyx_builtin_range, __pyx_t_2, NULL); if (unlikely(!__pyx_t_3))
__PYX_ERR(0, 236, __pyx_L1_error)
04479 __Pyx_GOTREF(__pyx_t_3);
04480 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
04481 if (likely(PyList_CheckExact(__pyx_t_3)) || PyTuple_CheckExact(__pyx_t_3)) {
04482 __pyx_t_2 = __pyx_t_3; __Pyx_INCREF(__pyx_t_2); __pyx_t_5 = 0;
04483 __pyx_t_6 = NULL;
04484 } else {
04485 __pyx_t_5 = -1; __pyx_t_2 = PyObject_GetIter(__pyx_t_3); if (unlikely(!__pyx_t_2)) __PYX_ERR(0,
236, __pyx_L1_error)
04486 __Pyx_GOTREF(__pyx_t_2);
04487 __pyx_t_6 = Py_TYPE(__pyx_t_2)->tp_iternext; if (unlikely(!__pyx_t_6)) __PYX_ERR(0, 236,
__pyx_L1_error)
04488 }
04489 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
04490 for (;;) {
04491 if (likely(!__pyx_t_6)) {
04492 if (likely(PyList_CheckExact(__pyx_t_2))) {
04493 if (__pyx_t_5 >= PyList_GET_SIZE(__pyx_t_2)) break;
04494 #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
04495 __pyx_t_3 = PyList_GET_ITEM(__pyx_t_2, __pyx_t_5); __Pyx_INCREF(__pyx_t_3); __pyx_t_5++; if
(unlikely(0 < 0)) __PYX_ERR(0, 236, __pyx_L1_error)
04496 #else
04497 __pyx_t_3 = PySequence_ITEM(__pyx_t_2, __pyx_t_5); __pyx_t_5++; if (unlikely(!__pyx_t_3))
__PYX_ERR(0, 236, __pyx_L1_error)
04498 __Pyx_GOTREF(__pyx_t_3);
04499 #endif
04500 } else {
04501 if (__pyx_t_5 >= PyTuple_GET_SIZE(__pyx_t_2)) break;
04502 #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
04503 __pyx_t_3 = PyTuple_GET_ITEM(__pyx_t_2, __pyx_t_5); __Pyx_INCREF(__pyx_t_3); __pyx_t_5++; if
(unlikely(0 < 0)) __PYX_ERR(0, 236, __pyx_L1_error)
04504 #else
04505 __pyx_t_3 = PySequence_ITEM(__pyx_t_2, __pyx_t_5); __pyx_t_5++; if (unlikely(!__pyx_t_3))
__PYX_ERR(0, 236, __pyx_L1_error)
04506 __Pyx_GOTREF(__pyx_t_3);
04507 #endif
04508 }
04509 } else {
04510 __pyx_t_3 = __pyx_t_6(__pyx_t_2);
04511 if (unlikely(!__pyx_t_3)) {
04512 PyObject* exc_type = PyErr_Occurred();
04513 if (exc_type) {
04514 if (likely(__Pyx_PyErr_GivenExceptionMatches(exc_type, PyExc_StopIteration))) PyErr_Clear();
04515 else __PYX_ERR(0, 236, __pyx_L1_error)
04516 }
04517 break;
04518 }
04519 __Pyx_GOTREF(__pyx_t_3);
04520 }
04521 __Pyx_XGOTREF(__pyx_cur_scope->__pyx_v_idx);
04522 __Pyx_XDECREF_SET(__pyx_cur_scope->__pyx_v_idx, __pyx_t_3);
04523 __Pyx_GIVEREF(__pyx_t_3);
04524 __pyx_t_3 = 0;
04525
04526 /* "PyClical.pyx":237
04527 *
04528 * for idx in range(self.min(), self.max()+1):
04529 * if idx in self:
04530 * # ««««««««
04531 * yield idx
04532 */
04533 __pyx_t_7 = (__Pyx_PySequence_ContainsTF(__pyx_cur_scope->__pyx_v_idx, ((PyObject
*)__pyx_cur_scope->__pyx_v_self), Py_EQ)); if (unlikely(__pyx_t_7 < 0)) __PYX_ERR(0, 237,
__pyx_L1_error)
04534 __pyx_t_8 = (__pyx_t_7 != 0);
04535 if (__pyx_t_8) {
04536
04537 /* "PyClical.pyx":238
04538 * for idx in range(self.min(), self.max()+1):
04539 * if idx in self:
04540 * yield idx
04541 * # ««««««««
04542 * def __invert__(self):

```



```

04543 */
04544 __Pyx_INCREF(__pyx_cur_scope->__pyx_v_idx);
04545 __pyx_r = __pyx_cur_scope->__pyx_v_idx;
04546 __Pyx_XGIVEREF(__pyx_t_2);
04547 __pyx_cur_scope->__pyx_t_0 = __pyx_t_2;
04548 __pyx_cur_scope->__pyx_t_1 = __pyx_t_5;
04549 __pyx_cur_scope->__pyx_t_2 = __pyx_t_6;
04550 __Pyx_XGIVEREF(__pyx_r);
04551 __Pyx_RefNannyFinishContext();
04552 __Pyx_Coroutine_ResetAndClearException(__pyx_generator);
04553 /* return from generator, yielding value */
04554 __pyx_generator->resume_label = 1;
04555 return __pyx_r;
04556 __pyx_L7_resume_from_yield:;
04557 __pyx_t_2 = __pyx_cur_scope->__pyx_t_0;
04558 __pyx_cur_scope->__pyx_t_0 = 0;
04559 __Pyx_XGOTREF(__pyx_t_2);
04560 __pyx_t_5 = __pyx_cur_scope->__pyx_t_1;
04561 __pyx_t_6 = __pyx_cur_scope->__pyx_t_2;
04562 if (unlikely(!__pyx_sent_value)) __PYX_ERR(0, 238, __pyx_L1_error)
04563
04564 /* "PyClical.pyx":237
04565 *
04566 * for idx in range(self.min(), self.max()+1):
04567 * if idx in self: # ««««««««
04568 * yield idx
04569 *
04570 */
04571 }
04572
04573 /* "PyClical.pyx":236
04574 * -3,4,7,
04575 * """
04576 * for idx in range(self.min(), self.max()+1): # ««««««««
04577 * if idx in self:
04578 * yield idx
04579 */
04580 }
04581 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
04582 CYTHON_MAYBE_UNUSED_VAR(__pyx_cur_scope);
04583
04584 /* "PyClical.pyx":229
04585 * return self.instance.getitem(idx)
04586 *
04587 * def __iter__(self): # ««««««««
04588 * """
04589 * Iterate over the indices of an index_set.
04590 */
04591
04592 /* function exit code */
04593 PyErr_SetNone(PyExc_StopIteration);
04594 goto __pyx_L0;
04595 __pyx_L1_error:;
04596 __Pyx_XDECREF(__pyx_t_1);
04597 __Pyx_XDECREF(__pyx_t_2);
04598 __Pyx_XDECREF(__pyx_t_3);
04599 __Pyx_XDECREF(__pyx_t_4);
04600 __Pyx_AddTraceback("__iter__", __pyx_clineno, __pyx_lineno, __pyx_filename);
04601 __pyx_L0:;
04602 __Pyx_XDECREF(__pyx_r); __pyx_r = 0;
04603 #if !CYTHON_USE_EXC_INFO_STACK
04604 __Pyx_Coroutine_ResetAndClearException(__pyx_generator);
04605 #endif
04606 __pyx_generator->resume_label = -1;
04607 __Pyx_Coroutine_clear((PyObject*)__pyx_generator);
04608 __Pyx_RefNannyFinishContext();
04609 return __pyx_r;
04610 }
04611
04612 /* "PyClical.pyx":240
04613 * yield idx
04614 *
04615 * def __invert__(self): # ««««««««
04616 * """
04617 * Set complement: not.
04618 */
04619
04620 /* Python wrapper */
04621 static PyObject *__pyx_pw_8PyClical_9index_set_18__invert__(PyObject *__pyx_v_self); /*proto*/
04622 static char __pyx_doc_8PyClical_9index_set_17__invert__[] = "\n Set complement: not.\n\n >>>
 print(~index_set({-16,-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}))\n
 {-32,-31,-30,-29,-28,-27,-26,-25,-24,-23,-22,-21,-20,-19,-18,-17,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32})\n
 ";
04623 #if CYTHON_COMPILING_IN_CPYTHON
04624 struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_17__invert__;
04625 #endif

```

```

04626 static PyObject *__pyx_pw_8PyClical_9index_set_18__invert__(PyObject *__pyx_v_self) {
04627 PyObject *__pyx_r = 0;
04628 __Pyx_RefNannyDeclarations
04629 __Pyx_RefNannySetupContext("__invert__ (wrapper)", 0);
04630 __pyx_r = __pyx_pf_8PyClical_9index_set_17__invert__((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self));
04631
04632 /* function exit code */
04633 __Pyx_RefNannyFinishContext();
04634 return __pyx_r;
04635 }
04636
04637 static PyObject *__pyx_pf_8PyClical_9index_set_17__invert__(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self) {
04638 PyObject *__pyx_r = NULL;
04639 __Pyx_RefNannyDeclarations
04640 PyObject *__pyx_t_1 = NULL;
04641 PyObject *__pyx_t_2 = NULL;
04642 int __pyx_lineno = 0;
04643 const char *__pyx_filename = NULL;
04644 int __pyx_clineno = 0;
04645 __Pyx_RefNannySetupContext("__invert__", 0);
04646
04647 /* "PyClical.pyx":247
04648 *
04649 * """
04650 * return index_set().wrap(self.instance.invert()) # ««««««««
04651 *
04652 * def __xor__(lhs, rhs):
04653 */
04654 __Pyx_XDECREF(__pyx_r);
04655 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_index_set)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 247, __pyx_L1_error)
04656 __Pyx_GOTREF(__pyx_t_1);
04657 __pyx_t_2 = __pyx_f_8PyClical_9index_set_wrap(((struct __pyx_obj_8PyClical_index_set *)__pyx_t_1),
__pyx_v_self->instance->operator~()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 247, __pyx_L1_error)
04658 __Pyx_GOTREF(__pyx_t_2);
04659 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
04660 __pyx_r = __pyx_t_2;
04661 __pyx_t_2 = 0;
04662 goto __pyx_L0;
04663
04664 /* "PyClical.pyx":240
04665 * yield idx
04666 *
04667 * def __invert__(self): # ««««««««
04668 * """
04669 * Set complement: not.
04670 */
04671
04672 /* function exit code */
04673 __pyx_L1_error:;
04674 __Pyx_XDECREF(__pyx_t_1);
04675 __Pyx_XDECREF(__pyx_t_2);
04676 __Pyx_AddTraceback("PyClical.index_set.__invert__", __pyx_clineno, __pyx_lineno, __pyx_filename);
04677 __pyx_r = NULL;
04678 __pyx_L0:;
04679 __Pyx_XGIVEREF(__pyx_r);
04680 __Pyx_RefNannyFinishContext();
04681 return __pyx_r;
04682 }
04683
04684 /* "PyClical.pyx":249
04685 * return index_set().wrap(self.instance.invert())
04686 *
04687 * def __xor__(lhs, rhs): # ««««««««
04688 * """
04689 * Symmetric set difference: exclusive or.
04690 */
04691
04692 /* Python wrapper */
04693 static PyObject *__pyx_pw_8PyClical_9index_set_20__xor__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs); /*proto*/
04694 static char __pyx_doc_8PyClical_9index_set_19__xor__[] = "\n Symmetric set difference:
exclusive or.\n\n >> print(index_set({1}) ^ index_set({2}))\n {1,2}\n >>
print(index_set({1,2}) ^ index_set({2}))\n {1}\n ";
04695 #if CYTHON_COMPILING_IN_CPYTHON
04696 struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_19__xor__;
04697 #endif
04698 static PyObject *__pyx_pw_8PyClical_9index_set_20__xor__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs)
{
04699 PyObject *__pyx_r = 0;
04700 __Pyx_RefNannyDeclarations
04701 __Pyx_RefNannySetupContext("__xor__ (wrapper)", 0);
04702 __pyx_r = __pyx_pf_8PyClical_9index_set_19__xor__(((PyObject *)__pyx_v_lhs), ((PyObject
*)__pyx_v_rhs));

```

```

04703
04704 /* function exit code */
04705 __Pyx_RefNannyFinishContext();
04706 return __pyx_r;
04707 }
04708
04709 static PyObject *__pyx_pf_8PyClical_9index_set_19__xor__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs)
04710 {
04711 PyObject *__pyx_r = NULL;
04712 __Pyx_RefNannyDeclarations
04713 PyObject *__pyx_t_1 = NULL;
04714 PyObject *__pyx_t_2 = NULL;
04715 int __pyx_lineno = 0;
04716 const char *__pyx_filename = NULL;
04717 int __pyx_clineno = 0;
04718 __Pyx_RefNannySetupContext("__xor__", 0);
04719
04720 /* "PyClical.pyx":258
04721 * {1}
04722 * return index_set().wrap(toIndexSet(lhs) ^ toIndexSet(rhs)) # ««««««««
04723 *
04724 * def __ixor__(self, rhs):
04725 */
04726 __Pyx_XDECREF(__pyx_r);
04727 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_index_set)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 258, __pyx_L1_error)
04728 __Pyx_GOTREF(__pyx_t_1);
04729 __pyx_t_2 = __pyx_f_8PyClical_9index_set_wrap(((struct __pyx_obj_8PyClical_index_set *)__pyx_t_1),
(__pyx_f_8PyClical_toIndexSet(__pyx_v_lhs) ^ __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if
(unlikely(!__pyx_t_2)) __PYX_ERR(0, 258, __pyx_L1_error)
04730 __Pyx_GOTREF(__pyx_t_2);
04731 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
04732 __pyx_r = __pyx_t_2;
04733 __pyx_t_2 = 0;
04734 goto __pyx_L0;
04735
04736 /* "PyClical.pyx":249
04737 * return index_set().wrap(self.instance.invert())
04738 *
04739 * def __xor__(lhs, rhs): # ««««««««
04740 * """
04741 * Symmetric set difference: exclusive or.
04742 */
04743
04744 /* function exit code */
04745 __pyx_L1_error:;
04746 __Pyx_XDECREF(__pyx_t_1);
04747 __Pyx_XDECREF(__pyx_t_2);
04748 __Pyx_AddTraceback("PyClical.index_set.__xor__", __pyx_clineno, __pyx_lineno, __pyx_filename);
04749 __pyx_r = NULL;
04750 __pyx_L0:;
04751 __Pyx_XGIVEREF(__pyx_r);
04752 __Pyx_RefNannyFinishContext();
04753 return __pyx_r;
04754 }
04755
04756 /* "PyClical.pyx":260
04757 * return index_set().wrap(toIndexSet(lhs) ^ toIndexSet(rhs))
04758 *
04759 * def __ixor__(self, rhs): # ««««««««
04760 * """
04761 * Symmetric set difference: exclusive or.
04762 */
04763
04764 /* Python wrapper */
04765 static PyObject *__pyx_pw_8PyClical_9index_set_22__ixor__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
04766 static char __pyx_doc_8PyClical_9index_set_21__ixor__[] = "\n Symmetric set difference:
exclusive or.\n\n >> x = index_set({1}); x ^= index_set({2}); print(x)\n {1,2}\n
>> x = index_set({1,2}); x ^= index_set({2}); print(x)\n {1}\n ";
04767 #if CYTHON_COMPILING_IN_CPYTHON
04768 struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_21__ixor__;
04769 #endif
04770 static PyObject *__pyx_pw_8PyClical_9index_set_22__ixor__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
04771 PyObject *__pyx_r = 0;
04772 __Pyx_RefNannyDeclarations
04773 __Pyx_RefNannySetupContext("__ixor__ (wrapper)", 0);
04774 __pyx_r = __pyx_pf_8PyClical_9index_set_21__ixor__(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));
04775
04776 /* function exit code */
04777 __Pyx_RefNannyFinishContext();
04778 return __pyx_r;
04779 }
04780

```

```

04781 static PyObject *__pyx_pf_8PyClical_9index_set_21_ixor__(struct __pyx_obj_8PyClical_index_set
 *__pyx_v_self, PyObject *__pyx_v_rhs) {
04782 PyObject *__pyx_r = NULL;
04783 __Pyx_RefNannyDeclarations
04784 PyObject *__pyx_t_1 = NULL;
04785 int __pyx_lineno = 0;
04786 const char *__pyx_filename = NULL;
04787 int __pyx_clineno = 0;
04788 __Pyx_RefNannySetupContext("__ixor__", 0);
04789
04790 /* "PyClical.pyx":269
04791 * {1}
04792 * """
04793 * return self.wrap(self.unwrap() ^ toIndexSet(rhs)) # ««««««««
04794 *
04795 * def __and__(lhs, rhs):
04796 */
04797 __Pyx_XDECREF(__pyx_r);
04798 __pyx_t_1 = __pyx_f_8PyClical_9index_set_wrap(__pyx_v_self,
 (__pyx_f_8PyClical_9index_set_unwrap(__pyx_v_self) ^ __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if
 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 269, __pyx_L1_error)
04799 __Pyx_GOTREF(__pyx_t_1);
04800 __pyx_r = __pyx_t_1;
04801 __pyx_t_1 = 0;
04802 goto __pyx_L0;
04803
04804 /* "PyClical.pyx":260
04805 * return index_set().wrap(toIndexSet(lhs) ^ toIndexSet(rhs))
04806 *
04807 * def __ixor__(self, rhs): # ««««««««
04808 * """
04809 * Symmetric set difference: exclusive or.
04810 */
04811
04812 /* function exit code */
04813 __pyx_L1_error:;
04814 __Pyx_XDECREF(__pyx_t_1);
04815 __Pyx_AddTraceback("PyClical.index_set.__ixor__", __pyx_clineno, __pyx_lineno, __pyx_filename);
04816 __pyx_r = NULL;
04817 __pyx_L0:;
04818 __Pyx_XGIVEREF(__pyx_r);
04819 __Pyx_RefNannyFinishContext();
04820 return __pyx_r;
04821 }
04822
04823 /* "PyClical.pyx":271
04824 * return self.wrap(self.unwrap() ^ toIndexSet(rhs))
04825 *
04826 * def __and__(lhs, rhs): # ««««««««
04827 * """
04828 * Set intersection: and.
04829 */
04830
04831 /* Python wrapper */
04832 static PyObject *__pyx_pw_8PyClical_9index_set_24_and__(PyObject *__pyx_v_lhs, PyObject
 *__pyx_v_rhs); /*proto*/
04833 static char __pyx_doc_8PyClical_9index_set_23_and__[] = "\n Set intersection: and.\n\n
 >> print(index_set({1}) & index_set({2}))\n {}\n >> print(index_set({1,2}) &
 index_set({2}))\n {2}\n ";
04834 #if CYTHON_COMPILING_IN_CPYTHON
04835 struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_23_and__;
04836 #endif
04837 static PyObject *__pyx_pw_8PyClical_9index_set_24_and__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs)
 {
04838 PyObject *__pyx_r = 0;
04839 __Pyx_RefNannyDeclarations
04840 __Pyx_RefNannySetupContext("__and__ (wrapper)", 0);
04841 __pyx_r = __pyx_pf_8PyClical_9index_set_23_and__(((PyObject *)__pyx_v_lhs), ((PyObject
 *)__pyx_v_rhs));
04842
04843 /* function exit code */
04844 __Pyx_RefNannyFinishContext();
04845 return __pyx_r;
04846 }
04847
04848 static PyObject *__pyx_pf_8PyClical_9index_set_23_and__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs)
 {
04849 PyObject *__pyx_r = NULL;
04850 __Pyx_RefNannyDeclarations
04851 PyObject *__pyx_t_1 = NULL;
04852 PyObject *__pyx_t_2 = NULL;
04853 int __pyx_lineno = 0;
04854 const char *__pyx_filename = NULL;
04855 int __pyx_clineno = 0;
04856 __Pyx_RefNannySetupContext("__and__", 0);
04857
04858 /* "PyClical.pyx":280

```

```

04859 * {2}
04860 * """
04861 * return index_set().wrap(toIndexSet(lhs) & toIndexSet(rhs)) # ««««««««
04862 *
04863 * def __iand__(self, rhs):
04864 */
04865 __Pyx_XDECREF(__pyx_r);
04866 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_index_set)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 280, __pyx_L1_error)
04867 __Pyx_GOTREF(__pyx_t_1);
04868 __pyx_t_2 = __pyx_f_8PyClical_9index_set_wrap(((struct __pyx_obj_8PyClical_index_set *)__pyx_t_1),
(__pyx_f_8PyClical_toIndexSet(__pyx_v_lhs) & __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if
(unlikely(!__pyx_t_2)) __PYX_ERR(0, 280, __pyx_L1_error)
04869 __Pyx_GOTREF(__pyx_t_2);
04870 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
04871 __pyx_r = __pyx_t_2;
04872 __pyx_t_2 = 0;
04873 goto __pyx_L0;
04874
04875 /* "PyClical.pyx":271
04876 * return self.wrap(self.unwrap() ^ toIndexSet(rhs))
04877 *
04878 * def __and__(lhs, rhs): # ««««««««
04879 * """
04880 * Set intersection: and.
04881 */
04882
04883 /* function exit code */
04884 __pyx_L1_error:;
04885 __Pyx_XDECREF(__pyx_t_1);
04886 __Pyx_XDECREF(__pyx_t_2);
04887 __Pyx_AddTraceback("PyClical.index_set.__and__", __pyx_clineno, __pyx_lineno, __pyx_filename);
04888 __pyx_r = NULL;
04889 __pyx_L0:;
04890 __Pyx_XGIVEREF(__pyx_r);
04891 __Pyx_RefNannyFinishContext();
04892 return __pyx_r;
04893 }
04894
04895 /* "PyClical.pyx":282
04896 * return index_set().wrap(toIndexSet(lhs) & toIndexSet(rhs))
04897 *
04898 * def __iand__(self, rhs): # ««««««««
04899 * """
04900 * Set intersection: and.
04901 */
04902
04903 /* Python wrapper */
04904 static PyObject *__pyx_pw_8PyClical_9index_set_26__iand__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
04905 static char __pyx_doc_8PyClical_9index_set_25__iand__[] = "\n Set intersection: and.\n\n
>> x = index_set({1}); x &= index_set({2}); print(x)\n {}\n >> x = index_set({1,2}); x
&= index_set({2}); print(x)\n {2}\n ";
04906 #if CYTHON_COMPILING_IN_CPYTHON
04907 struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_25__iand__;
04908 #endif
04909 static PyObject *__pyx_pw_8PyClical_9index_set_26__iand__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
04910 PyObject *__pyx_r = 0;
04911 __Pyx_RefNannyDeclarations
04912 __Pyx_RefNannySetupContext("__iand__ (wrapper)", 0);
04913 __pyx_r = __pyx_pf_8PyClical_9index_set_25__iand__(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));
04914
04915 /* function exit code */
04916 __Pyx_RefNannyFinishContext();
04917 return __pyx_r;
04918 }
04919
04920 static PyObject *__pyx_pf_8PyClical_9index_set_25__iand__(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self, PyObject *__pyx_v_rhs) {
04921 PyObject *__pyx_r = NULL;
04922 __Pyx_RefNannyDeclarations
04923 PyObject *__pyx_t_1 = NULL;
04924 int __pyx_lineno = 0;
04925 const char *__pyx_filename = NULL;
04926 int __pyx_clineno = 0;
04927 __Pyx_RefNannySetupContext("__iand__", 0);
04928
04929 /* "PyClical.pyx":291
04930 * {2}
04931 * """
04932 * return self.wrap(self.unwrap() & toIndexSet(rhs)) # ««««««««
04933 *
04934 * def __or__(lhs, rhs):
04935 */
04936 __Pyx_XDECREF(__pyx_r);

```

```

04937 __pyx_t_1 = __pyx_f_8PyClical_9index_set_wrap(__pyx_v_self,
(__pyx_f_8PyClical_9index_set_unwrap(__pyx_v_self) & __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 291, __pyx_L1_error)
04938 __Pyx_GOTREF(__pyx_t_1);
04939 __pyx_r = __pyx_t_1;
04940 __pyx_t_1 = 0;
04941 goto __pyx_L0;
04942
04943 /* "PyClical.pyx":282
04944 * return index_set().wrap(toIndexSet(lhs) & toIndexSet(rhs))
04945 *
04946 * def __iand__(self, rhs): # ««««««««
04947 * """
04948 * Set intersection: and.
04949 */
04950
04951 /* function exit code */
04952 __pyx_L1_error:;
04953 __Pyx_XDECREF(__pyx_t_1);
04954 __Pyx_AddTraceback("PyClical.index_set.__iand__", __pyx_clineno, __pyx_lineno, __pyx_filename);
04955 __pyx_r = NULL;
04956 __pyx_L0:;
04957 __Pyx_XGIVEREF(__pyx_r);
04958 __Pyx_RefNannyFinishContext();
04959 return __pyx_r;
04960 }
04961
04962 /* "PyClical.pyx":293
04963 * return self.wrap(self.unwrap() & toIndexSet(rhs))
04964 *
04965 * def __or__(lhs, rhs): # ««««««««
04966 * """
04967 * Set union: or.
04968 */
04969
04970 /* Python wrapper */
04971 static PyObject *__pyx_pw_8PyClical_9index_set_28__or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs);
/*proto*/
04972 static char __pyx_doc_8PyClical_9index_set_27__or__[] = "\n Set union: or.\n\n >>
print(index_set({1}) | index_set({2}))\n {1,2}\n >> print(index_set({1,2}) |
index_set({2}))\n {1,2}\n ";
04973 #if CYTHON_COMPILING_IN_CPYTHON
04974 struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_27__or__;
04975 #endif
04976 static PyObject *__pyx_pw_8PyClical_9index_set_28__or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs)
{
04977 PyObject *__pyx_r = 0;
04978 __Pyx_RefNannyDeclarations
04979 __Pyx_RefNannySetupContext("__or__ (wrapper)", 0);
04980 __pyx_r = __pyx_pf_8PyClical_9index_set_27__or__(((PyObject *)__pyx_v_lhs), ((PyObject
*)__pyx_v_rhs));
04981
04982 /* function exit code */
04983 __Pyx_RefNannyFinishContext();
04984 return __pyx_r;
04985 }
04986
04987 static PyObject *__pyx_pf_8PyClical_9index_set_27__or__(PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs)
{
04988 PyObject *__pyx_r = NULL;
04989 __Pyx_RefNannyDeclarations
04990 PyObject *__pyx_t_1 = NULL;
04991 PyObject *__pyx_t_2 = NULL;
04992 int __pyx_lineno = 0;
04993 const char *__pyx_filename = NULL;
04994 int __pyx_clineno = 0;
04995 __Pyx_RefNannySetupContext("__or__", 0);
04996
04997 /* "PyClical.pyx":302
04998 * {1,2}
04999 * """
05000 * return index_set().wrap(toIndexSet(lhs) | toIndexSet(rhs)) # ««««««««
05001 *
05002 * def __ior__(self, rhs):
05003 */
05004 __Pyx_XDECREF(__pyx_r);
05005 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_index_set)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 302, __pyx_L1_error)
05006 __Pyx_GOTREF(__pyx_t_1);
05007 __pyx_t_2 = __pyx_f_8PyClical_9index_set_wrap(((struct __pyx_obj_8PyClical_index_set *)__pyx_t_1),
(__pyx_f_8PyClical_toIndexSet(__pyx_v_lhs) | __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if
(unlikely(!__pyx_t_2)) __PYX_ERR(0, 302, __pyx_L1_error)
05008 __Pyx_GOTREF(__pyx_t_2);
05009 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
05010 __pyx_r = __pyx_t_2;
05011 __pyx_t_2 = 0;
05012 goto __pyx_L0;

```

```

05013
05014 /* "PyClical.pyx":293
05015 * return self.wrap(self.unwrap() & toIndexSet(rhs))
05016 *
05017 * def __or__(lhs, rhs): # ««««««««
05018 * """
05019 * Set union: or.
05020 */
05021
05022 /* function exit code */
05023 __pyx_L1_error:;
05024 __Pyx_XDECREF(__pyx_t_1);
05025 __Pyx_XDECREF(__pyx_t_2);
05026 __Pyx_AddTraceback("PyClical.index_set.__or__", __pyx_clineno, __pyx_lineno, __pyx_filename);
05027 __pyx_r = NULL;
05028 __pyx_L0:;
05029 __Pyx_XGIVEREF(__pyx_r);
05030 __Pyx_RefNannyFinishContext();
05031 return __pyx_r;
05032 }
05033
05034 /* "PyClical.pyx":304
05035 * return index_set().wrap(toIndexSet(lhs) | toIndexSet(rhs))
05036 *
05037 * def __ior__(self, rhs): # ««««««««
05038 * """
05039 * Set union: or.
05040 */
05041
05042 /* Python wrapper */
05043 static PyObject *__pyx_pw_8PyClical_9index_set_30__ior__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
05044 static char __pyx_doc_8PyClical_9index_set_29__ior__[] = "\n Set union: or.\n\n >> x =
index_set({1}); x |= index_set({2}); print(x)\n {1,2}\n >> x = index_set({1,2}); x |=
index_set({2}); print(x)\n {1,2}\n ";
05045 #if CYTHON_COMPILING_IN_CPYTHON
05046 struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_29__ior__;
05047 #endif
05048 static PyObject *__pyx_pw_8PyClical_9index_set_30__ior__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
05049 PyObject *__pyx_r = 0;
05050 __Pyx_RefNannyDeclarations
05051 __Pyx_RefNannySetupContext("__ior__ (wrapper)", 0);
05052 __pyx_r = __pyx_pf_8PyClical_9index_set_29__ior__(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));
05053
05054 /* function exit code */
05055 __Pyx_RefNannyFinishContext();
05056 return __pyx_r;
05057 }
05058
05059 static PyObject *__pyx_pf_8PyClical_9index_set_29__ior__(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self, PyObject *__pyx_v_rhs) {
05060 PyObject *__pyx_r = NULL;
05061 __Pyx_RefNannyDeclarations
05062 PyObject *__pyx_t_1 = NULL;
05063 int __pyx_lineno = 0;
05064 const char *__pyx_filename = NULL;
05065 int __pyx_clineno = 0;
05066 __Pyx_RefNannySetupContext("__ior__", 0);
05067
05068 /* "PyClical.pyx":313
05069 * {1,2}
05070 * """
05071 * return self.wrap(self.unwrap() | toIndexSet(rhs)) # ««««««««
05072 *
05073 * def count(self):
05074 */
05075 __Pyx_XDECREF(__pyx_r);
05076 __pyx_t_1 = __pyx_f_8PyClical_9index_set_wrap(__pyx_v_self,
(__pyx_f_8PyClical_9index_set_unwrap(__pyx_v_self) | __pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 313, __pyx_L1_error)
05077 __Pyx_GOTREF(__pyx_t_1);
05078 __pyx_r = __pyx_t_1;
05079 __pyx_t_1 = 0;
05080 goto __pyx_L0;
05081
05082 /* "PyClical.pyx":304
05083 * return index_set().wrap(toIndexSet(lhs) | toIndexSet(rhs))
05084 *
05085 * def __ior__(self, rhs): # ««««««««
05086 * """
05087 * Set union: or.
05088 */
05089
05090 /* function exit code */
05091 __pyx_L1_error:;

```

```

05092 __Pyx_XDECREF(__pyx_t_1);
05093 __Pyx_AddTraceback("PyClical.index_set.__ior__", __pyx_clineno, __pyx_lineno, __pyx_filename);
05094 __pyx_r = NULL;
05095 __pyx_L0:;
05096 __Pyx_XGIVEREF(__pyx_r);
05097 __Pyx_RefNannyFinishContext();
05098 return __pyx_r;
05099 }
05100
05101 /* "PyClical.pyx":315
05102 * return self.wrap(self.unwrap() | toIndexSet(rhs))
05103 *
05104 * def count(self):
05105 * """
05106 * Cardinality: Number of indices included in set.
05107 */
05108
05109 /* Python wrapper */
05110 static PyObject *__pyx_pw_8PyClical_9index_set_32count(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject
05111 *unused); /*proto*/
05112 static char __pyx_doc_8PyClical_9index_set_31count[] = "\n Cardinality: Number of indices\n included in set.\n\n >> index_set({-1,1,2}).count()\n 3\n ";
05113 static PyObject *__pyx_pw_8PyClical_9index_set_32count(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject
05114 *unused) {
05115 PyObject *__pyx_r = 0;
05116 __Pyx_RefNannyDeclarations
05117 __Pyx_RefNannySetupContext("count (wrapper)", 0);
05118 __pyx_r = __pyx_pf_8PyClical_9index_set_31count(((struct __pyx_obj_8PyClical_index_set
05119 *)__pyx_v_self));
05120
05121 /* function exit code */
05122 __Pyx_RefNannyFinishContext();
05123 return __pyx_r;
05124 }
05125
05126 static PyObject *__pyx_pf_8PyClical_9index_set_31count(struct __pyx_obj_8PyClical_index_set
05127 *__pyx_v_self) {
05128 PyObject *__pyx_r = NULL;
05129 __Pyx_RefNannyDeclarations
05130 PyObject *__pyx_t_1 = NULL;
05131 int __pyx_lineno = 0;
05132 const char *__pyx_filename = NULL;
05133 int __pyx_clineno = 0;
05134 __Pyx_RefNannySetupContext("count", 0);
05135
05136 /* "PyClical.pyx":322
05137 * 3
05138 * """
05139 * return self.instance.count()
05140 * # ««««««««
05141 *
05142 * def count_neg(self):
05143 * """
05144 * Cardinality: Number of indices included in set.
05145 */
05146
05147 __Pyx_XDECREF(__pyx_r);
05148 __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->count()); if (unlikely(!__pyx_t_1))
05149 __PYX_ERR(0, 322, __pyx_L1_error)
05150 __Pyx_GOTREF(__pyx_t_1);
05151 __pyx_r = __pyx_t_1;
05152 __pyx_t_1 = 0;
05153 goto __pyx_L0;
05154
05155 /* "PyClical.pyx":315
05156 * return self.wrap(self.unwrap() | toIndexSet(rhs))
05157 *
05158 * def count(self):
05159 * """
05160 * Cardinality: Number of indices included in set.
05161 */
05162
05163 /* function exit code */
05164 __pyx_L1_error:;
05165 __Pyx_XDECREF(__pyx_t_1);
05166 __Pyx_AddTraceback("PyClical.index_set.count", __pyx_clineno, __pyx_lineno, __pyx_filename);
05167 __pyx_r = NULL;
05168 __pyx_L0:;
05169 __Pyx_XGIVEREF(__pyx_r);
05170 __Pyx_RefNannyFinishContext();
05171 return __pyx_r;
05172 }
05173
05174 /* "PyClical.pyx":324
05175 * return self.instance.count()
05176 *
05177 * def count_neg(self):
05178 * """
05179 * Number of negative indices included in set.
05180 */
05181

```



```

05173 /* Python wrapper */
05174 static PyObject *__pyx_pw_8PyClical_9index_set_34count_neg(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
05175 static char __pyx_doc_8PyClical_9index_set_33count_neg[] = "\n Number of negative indices
included in set.\n\n >> index_set({-1,1,2}).count_neg()\n 1\n ";
05176 static PyObject *__pyx_pw_8PyClical_9index_set_34count_neg(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
05177 PyObject *__pyx_r = 0;
05178 __Pyx_RefNannyDeclarations
05179 __Pyx_RefNannySetupContext("count_neg (wrapper)", 0);
05180 __pyx_r = __pyx_pf_8PyClical_9index_set_33count_neg(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self));
05181
05182 /* function exit code */
05183 __Pyx_RefNannyFinishContext();
05184 return __pyx_r;
05185 }
05186
05187 static PyObject *__pyx_pf_8PyClical_9index_set_33count_neg(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self) {
05188 PyObject *__pyx_r = NULL;
05189 __Pyx_RefNannyDeclarations
05190 PyObject *__pyx_t_1 = NULL;
05191 int __pyx_lineno = 0;
05192 const char *__pyx_filename = NULL;
05193 int __pyx_clineno = 0;
05194 __Pyx_RefNannySetupContext("count_neg", 0);
05195
05196 /* "PyClical.pyx":331
05197 * 1
05198 * """
05199 * return self.instance.count_neg() # ««««««««
05200 *
05201 * def count_pos(self):
05202 */
05203 __Pyx_XDECREF(__pyx_r);
05204 __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->count_neg()); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 331, __pyx_L1_error)
05205 __Pyx_GOTREF(__pyx_t_1);
05206 __pyx_r = __pyx_t_1;
05207 __pyx_t_1 = 0;
05208 goto __pyx_L0;
05209
05210 /* "PyClical.pyx":324
05211 * return self.instance.count()
05212 *
05213 * def count_neg(self): # ««««««««
05214 * """
05215 * Number of negative indices included in set.
05216 */
05217
05218 /* function exit code */
05219 __pyx_L1_error:;
05220 __Pyx_XDECREF(__pyx_t_1);
05221 __Pyx_AddTraceback("PyClical.index_set.count_neg", __pyx_clineno, __pyx_lineno, __pyx_filename);
05222 __pyx_r = NULL;
05223 __pyx_L0:;
05224 __Pyx_XGIVEREF(__pyx_r);
05225 __Pyx_RefNannyFinishContext();
05226 return __pyx_r;
05227 }
05228
05229 /* "PyClical.pyx":333
05230 * return self.instance.count_neg()
05231 *
05232 * def count_pos(self): # ««««««««
05233 * """
05234 * Number of positive indices included in set.
05235 */
05236
05237 /* Python wrapper */
05238 static PyObject *__pyx_pw_8PyClical_9index_set_36count_pos(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
05239 static char __pyx_doc_8PyClical_9index_set_35count_pos[] = "\n Number of positive indices
included in set.\n\n >> index_set({-1,1,2}).count_pos()\n 2\n ";
05240 static PyObject *__pyx_pw_8PyClical_9index_set_36count_pos(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
05241 PyObject *__pyx_r = 0;
05242 __Pyx_RefNannyDeclarations
05243 __Pyx_RefNannySetupContext("count_pos (wrapper)", 0);
05244 __pyx_r = __pyx_pf_8PyClical_9index_set_35count_pos(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self));
05245
05246 /* function exit code */
05247 __Pyx_RefNannyFinishContext();
05248 return __pyx_r;
05249 }

```

```

05250
05251 static PyObject *__pyx_pf_8PyClical_9index_set_35count_pos(struct __pyx_obj_8PyClical_index_set
05252 *__pyx_v_self) {
05253 PyObject *__pyx_r = NULL;
05254 __Pyx_RefNannyDeclarations
05255 PyObject *__pyx_t_1 = NULL;
05256 int __pyx_lineno = 0;
05257 const char *__pyx_filename = NULL;
05258 int __pyx_clineno = 0;
05259 __Pyx_RefNannySetupContext("count_pos", 0);
05260
05261 /* "PyClical.pyx":340
05262 * 2
05263 * """
05264 * return self.instance.count_pos() # ««««««««
05265 *
05266 * def min(self):
05267 */
05268 __Pyx_XDECREF(__pyx_r);
05269 __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->count_pos()); if (unlikely(!__pyx_t_1))
05270 __PYX_ERR(0, 340, __pyx_L1_error)
05271 __Pyx_GOTREF(__pyx_t_1);
05272 __pyx_r = __pyx_t_1;
05273 __pyx_t_1 = 0;
05274 goto __pyx_L0;
05275
05276 /* "PyClical.pyx":333
05277 * return self.instance.count_neg()
05278 *
05279 * def count_pos(self): # ««««««««
05280 * """
05281 * Number of positive indices included in set.
05282 */
05283 /* function exit code */
05284 __pyx_L1_error;
05285 __Pyx_XDECREF(__pyx_t_1);
05286 __Pyx_AddTraceback("PyClical.index_set.count_pos", __pyx_clineno, __pyx_lineno, __pyx_filename);
05287 __pyx_r = NULL;
05288 __pyx_L0;
05289 __Pyx_XGIVEREF(__pyx_r);
05290 __Pyx_RefNannyFinishContext();
05291 return __pyx_r;
05292 }
05293
05294 /* "PyClical.pyx":342
05295 * return self.instance.count_pos()
05296 *
05297 * def min(self): # ««««««««
05298 * """
05299 * Minimum member.
05300 */
05301
05302 /* Python wrapper */
05303 static PyObject *__pyx_pw_8PyClical_9index_set_38min(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject
05304 *unused); /*proto*/
05305 static char __pyx_doc_8PyClical_9index_set_37min[] = "\n Minimum member.\n\n >>
05306 index_set({-1,1,2}).min()\n -1\n ";
05307 static PyObject *__pyx_pw_8PyClical_9index_set_38min(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject
05308 *unused) {
05309 PyObject *__pyx_r = 0;
05310 __Pyx_RefNannyDeclarations
05311 __Pyx_RefNannySetupContext("min (wrapper)", 0);
05312 __pyx_r = __pyx_pf_8PyClical_9index_set_37min(((struct __pyx_obj_8PyClical_index_set
05313 *)__pyx_v_self));
05314
05315 /* function exit code */
05316 __Pyx_RefNannyFinishContext();
05317 return __pyx_r;
05318 }
05319
05320 static PyObject *__pyx_pf_8PyClical_9index_set_37min(struct __pyx_obj_8PyClical_index_set
05321 *__pyx_v_self) {
05322 PyObject *__pyx_r = NULL;
05323 __Pyx_RefNannyDeclarations
05324 PyObject *__pyx_t_1 = NULL;
05325 int __pyx_lineno = 0;
05326 const char *__pyx_filename = NULL;
05327 int __pyx_clineno = 0;
05328 __Pyx_RefNannySetupContext("min", 0);
05329
05330 /* "PyClical.pyx":349
05331 * -1
05332 * """
05333 * return self.instance.min() # ««««««««
05334 *
05335 * def max(self):

```

```

05330 */
05331 __Pyx_XDECREF(__pyx_r);
05332 __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->min()); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 349, __pyx_L1_error)
05333 __Pyx_GOTREF(__pyx_t_1);
05334 __pyx_r = __pyx_t_1;
05335 __pyx_t_1 = 0;
05336 goto __pyx_L0;
05337
05338 /* "PyClical.pyx":342
05339 * return self.instance.count_pos()
05340 *
05341 * def min(self): # ««««««««
05342 * """
05343 * Minimum member.
05344 */
05345
05346 /* function exit code */
05347 __pyx_L1_error:;
05348 __Pyx_XDECREF(__pyx_t_1);
05349 __Pyx_AddTraceback("PyClical.index_set.min", __pyx_clineno, __pyx_lineno, __pyx_filename);
05350 __pyx_r = NULL;
05351 __pyx_L0:;
05352 __Pyx_XGIVEREF(__pyx_r);
05353 __Pyx_RefNannyFinishContext();
05354 return __pyx_r;
05355 }
05356
05357 /* "PyClical.pyx":351
05358 * return self.instance.min()
05359 *
05360 * def max(self): # ««««««««
05361 * """
05362 * Maximum member.
05363 */
05364
05365 /* Python wrapper */
05366 static PyObject *__pyx_pw_8PyClical_9index_set_40max(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject
__unused); /*proto*/
05367 static char __pyx_doc_8PyClical_9index_set_39max[] = "\n Maximum member.\n\n >>
index_set({-1,1,2}).max()\n 2\n ";
05368 static PyObject *__pyx_pw_8PyClical_9index_set_40max(PyObject *__pyx_v_self, CYTHON_UNUSED PyObject
__unused) {
05369 PyObject *__pyx_r = 0;
05370 __Pyx_RefNannyDeclarations
05371 __Pyx_RefNannySetupContext("max (wrapper)", 0);
05372 __pyx_r = __pyx_pf_8PyClical_9index_set_39max(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self));
05373
05374 /* function exit code */
05375 __Pyx_RefNannyFinishContext();
05376 return __pyx_r;
05377 }
05378
05379 static PyObject *__pyx_pf_8PyClical_9index_set_39max(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self) {
05380 PyObject *__pyx_r = NULL;
05381 __Pyx_RefNannyDeclarations
05382 PyObject *__pyx_t_1 = NULL;
05383 int __pyx_lineno = 0;
05384 const char *__pyx_filename = NULL;
05385 int __pyx_clineno = 0;
05386 __Pyx_RefNannySetupContext("max", 0);
05387
05388 /* "PyClical.pyx":358
05389 * 2
05390 * """
05391 * return self.instance.max() # ««««««««
05392 *
05393 * def hash_fn(self):
05394 */
05395 __Pyx_XDECREF(__pyx_r);
05396 __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->max()); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 358, __pyx_L1_error)
05397 __Pyx_GOTREF(__pyx_t_1);
05398 __pyx_r = __pyx_t_1;
05399 __pyx_t_1 = 0;
05400 goto __pyx_L0;
05401
05402 /* "PyClical.pyx":351
05403 * return self.instance.min()
05404 *
05405 * def max(self): # ««««««««
05406 * """
05407 * Maximum member.
05408 */
05409

```

```

05410 /* function exit code */
05411 __pyx_L1_error:;
05412 __Pyx_XDECREF(__pyx_t_1);
05413 __Pyx_AddTraceback("PyCliclal.index_set.max", __pyx_clineno, __pyx_lineno, __pyx_filename);
05414 __pyx_r = NULL;
05415 __pyx_L0:;
05416 __Pyx_XGIVEREF(__pyx_r);
05417 __Pyx_RefNannyFinishContext();
05418 return __pyx_r;
05419 }
05420
05421 /* "PyCliclal.pyx":360
05422 * return self.instance.max()
05423 *
05424 * def hash_fn(self): # ««««««««
05425 * """
05426 * Hash function.
05427 */
05428
05429 /* Python wrapper */
05430 static PyObject *__pyx_pw_8PyCliclal_9index_set_42hash_fn(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
05431 static char __pyx_doc_8PyCliclal_9index_set_41hash_fn[] = "\n Hash function.\n ";
05432 static PyObject *__pyx_pw_8PyCliclal_9index_set_42hash_fn(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
05433 PyObject *__pyx_r = 0;
05434 __Pyx_RefNannyDeclarations
05435 __Pyx_RefNannySetupContext("hash_fn (wrapper)", 0);
05436 __pyx_r = __pyx_pf_8PyCliclal_9index_set_41hash_fn(((struct __pyx_obj_8PyCliclal_index_set
*)__pyx_v_self));
05437
05438 /* function exit code */
05439 __Pyx_RefNannyFinishContext();
05440 return __pyx_r;
05441 }
05442
05443 static PyObject *__pyx_pf_8PyCliclal_9index_set_41hash_fn(struct __pyx_obj_8PyCliclal_index_set
*__pyx_v_self) {
05444 PyObject *__pyx_r = NULL;
05445 __Pyx_RefNannyDeclarations
05446 PyObject *__pyx_t_1 = NULL;
05447 int __pyx_lineno = 0;
05448 const char *__pyx_filename = NULL;
05449 int __pyx_clineno = 0;
05450 __Pyx_RefNannySetupContext("hash_fn", 0);
05451
05452 /* "PyCliclal.pyx":364
05453 * Hash function.
05454 *
05455 * return self.instance.hash_fn() # ««««««««
05456 *
05457 * def sign_of_mult(self, rhs):
05458 */
05459 __Pyx_XDECREF(__pyx_r);
05460 __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->hash_fn()); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 364, __pyx_L1_error)
05461 __Pyx_GOTREF(__pyx_t_1);
05462 __pyx_r = __pyx_t_1;
05463 __pyx_t_1 = 0;
05464 goto __pyx_L0;
05465
05466 /* "PyCliclal.pyx":360
05467 * return self.instance.max()
05468 *
05469 * def hash_fn(self): # ««««««««
05470 * """
05471 * Hash function.
05472 */
05473
05474 /* function exit code */
05475 __pyx_L1_error:;
05476 __Pyx_XDECREF(__pyx_t_1);
05477 __Pyx_AddTraceback("PyCliclal.index_set.hash_fn", __pyx_clineno, __pyx_lineno, __pyx_filename);
05478 __pyx_r = NULL;
05479 __pyx_L0:;
05480 __Pyx_XGIVEREF(__pyx_r);
05481 __Pyx_RefNannyFinishContext();
05482 return __pyx_r;
05483 }
05484
05485 /* "PyCliclal.pyx":366
05486 * return self.instance.hash_fn()
05487 *
05488 * def sign_of_mult(self, rhs): # ««««««««
05489 * """
05490 * Sign of geometric product of two Clifford basis elements.
05491 */

```

```

05492
05493 /* Python wrapper */
05494 static PyObject *__pyx_pw_8PyClical_9index_set_44sign_of_mult(PyObject *__pyx_v_self, PyObject
05495 *__pyx_v_rhs); /*proto*/
05496 static char __pyx_doc_8PyClical_9index_set_43sign_of_mult[] = "\n Sign of geometric product of
05497 two Clifford basis elements.\n\n >> s = index_set({1,2}); t=index_set({-1});
05498 s.sign_of_mult(t)\n 1\n ";
05499 static PyObject *__pyx_pw_8PyClical_9index_set_44sign_of_mult(PyObject *__pyx_v_self, PyObject
05500 *__pyx_v_rhs) {
05501 PyObject *__pyx_r = 0;
05502 __Pyx_RefNannyDeclarations
05503 __Pyx_RefNannySetupContext("sign_of_mult (wrapper)", 0);
05504 __pyx_r = __pyx_pf_8PyClical_9index_set_43sign_of_mult(((struct __pyx_obj_8PyClical_index_set
05505 *)__pyx_v_self), ((PyObject *)__pyx_v_rhs));
05506
05507 /* function exit code */
05508 __Pyx_RefNannyFinishContext();
05509 return __pyx_r;
05510 }
05511
05512 static PyObject *__pyx_pf_8PyClical_9index_set_43sign_of_mult(struct __pyx_obj_8PyClical_index_set
05513 *__pyx_v_self, PyObject *__pyx_v_rhs) {
05514 PyObject *__pyx_r = NULL;
05515 __Pyx_RefNannyDeclarations
05516 PyObject *__pyx_t_1 = NULL;
05517 int __pyx_lineno = 0;
05518 const char *__pyx_filename = NULL;
05519 int __pyx_clineno = 0;
05520 __Pyx_RefNannySetupContext("sign_of_mult", 0);
05521
05522 /* "PyClical.pyx":373
05523 * 1
05524 * """
05525 * return self.instance.sign_of_mult(toIndexSet(rhs)) # ««««««««
05526 *
05527 * def sign_of_square(self):
05528 */
05529 __Pyx_XDECREF(__pyx_r);
05530 __pyx_t_1 =
05531 __Pyx_PyInt_From_int(__pyx_v_self->instance->sign_of_mult(__pyx_f_8PyClical_toIndexSet(__pyx_v_rhs)));
05532 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 373, __pyx_L1_error)
05533 __Pyx_GOTREF(__pyx_t_1);
05534 __pyx_r = __pyx_t_1;
05535 __pyx_t_1 = 0;
05536 goto __pyx_L0;
05537
05538 /* "PyClical.pyx":366
05539 * return self.instance.hash_fn()
05540 *
05541 * def sign_of_mult(self, rhs): # ««««««««
05542 * """
05543 * Sign of geometric product of two Clifford basis elements.
05544 */
05545 __Pyx_XDECREF(__pyx_r);
05546 __Pyx_GOTREF(__pyx_t_1);
05547 __Pyx_ADDTRACEBACK("PyClical.index_set.sign_of_mult", __pyx_clineno, __pyx_lineno, __pyx_filename);
05548 __pyx_r = NULL;
05549 __pyx_L0:;
05550 __Pyx_XGIVEREF(__pyx_r);
05551 __Pyx_RefNannyFinishContext();
05552 return __pyx_r;
05553 }
05554
05555 /* "PyClical.pyx":375
05556 * return self.instance.sign_of_mult(toIndexSet(rhs))
05557 *
05558 * def sign_of_square(self): # ««««««««
05559 * """
05560 * Sign of geometric square of a Clifford basis element.
05561 */
05562
05563 /* Python wrapper */
05564 static PyObject *__pyx_pw_8PyClical_9index_set_46sign_of_square(PyObject *__pyx_v_self, CYTHON_UNUSED
05565 PyObject *unused); /*proto*/
05566 static char __pyx_doc_8PyClical_9index_set_45sign_of_square[] = "\n Sign of geometric square of
05567 a Clifford basis element.\n\n >> s = index_set({1,2}); s.sign_of_square()\n -1\n ";
05568 static PyObject *__pyx_pw_8PyClical_9index_set_46sign_of_square(PyObject *__pyx_v_self, CYTHON_UNUSED
05569 PyObject *unused) {
05570 PyObject *__pyx_r = 0;
05571 __Pyx_RefNannyDeclarations
05572 __Pyx_RefNannySetupContext("sign_of_square (wrapper)", 0);
05573 __pyx_r = __pyx_pf_8PyClical_9index_set_45sign_of_square(((struct __pyx_obj_8PyClical_index_set
05574 *)__pyx_v_self));
05575

```

```

05566 /* function exit code */
05567 __Pyx_RefNannyFinishContext();
05568 return __pyx_r;
05569 }
05570
05571 static PyObject *__pyx_pf_8PyClical_9index_set_45sign_of_square(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self) {
05572 PyObject *__pyx_r = NULL;
05573 __Pyx_RefNannyDeclarations
05574 PyObject *__pyx_t_1 = NULL;
05575 int __pyx_lineno = 0;
05576 const char *__pyx_filename = NULL;
05577 int __pyx_clineno = 0;
05578 __Pyx_RefNannySetupContext("sign_of_square", 0);
05579
05580 /* "PyClical.pyx":382
05581 * -1
05582 * """
05583 * return self.instance.sign_of_square() # ««««««««
05584 *
05585 * def __repr__(self):
05586 */
05587 __Pyx_XDECREF(__pyx_r);
05588 __pyx_t_1 = __Pyx_PyInt_From_int(__pyx_v_self->instance->sign_of_square()); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 382, __pyx_L1_error)
05589 __Pyx_GOTREF(__pyx_t_1);
05590 __pyx_r = __pyx_t_1;
05591 __pyx_t_1 = 0;
05592 goto __pyx_L0;
05593
05594 /* "PyClical.pyx":375
05595 * return self.instance.sign_of_mult(toIndexSet(rhs))
05596 *
05597 * def sign_of_square(self): # ««««««««
05598 * """
05599 * Sign of geometric square of a Clifford basis element.
05600 */
05601
05602 /* function exit code */
05603 __pyx_L1_error:;
05604 __Pyx_XDECREF(__pyx_t_1);
05605 __Pyx_AddTraceback("PyClical.index_set.sign_of_square", __pyx_clineno, __pyx_lineno,
__pyx_filename);
05606 __pyx_r = NULL;
05607 __pyx_L0:;
05608 __Pyx_XGIVEREF(__pyx_r);
05609 __Pyx_RefNannyFinishContext();
05610 return __pyx_r;
05611 }
05612
05613 /* "PyClical.pyx":384
05614 * return self.instance.sign_of_square()
05615 *
05616 * def __repr__(self): # ««««««««
05617 * """
05618 * The official string representation of self.
05619 *
05620 */
05621 /* Python wrapper */
05622 static PyObject *__pyx_pw_8PyClical_9index_set_48__repr__(PyObject *__pyx_v_self); /*proto*/
05623 static char __pyx_doc_8PyClical_9index_set_47__repr__[] = "\n The
\n \342\200\234official\342\200\235 string representation of self.\n\n >>
index_set({1,2}).__repr__()\n 'index_set({1,2})'\n >> repr(index_set({1,2}))\n
'index_set({1,2})'\n ";
05624 #if CYTHON_COMPILING_IN_CPYTHON
05625 struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_47__repr__;
05626 #endif
05627 static PyObject *__pyx_pw_8PyClical_9index_set_48__repr__(PyObject *__pyx_v_self) {
05628 PyObject *__pyx_r = 0;
05629 __Pyx_RefNannyDeclarations
05630 __Pyx_RefNannySetupContext("__repr__ (wrapper)", 0);
05631 __pyx_r = __pyx_pf_8PyClical_9index_set_47__repr__(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self));
05632
05633 /* function exit code */
05634 __Pyx_RefNannyFinishContext();
05635 return __pyx_r;
05636 }
05637
05638 static PyObject *__pyx_pf_8PyClical_9index_set_47__repr__(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self) {
05639 PyObject *__pyx_r = NULL;
05640 __Pyx_RefNannyDeclarations
05641 PyObject *__pyx_t_1 = NULL;
05642 int __pyx_lineno = 0;
05643 const char *__pyx_filename = NULL;
05644 int __pyx_clineno = 0;

```

```

05645 __Pyx_RefNannySetupContext("__repr__", 0);
05646
05647 /* "PyClical.pyx":393
05648 * 'index_set({1,2})'
05649 * """
05650 * return index_set_to_repr(self.unwrap()).decode() # ««««««««
05651 *
05652 * def __str__(self):
05653 */
05654 __Pyx_XDECREF(__pyx_r);
05655 __pyx_t_1 =
__Pyx_decode_cpp_string(index_set_to_repr(__pyx_f_8PyClical_9index_set_unwrap(__pyx_v_self)), 0,
PY_SSIZE_T_MAX, NULL, NULL, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 393, __pyx_L1_error)
05656 __Pyx_GOTREF(__pyx_t_1);
05657 __pyx_r = __pyx_t_1;
05658 __pyx_t_1 = 0;
05659 goto __pyx_L0;
05660
05661 /* "PyClical.pyx":384
05662 * return self.instance.sign_of_square()
05663 *
05664 * def __repr__(self): # ««««««««
05665 * """
05666 * The official string representation of self.
05667 */
05668
05669 /* function exit code */
05670 __pyx_L1_error:;
05671 __Pyx_XDECREF(__pyx_t_1);
05672 __Pyx_AddTraceback("PyClical.index_set.__repr__", __pyx_clineno, __pyx_lineno, __pyx_filename);
05673 __pyx_r = NULL;
05674 __pyx_L0:;
05675 __Pyx_XGIVEREF(__pyx_r);
05676 __Pyx_RefNannyFinishContext();
05677 return __pyx_r;
05678 }
05679
05680 /* "PyClical.pyx":395
05681 * return index_set_to_repr(self.unwrap()).decode()
05682 *
05683 * def __str__(self): # ««««««««
05684 * """
05685 * The informal string representation of self.
05686 */
05687
05688 /* Python wrapper */
05689 static PyObject *__pyx_pw_8PyClical_9index_set_50_str__(PyObject *__pyx_v_self); /*proto*/
05690 static char __pyx_doc_8PyClical_9index_set_49_str__[] = "\n The\n \342\200\234informal\342\200\235 string representation of self.\n\n >>\n index_set({1,2}).__str__()\n ' {1,2} '\n >> str(index_set({1,2}))\n ' {1,2} '\n";
05691 #if CYTHON_COMPILING_IN_CPYTHON
05692 struct wrapperbase __pyx_wrapperbase_8PyClical_9index_set_49_str__;
05693 #endif
05694 static PyObject *__pyx_pw_8PyClical_9index_set_50_str__(PyObject *__pyx_v_self) {
05695 PyObject *__pyx_r = 0;
05696 __Pyx_RefNannyDeclarations
05697 __Pyx_RefNannySetupContext("__str__ (wrapper)", 0);
05698 __pyx_r = __pyx_pf_8PyClical_9index_set_49_str__(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self));
05699
05700 /* function exit code */
05701 __Pyx_RefNannyFinishContext();
05702 return __pyx_r;
05703 }
05704
05705 static PyObject *__pyx_pf_8PyClical_9index_set_49_str__(struct __pyx_obj_8PyClical_index_set
*__pyx_v_self) {
05706 PyObject *__pyx_r = NULL;
05707 __Pyx_RefNannyDeclarations
05708 PyObject *__pyx_t_1 = NULL;
05709 int __pyx_lineno = 0;
05710 const char *__pyx_filename = NULL;
05711 int __pyx_clineno = 0;
05712 __Pyx_RefNannySetupContext("__str__", 0);
05713
05714 /* "PyClical.pyx":404
05715 * ' {1,2} '
05716 * """
05717 * return index_set_to_str(self.unwrap()).decode() # ««««««««
05718 *
05719 * def index_set_hidden_doctests():
05720 */
05721 __Pyx_XDECREF(__pyx_r);
05722 __pyx_t_1 =
__Pyx_decode_cpp_string(index_set_to_str(__pyx_f_8PyClical_9index_set_unwrap(__pyx_v_self)), 0,
PY_SSIZE_T_MAX, NULL, NULL, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 404, __pyx_L1_error)

```

```

05723 __Pyx_GOTREF(__pyx_t_1);
05724 __pyx_r = __pyx_t_1;
05725 __pyx_t_1 = 0;
05726 goto __pyx_L0;
05727
05728 /* "PyClical.pyx":395
05729 * return index_set_to_repr(self.unwrap()).decode()
05730 *
05731 * def __str__(self): # ««««««««
05732 * """
05733 * The informal string representation of self.
05734 */
05735
05736 /* function exit code */
05737 __pyx_L1_error:;
05738 __Pyx_XDECREF(__pyx_t_1);
05739 __Pyx_AddTraceback("PyClical.index_set.__str__", __pyx_clineno, __pyx_lineno, __pyx_filename);
05740 __pyx_r = NULL;
05741 __pyx_L0:;
05742 __Pyx_XGIVEREF(__pyx_r);
05743 __Pyx_RefNannyFinishContext();
05744 return __pyx_r;
05745 }
05746
05747 /* "(tree fragment)":1
05748 * def __reduce_cython__(self): # ««««««««
05749 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
05750 * def __setstate_cython__(self, __pyx_state):
05751 */
05752
05753 /* Python wrapper */
05754 static PyObject *__pyx_pw_8PyClical_9index_set_52__reduce_cython__(PyObject *__pyx_v_self,
05755 CYTHON_UNUSED PyObject *unused); /*proto*/
05756 static PyObject *__pyx_pw_8PyClical_9index_set_52__reduce_cython__(PyObject *__pyx_v_self,
05757 CYTHON_UNUSED PyObject *unused) {
05758 PyObject *__pyx_r = 0;
05759 __Pyx_RefNannyDeclarations
05760 __Pyx_RefNannySetupContext("__reduce_cython__ (wrapper)", 0);
05761 __pyx_r = __pyx_pf_8PyClical_9index_set_51__reduce_cython__(((struct __pyx_obj_8PyClical_index_set
05762 *)__pyx_v_self));
05763
05764 /* function exit code */
05765 __Pyx_RefNannyFinishContext();
05766 return __pyx_r;
05767 }
05768
05769 static PyObject *__pyx_pf_8PyClical_9index_set_51__reduce_cython__(CYTHON_UNUSED struct
05770 __pyx_obj_8PyClical_index_set *__pyx_v_self) {
05771 PyObject *__pyx_r = NULL;
05772 __Pyx_RefNannyDeclarations
05773 PyObject *__pyx_t_1 = NULL;
05774 int __pyx_lineno = 0;
05775 const char *__pyx_filename = NULL;
05776 int __pyx_clineno = 0;
05777 __Pyx_RefNannySetupContext("__reduce_cython__", 0);
05778
05779 /* "(tree fragment)":2
05780 * def __reduce_cython__(self):
05781 * raise TypeError("no default __reduce__ due to non-trivial __cinit__") # ««««««««
05782 * def __setstate_cython__(self, __pyx_state):
05783 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
05784 */
05785 __pyx_t_1 = __Pyx_PyObject_Call(__pyx_builtin_TypeError, __pyx_tuple__3, NULL); if
05786 (unlikely(!__pyx_t_1)) __PYX_ERR(1, 2, __pyx_L1_error)
05787 __Pyx_GOTREF(__pyx_t_1);
05788 __Pyx_Raise(__pyx_t_1, 0, 0, 0);
05789 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
05790 __PYX_ERR(1, 2, __pyx_L1_error)
05791
05792 /* "(tree fragment)":1
05793 * def __reduce_cython__(self): # ««««««««
05794 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
05795 * def __setstate_cython__(self, __pyx_state):
05796 */
05797
05798 /* function exit code */
05799 __pyx_L1_error:;
05800 __Pyx_XDECREF(__pyx_t_1);
05801 __Pyx_AddTraceback("PyClical.index_set.__reduce_cython__", __pyx_clineno, __pyx_lineno,
05802 __pyx_filename);
05803 __pyx_r = NULL;
05804 __Pyx_XGIVEREF(__pyx_r);
05805 __Pyx_RefNannyFinishContext();
05806 return __pyx_r;
05807 }
05808
05809 /* "(tree fragment)":3

```



```

05804 * def __reduce_cython__(self):
05805 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
05806 * def __setstate_cython__(self, __pyx_state): # ««««««««
05807 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
05808 */
05809
05810 /* Python wrapper */
05811 static PyObject *__pyx_pw_8PyClical_9index_set_54__setstate_cython__(PyObject *__pyx_v_self, PyObject
*__pyx_v__pyx_state); /*proto*/
05812 static PyObject *__pyx_pw_8PyClical_9index_set_54__setstate_cython__(PyObject *__pyx_v_self, PyObject
*__pyx_v__pyx_state) {
05813 PyObject *__pyx_r = 0;
05814 __Pyx_RefNannyDeclarations
05815 __Pyx_RefNannySetupContext("__setstate_cython__ (wrapper)", 0);
05816 __pyx_r = __pyx_pf_8PyClical_9index_set_53__setstate_cython__(((struct __pyx_obj_8PyClical_index_set
*)__pyx_v_self), ((PyObject *)__pyx_v__pyx_state));
05817
05818 /* function exit code */
05819 __Pyx_RefNannyFinishContext();
05820 return __pyx_r;
05821 }
05822
05823 static PyObject *__pyx_pf_8PyClical_9index_set_53__setstate_cython__(CYTHON_UNUSED struct
__pyx_obj_8PyClical_index_set *__pyx_v_self, CYTHON_UNUSED PyObject *__pyx_v__pyx_state) {
05824 PyObject *__pyx_r = NULL;
05825 __Pyx_RefNannyDeclarations
05826 PyObject *__pyx_t_1 = NULL;
05827 int __pyx_lineno = 0;
05828 const char *__pyx_filename = NULL;
05829 int __pyx_clineno = 0;
05830 __Pyx_RefNannySetupContext("__setstate_cython__", 0);
05831
05832 /* (tree fragment)":4
05833 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
05834 * def __setstate_cython__(self, __pyx_state): # ««««««««
05835 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
05836 */
05837 __pyx_t_1 = __Pyx_PyObject_Call(__pyx_builtin_TypeError, __pyx_tuple__4, NULL); if
(unlikely(!__pyx_t_1)) __PYX_ERR(1, 4, __pyx_L1_error)
05838 __Pyx_GOTREF(__pyx_t_1);
05839 __Pyx_Raise(__pyx_t_1, 0, 0, 0);
05840 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
05841 __PYX_ERR(1, 4, __pyx_L1_error)
05842
05843 /* (tree fragment)":3
05844 * def __reduce_cython__(self):
05845 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
05846 * def __setstate_cython__(self, __pyx_state): # ««««««««
05847 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
05848 */
05849
05850 /* function exit code */
05851 __pyx_L1_error:;
05852 __Pyx_XDECREF(__pyx_t_1);
05853 __Pyx_AddTraceback("PyClical.index_set.__setstate_cython__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
05854 __pyx_r = NULL;
05855 __Pyx_XGIVEREF(__pyx_r);
05856 __Pyx_RefNannyFinishContext();
05857 return __pyx_r;
05858 }
05859
05860 /* "PyClical.pyx":406
05861 * return index_set_to_str(self.unwrap()).decode()
05862 *
05863 * def index_set_hidden_doctests(): # ««««««««
05864 * """
05865 * Tests for functions that Doctest cannot see.
05866 *
05867 */
05868 /* Python wrapper */
05869 static PyObject *__pyx_pw_8PyClical_1index_set_hidden_doctests(PyObject *__pyx_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
05870 static char __pyx_doc_8PyClical_index_set_hidden_doctests[] = "\n Tests for functions that Doctest
cannot see.\n\n For index_set.__cinit__: Construct index_set.\n\n >> print(index_set(1))\n
{1}\n >> print(index_set({1,2}))\n {1,2}\n >> print(index_set(index_set({1,2})))\n {1,2}\n
>> print(index_set({1,2}))\n {1,2}\n >> print(index_set({1,2,1}))\n {1,2}\n >>
print(index_set({1,2,1}))\n {1,2}\n >> print(index_set(\"{1}\"))\n {} \n >>
print(index_set(\"{1}\"))\n Traceback (most recent call last):\n ... \n ValueError: Cannot
initialize index_set object from invalid string '{1}'.\n >> print(index_set(\"{1}\"))\n Traceback
(most recent call last):\n ... \n ValueError: Cannot initialize index_set object from invalid
string '{1}'.\n >> print(index_set(\"{1,2,100}\"))\n Traceback (most recent call last):\n
... \n ValueError: Cannot initialize index_set object from invalid string '{1,2,100}'.\n >>
print(index_set({1,2,100}))\n Traceback (most recent call last):\n ... \n IndexError: Cannot
initialize index_set object from invalid {1, 2, 100}.\n >> print(index_set([1,2]))\n Traceback
(most recent call last):\n ... \n TypeError: Cannot initialize index_set object from <class
'list'>.\n\n For index_set.__richcmp__: Compare two objects of class index_set.\n\n >>

```

```

index_set(1) == index_set({1})\n True\n >> index_set({1}) != index_set({1})\n False\n >>
index_set({1}) != index_set({2})\n True\n >> index_set({1}) == index_set({2})\n False\n >>
index_set({1}) < index_set({2})\n True\n >> index_set({1}) <= index_set({2})\n True\n >>
index_set({1}) > index_set({2})\n False\n >> index_set({1}) >= index_set({2})\n False\n >>
None == index_set({1,2})\n False\n >> None != index_set({1,2})\n True\n >> None <
index_set({1,2})\n False\n >> None <= index_set({1,2})\n False\n >> None >
index_set({1,2})\n False\n >> None >= index_set({1,2})\n False\n >> "index_set({1,2}) ==
None\n False\n >> index_set({1,2}) != None\n True\n >> index_set({1,2}) < None\n
False\n >> index_set({1,2}) <= None\n False\n >> index_set({1,2}) > None\n False\n >>
index_set({1,2}) >= None\n False\n ";
05871 static PyMethodDef __pyx_mdef_8PyClical_lindex_set_hidden_doctests =
{"index_set_hidden_doctests", (PyCFunction)__pyx_pw_8PyClical_lindex_set_hidden_doctests, METH_NOARGS,
__pyx_doc_8PyClical_index_set_hidden_doctests};
05872 static PyObject *__pyx_pw_8PyClical_lindex_set_hidden_doctests(PyObject *__pyx_self,
CYTHON_UNUSED PyObject *unused) {
05873 PyObject *__pyx_r = 0;
05874 __Pyx_RefNannyDeclarations
05875 __Pyx_RefNannySetupContext("index_set_hidden_doctests (wrapper)", 0);
05876 __pyx_r = __pyx_pf_8PyClical_index_set_hidden_doctests(__pyx_self);
05877
05878 /* function exit code */
05879 __Pyx_RefNannyFinishContext();
05880 return __pyx_r;
05881 }
05882
05883 static PyObject *__pyx_pf_8PyClical_index_set_hidden_doctests(CYTHON_UNUSED PyObject
*__pyx_self) {
05884 PyObject *__pyx_r = NULL;
05885 __Pyx_RefNannyDeclarations
05886 __Pyx_RefNannySetupContext("index_set_hidden_doctests", 0);
05887
05888 /* "PyClical.pyx":490
05889 * False
05890 * """
05891 * return # ««««««««
05892 *
05893 * cpdef inline compare(lhs,rhs):
05894 */
05895 __Pyx_XDECREF(__pyx_r);
05896 __pyx_r = Py_None; __Pyx_INCREF(Py_None);
05897 goto __pyx_L0;
05898
05899 /* "PyClical.pyx":406
05900 * return index_set_to_str(self.unwrap()).decode()
05901 *
05902 * def index_set_hidden_doctests(): # ««««««««
05903 * """
05904 * Tests for functions that Doctest cannot see.
05905 */
05906
05907 /* function exit code */
05908 __pyx_L0:;
05909 __Pyx_XGIVEREF(__pyx_r);
05910 __Pyx_RefNannyFinishContext();
05911 return __pyx_r;
05912 }
05913
05914 /* "PyClical.pyx":492
05915 * return
05916 *
05917 * cpdef inline compare(lhs,rhs): # ««««««««
05918 * """
05919 * "lexicographic compare" eg. {3,4,5} is less than {3,7,8};
05920 */
05921
05922 static PyObject *__pyx_pw_8PyClical_3compare(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds); /*proto*/
05923 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_compare(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs, CYTHON_UNUSED int __pyx_skip_dispatch) {
05924 PyObject *__pyx_r = NULL;
05925 __Pyx_RefNannyDeclarations
05926 PyObject *__pyx_t_1 = NULL;
05927 int __pyx_lineno = 0;
05928 const char *__pyx_filename = NULL;
05929 int __pyx_clineno = 0;
05930 __Pyx_RefNannySetupContext("compare", 0);
05931
05932 /* "PyClical.pyx":502
05933 * 1
05934 * """
05935 * return glucat.compare(toIndexSet(lhs), toIndexSet(rhs)) # ««««««««
05936 *
05937 * cpdef inline min_neg(obj):
05938 */
05939 __Pyx_XDECREF(__pyx_r);
05940 __pyx_t_1 = __Pyx_PyInt_From_int(compare(__pyx_f_8PyClical_toIndexSet(__pyx_v_lhs),
__pyx_f_8PyClical_toIndexSet(__pyx_v_rhs))); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 502,

```

```

__pyx_L1_error)
05941 __Pyx_GOTREF(__pyx_t_1);
05942 __pyx_r = __pyx_t_1;
05943 __pyx_t_1 = 0;
05944 goto __pyx_L0;
05945
05946 /* "PyClical.pyx":492
05947 * return
05948 *
05949 * cpdef inline compare(lhs,rhs): # ««««««««
05950 * """
05951 * "lexicographic compare" eg. {3,4,5} is less than {3,7,8};
05952 */
05953
05954 /* function exit code */
05955 __pyx_L1_error++;
05956 __Pyx_XDECREF(__pyx_t_1);
05957 __Pyx_AddTraceback("PyClical.compare", __pyx_clineno, __pyx_lineno, __pyx_filename);
05958 __pyx_r = 0;
05959 __pyx_L0++;
05960 __Pyx_XGIVEREF(__pyx_r);
05961 __Pyx_RefNannyFinishContext();
05962 return __pyx_r;
05963 }
05964
05965 /* Python wrapper */
05966 static PyObject *__pyx_pw_8PyClical_3compare(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds); /*proto*/
05967 static char __pyx_doc_8PyClical_2compare[] = "\n \"lexicographic compare\" eg. {3,4,5} is
less than {3,7,8};\n -1 if a<b, +1 if a>b, 0 if a==b.\n\n >>
compare(index_set({1,2}),index_set({-1,3}))\n -1\n >>
compare(index_set({-1,4}),index_set({-1,3}))\n 1\n ";
05968 static PyObject *__pyx_pw_8PyClical_3compare(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds) {
05969 PyObject *__pyx_v_lhs = 0;
05970 PyObject *__pyx_v_rhs = 0;
05971 int __pyx_lineno = 0;
05972 const char *__pyx_filename = NULL;
05973 int __pyx_clineno = 0;
05974 PyObject *__pyx_r = 0;
05975 __Pyx_RefNannyDeclarations
05976 __Pyx_RefNannySetupContext("compare (wrapper)", 0);
05977 {
05978 static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_lhs,&__pyx_n_s_rhs,0};
05979 PyObject* values[2] = {0,0};
05980 if (unlikely(__pyx_kwds)) {
05981 Py_ssize_t kw_args;
05982 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
05983 switch (pos_args) {
05984 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
05985 CYTHON_FALLTHROUGH;
05986 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
05987 CYTHON_FALLTHROUGH;
05988 case 0: break;
05989 default: goto __pyx_L5_argtuple_error;
05990 }
05991 kw_args = PyDict_Size(__pyx_kwds);
05992 switch (pos_args) {
05993 case 0:
05994 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_lhs)) != 0))
kw_args--;
05995 else goto __pyx_L5_argtuple_error;
05996 CYTHON_FALLTHROUGH;
05997 case 1:
05998 if (likely((values[1] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_rhs)) != 0))
kw_args--;
05999 else {
06000 __Pyx_RaiseArgtupleInvalid("compare", 1, 2, 2, 1); __PYX_ERR(0, 492, __pyx_L3_error)
06001 }
06002 if (unlikely(kw_args > 0)) {
06003 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values,
pos_args, "compare") < 0)) __PYX_ERR(0, 492, __pyx_L3_error)
06004 }
06005 } else if (PyTuple_GET_SIZE(__pyx_args) != 2) {
06006 goto __pyx_L5_argtuple_error;
06007 } else {
06008 values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
06009 values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
06010 }
06011 __pyx_v_lhs = values[0];
06012 __pyx_v_rhs = values[1];
06013 }
06014 goto __pyx_L4_argument_unpacking_done;
06015 __pyx_L5_argtuple_error++;
06016 __Pyx_RaiseArgtupleInvalid("compare", 1, 2, 2, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0,
492, __pyx_L3_error)

```

```

06018 __pyx_L3_error;;
06019 __Pyx_AddTraceback("PyClical.compare", __pyx_clineno, __pyx_lineno, __pyx_filename);
06020 __Pyx_RefNannyFinishContext();
06021 return NULL;
06022 __pyx_L4_argument_unpacking_done;;
06023 __pyx_r = __pyx_pf_8PyClical_2compare(__pyx_self, __pyx_v_lhs, __pyx_v_rhs);
06024
06025 /* function exit code */
06026 __Pyx_RefNannyFinishContext();
06027 return __pyx_r;
06028 }
06029
06030 static PyObject *__pyx_pf_8PyClical_2compare(CYTHON_UNUSED PyObject *__pyx_self, PyObject
06031 *__pyx_v_lhs, PyObject *__pyx_v_rhs) {
06032 PyObject *__pyx_r = NULL;
06033 __Pyx_RefNannyDeclarations
06034 PyObject *__pyx_t_1 = NULL;
06035 int __pyx_lineno = 0;
06036 const char *__pyx_filename = NULL;
06037 int __pyx_clineno = 0;
06038 __Pyx_RefNannySetupContext("compare", 0);
06039 __Pyx_XDECREF(__pyx_r);
06040 __pyx_t_1 = __pyx_f_8PyClical_compare(__pyx_v_lhs, __pyx_v_rhs, 0); if
06041 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 492, __pyx_L1_error)
06042 __Pyx_GOTREF(__pyx_t_1);
06043 __pyx_r = __pyx_t_1;
06044 __pyx_t_1 = 0;
06045 goto __pyx_L0;
06046
06047 /* function exit code */
06048 __Pyx_XDECREF(__pyx_t_1);
06049 __Pyx_AddTraceback("PyClical.compare", __pyx_clineno, __pyx_lineno, __pyx_filename);
06050 __pyx_r = NULL;
06051 __pyx_L0;;
06052 __Pyx_XGIVEREF(__pyx_r);
06053 __Pyx_RefNannyFinishContext();
06054 return __pyx_r;
06055 }
06056
06057 /* "PyClical.pyx":504
06058 * return glucat.compare(toIndexSet(lhs), toIndexSet(rhs))
06059 *
06060 * cpdef inline min_neg(obj): # ««««««««
06061 * """
06062 * Minimum negative index, or 0 if none.
06063 */
06064 static PyObject *__pyx_pw_8PyClical_5min_neg(PyObject *__pyx_self, PyObject *__pyx_v_obj);
06065
06066 /*proto*/
06067 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_min_neg(PyObject *__pyx_v_obj, CYTHON_UNUSED
06068 int __pyx_skip_dispatch) {
06069 PyObject *__pyx_r = NULL;
06070 __Pyx_RefNannyDeclarations
06071 PyObject *__pyx_t_1 = NULL;
06072 int __pyx_lineno = 0;
06073 const char *__pyx_filename = NULL;
06074 int __pyx_clineno = 0;
06075 __Pyx_RefNannySetupContext("min_neg", 0);
06076
06077 /* "PyClical.pyx":511
06078 * 0
06079 * """
06080 * return glucat.min_neg(toIndexSet(obj)) # ««««««««
06081 *
06082 * cpdef inline max_pos(obj):
06083 */
06084 __Pyx_XDECREF(__pyx_r);
06085 __pyx_t_1 = __Pyx_PyInt_From_int(min_neg(__pyx_f_8PyClical_toIndexSet(__pyx_v_obj))); if
06086 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 511, __pyx_L1_error)
06087 __Pyx_GOTREF(__pyx_t_1);
06088 __pyx_r = __pyx_t_1;
06089 __pyx_t_1 = 0;
06090 goto __pyx_L0;
06091
06092 /* "PyClical.pyx":504
06093 * return glucat.compare(toIndexSet(lhs), toIndexSet(rhs))
06094 *
06095 * cpdef inline min_neg(obj): # ««««««««
06096 * """
06097 * Minimum negative index, or 0 if none.
06098 */
06099 /* function exit code */
06100 __Pyx_L1_error;;
06101 __Pyx_XDECREF(__pyx_t_1);
06102 __Pyx_AddTraceback("PyClical.min_neg", __pyx_clineno, __pyx_lineno, __pyx_filename);

```

```

06100 __pyx_r = 0;
06101 __pyx_L0:;
06102 __Pyx_XGIVEREF(__pyx_r);
06103 __Pyx_RefNannyFinishContext();
06104 return __pyx_r;
06105 }
06106
06107 /* Python wrapper */
06108 static PyObject *__pyx_pw_8PyClical_5min_neg(PyObject *__pyx_self, PyObject *__pyx_v_obj);
06109 /*proto*/
06110 static char __pyx_doc_8PyClical_4min_neg[] = "\n Minimum negative index, or 0 if none.\n\n
>> min_neg(index_set({1,2}))\n 0\n ";
06111 static PyObject *__pyx_pw_8PyClical_5min_neg(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
06112 PyObject *__pyx_r = 0;
06113 __Pyx_RefNannyDeclarations
06114 __Pyx_RefNannySetupContext("min_neg (wrapper)", 0);
06115 __pyx_r = __pyx_pf_8PyClical_4min_neg(__pyx_self, ((PyObject *)__pyx_v_obj));
06116
06117 /* function exit code */
06118 __Pyx_RefNannyFinishContext();
06119 return __pyx_r;
06120 }
06121
06122 static PyObject *__pyx_pf_8PyClical_4min_neg(CYTHON_UNUSED PyObject *__pyx_self, PyObject
__pyx_v_obj) {
06123 PyObject *__pyx_r = NULL;
06124 __Pyx_RefNannyDeclarations
06125 PyObject *__pyx_t_1 = NULL;
06126 int __pyx_lineno = 0;
06127 const char *__pyx_filename = NULL;
06128 int __pyx_clineno = 0;
06129 __Pyx_RefNannySetupContext("min_neg", 0);
06130 __Pyx_XDECREF(__pyx_r);
06131 __pyx_t_1 = __pyx_f_8PyClical_min_neg(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 504, __pyx_L1_error)
06132 __Pyx_GOTREF(__pyx_t_1);
06133 __pyx_r = __pyx_t_1;
06134 __pyx_t_1 = 0;
06135 goto __pyx_L0;
06136
06137 /* function exit code */
06138 __pyx_L1_error:;
06139 __Pyx_XDECREF(__pyx_t_1);
06140 __Pyx_AddTraceback("PyClical.min_neg", __pyx_clineno, __pyx_lineno, __pyx_filename);
06141 __pyx_r = NULL;
06142 __pyx_L0:;
06143 __Pyx_XGIVEREF(__pyx_r);
06144 __Pyx_RefNannyFinishContext();
06145 return __pyx_r;
06146 }
06147
06148 /* "PyClical.pyx":513
* return glucat.min_neg(toIndexSet(obj))
06149 *
06150 * cpdef inline max_pos(obj):
06151 * """
06152 * Maximum positive index, or 0 if none.
06153 * */
06154
06155 static PyObject *__pyx_pw_8PyClical_7max_pos(PyObject *__pyx_self, PyObject *__pyx_v_obj);
06156 /*proto*/
06157 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_max_pos(PyObject *__pyx_v_obj, CYTHON_UNUSED
int __pyx_skip_dispatch) {
06158 PyObject *__pyx_r = NULL;
06159 __Pyx_RefNannyDeclarations
06160 PyObject *__pyx_t_1 = NULL;
06161 int __pyx_lineno = 0;
06162 const char *__pyx_filename = NULL;
06163 int __pyx_clineno = 0;
06164 __Pyx_RefNannySetupContext("max_pos", 0);
06165
06166 /* "PyClical.pyx":520
* 2
* """
06167 * return glucat.max_pos(toIndexSet(obj))
06168 * # ««««««««
06169 *
06170 * cdef inline vector[sclar_t] list_to_vector(lst):
06171 */
06172 __Pyx_XDECREF(__pyx_r);
06173 __pyx_t_1 = __Pyx_PyInt_From_int(max_pos(__pyx_f_8PyClical_toIndexSet(__pyx_v_obj))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 520, __pyx_L1_error)
06174 __Pyx_GOTREF(__pyx_t_1);
06175 __pyx_r = __pyx_t_1;
06176 __pyx_t_1 = 0;
06177 goto __pyx_L0;
06178
06179 /* "PyClical.pyx":513

```

```

06180 * return glucat.min_neg(toIndexSet(obj))
06181 *
06182 * cpdef inline max_pos(obj): # ««««««««
06183 * """
06184 * Maximum positive index, or 0 if none.
06185 */
06186
06187 /* function exit code */
06188 __pyx_Ll_error++;
06189 __Pyx_XDECREF(__pyx_t_1);
06190 __Pyx_AddTraceback("PyClical.max_pos", __pyx_clineno, __pyx_lineno, __pyx_filename);
06191 __pyx_r = 0;
06192 __pyx_L0++;
06193 __Pyx_XGIVEREF(__pyx_r);
06194 __Pyx_RefNannyFinishContext();
06195 return __pyx_r;
06196 }
06197
06198 /* Python wrapper */
06199 static PyObject *__pyx_pw_8PyClical_7max_pos(PyObject *__pyx_self, PyObject *__pyx_v_obj);
/*proto*/
06200 static char __pyx_doc_8PyClical_6max_pos[] = "\n Maximum positive index, or 0 if none.\n\n
>> max_pos(index_set({1,2}))\n 2\n ";
06201 static PyObject *__pyx_pw_8PyClical_7max_pos(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
06202 PyObject *__pyx_r = 0;
06203 __Pyx_RefNannyDeclarations
06204 __Pyx_RefNannySetupContext("max_pos (wrapper)", 0);
06205 __pyx_r = __pyx_pf_8PyClical_6max_pos(__pyx_self, ((PyObject *)__pyx_v_obj));
06206
06207 /* function exit code */
06208 __Pyx_RefNannyFinishContext();
06209 return __pyx_r;
06210 }
06211
06212 static PyObject *__pyx_pf_8PyClical_6max_pos(CYTHON_UNUSED PyObject *__pyx_self, PyObject
__pyx_v_obj) {
06213 PyObject *__pyx_r = NULL;
06214 __Pyx_RefNannyDeclarations
06215 PyObject *__pyx_t_1 = NULL;
06216 int __pyx_lineno = 0;
06217 const char *__pyx_filename = NULL;
06218 int __pyx_clineno = 0;
06219 __Pyx_RefNannySetupContext("max_pos", 0);
06220 __Pyx_XDECREF(__pyx_r);
06221 __pyx_t_1 = __pyx_f_8PyClical_max_pos(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 513, __pyx_Ll_error)
06222 __Pyx_GOTREF(__pyx_t_1);
06223 __pyx_r = __pyx_t_1;
06224 __pyx_t_1 = 0;
06225 goto __pyx_L0;
06226
06227 /* function exit code */
06228 __pyx_Ll_error++;
06229 __Pyx_XDECREF(__pyx_t_1);
06230 __Pyx_AddTraceback("PyClical.max_pos", __pyx_clineno, __pyx_lineno, __pyx_filename);
06231 __pyx_r = NULL;
06232 __pyx_L0++;
06233 __Pyx_XGIVEREF(__pyx_r);
06234 __Pyx_RefNannyFinishContext();
06235 return __pyx_r;
06236 }
06237
06238 /* "PyClical.pyx":522
06239 * return glucat.max_pos(toIndexSet(obj))
06240 *
06241 * cdef inline vector[scalar_t] list_to_vector(lst): # ««««««««
06242 * """
06243 * Create a C++ std::vector[scalar_t] from an iterable Python object.
06244 */
06245
06246 static CYTHON_INLINE std::vector<scalar_t> __pyx_f_8PyClical_list_to_vector(PyObject
__pyx_v_lst) {
06247 std::vector<scalar_t> __pyx_v_v;
06248 PyObject *__pyx_v_s = NULL;
06249 std::vector<scalar_t> __pyx_r;
06250 __Pyx_RefNannyDeclarations
06251 PyObject *__pyx_t_1 = NULL;
06252 Py_ssize_t __pyx_t_2;
06253 PyObject *(*__pyx_t_3)(PyObject *);
06254 PyObject *__pyx_t_4 = NULL;
06255 scalar_t __pyx_t_5;
06256 int __pyx_lineno = 0;
06257 const char *__pyx_filename = NULL;
06258 int __pyx_clineno = 0;
06259 __Pyx_RefNannySetupContext("list_to_vector", 0);
06260
06261 /* "PyClical.pyx":527

```

```

06262 * """
06263 * cdef vector[scalar_t] v
06264 * for s in lst: # ««««««««
06265 * v.push_back(<scalar_t>s)
06266 * return v
06267 */
06268 if (likely(PyList_CheckExact(__pyx_v_lst)) || PyTuple_CheckExact(__pyx_v_lst)) {
06269 __pyx_t_1 = __pyx_v_lst; __Pyx_INCREF(__pyx_t_1); __pyx_t_2 = 0;
06270 __pyx_t_3 = NULL;
06271 } else {
06272 __pyx_t_2 = -1; __pyx_t_1 = PyObject_GetIter(__pyx_v_lst); if (unlikely(!__pyx_t_1))
06273 __PYX_ERR(0, 527, __pyx_L1_error)
06274 __Pyx_GOTREF(__pyx_t_1);
06275 __pyx_t_3 = Py_TYPE(__pyx_t_1)->tp_iternext; if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 527,
06276 __pyx_L1_error)
06277 }
06278 for (;;) {
06279 if (likely(!__pyx_t_3)) {
06280 if (likely(PyList_CheckExact(__pyx_t_1))) {
06281 if (__pyx_t_2 >= PyList_GET_SIZE(__pyx_t_1)) break;
06282 #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
06283 __pyx_t_4 = PyList_GET_ITEM(__pyx_t_1, __pyx_t_2); __Pyx_INCREF(__pyx_t_4);
06284 __pyx_t_2++; if (unlikely(0 < 0)) __PYX_ERR(0, 527, __pyx_L1_error)
06285 #else
06286 __pyx_t_4 = PySequence_ITEM(__pyx_t_1, __pyx_t_2); __pyx_t_2++; if
06287 (unlikely(!__pyx_t_4)) __PYX_ERR(0, 527, __pyx_L1_error)
06288 __Pyx_GOTREF(__pyx_t_4);
06289 #endif
06290 } else {
06291 if (__pyx_t_2 >= PyTuple_GET_SIZE(__pyx_t_1)) break;
06292 #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
06293 __pyx_t_4 = PyTuple_GET_ITEM(__pyx_t_1, __pyx_t_2); __Pyx_INCREF(__pyx_t_4);
06294 __pyx_t_2++; if (unlikely(0 < 0)) __PYX_ERR(0, 527, __pyx_L1_error)
06295 #else
06296 __pyx_t_4 = PySequence_ITEM(__pyx_t_1, __pyx_t_2); __pyx_t_2++; if
06297 (unlikely(!__pyx_t_4)) __PYX_ERR(0, 527, __pyx_L1_error)
06298 __Pyx_GOTREF(__pyx_t_4);
06299 #endif
06300 }
06301 } else {
06302 __pyx_t_4 = __pyx_t_3(__pyx_t_1);
06303 if (unlikely(!__pyx_t_4)) {
06304 PyObject* exc_type = PyErr_Occurred();
06305 if (exc_type) {
06306 if (likely(__Pyx_PyErr_GivenExceptionMatches(exc_type, PyExc_StopIteration)))
06307 PyErr_Clear();
06308 else __PYX_ERR(0, 527, __pyx_L1_error)
06309 }
06310 break;
06311 }
06312 __Pyx_GOTREF(__pyx_t_4);
06313 }
06314 __Pyx_XDECREF_SET(__pyx_v_s, __pyx_t_4);
06315 __pyx_t_4 = 0;
06316 }
06317 /* "PyClical.pyx":528
06318 * cdef vector[scalar_t] v
06319 * for s in lst:
06320 * v.push_back(<scalar_t>s) # ««««««««
06321 * return v
06322 */
06323 __pyx_t_5 = __pyx_PyFloat_AsDouble(__pyx_v_s); if (unlikely((__pyx_t_5 == ((scalar_t)-1))
06324 && PyErr_Occurred())) __PYX_ERR(0, 528, __pyx_L1_error)
06325 try {
06326 __pyx_v_v.push_back(((scalar_t)__pyx_t_5));
06327 } catch (...) {
06328 __Pyx_CppExn2PyErr();
06329 __PYX_ERR(0, 528, __pyx_L1_error)
06330 }
06331 }
06332 /* "PyClical.pyx":527
06333 * """
06334 * cdef vector[scalar_t] v
06335 * for s in lst: # ««««««««
06336 * v.push_back(<scalar_t>s)
06337 * return v
06338 */
06339 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
06340 /* "PyClical.pyx":529
06341 * for s in lst:
06342 * v.push_back(<scalar_t>s)
06343 * return v # ««««««««
06344 * # Forward reference.

```

```

06341 */
06342 __pyx_r = __pyx_v_v;
06343 goto __pyx_L0;
06344
06345 /* "PyClical.pyx":522
06346 * return glucat.max_pos(toIndexSet(obj))
06347 *
06348 * cdef inline vector[scalar_t] list_to_vector(lst): # ««««««««
06349 * """
06350 * Create a C++ std:vector[scalar_t] from an iterable Python object.
06351 */
06352
06353 /* function exit code */
06354 __pyx_L1_error:;
06355 __Pyx_XDECREF(__pyx_t_1);
06356 __Pyx_XDECREF(__pyx_t_4);
06357 __Pyx_WriteUnraisable("PyClical.list_to_vector", __pyx_clineno, __pyx_lineno,
__pyx_filename, 1, 0);
06358 __Pyx_pretend_to_initialize(&__pyx_r);
06359 __pyx_L0:;
06360 __Pyx_XDECREF(__pyx_v_s);
06361 __Pyx_RefNannyFinishContext();
06362 return __pyx_r;
06363 }
06364
06365 /* "PyClical.pyx":534
06366 * cdef class clifford
06367 *
06368 * cdef inline Clifford toClifford(obj): # ««««««««
06369 * return clifford(obj).instance[0]
06370 *
06371 */
06372
06373 static CYTHON_INLINE Clifford __pyx_f_8PyClical_toClifford(PyObject *__pyx_v_obj) {
06374 Clifford __pyx_r;
06375 __Pyx_RefNannyDeclarations
06376 PyObject *__pyx_t_1 = NULL;
06377 int __pyx_lineno = 0;
06378 const char *__pyx_filename = NULL;
06379 int __pyx_clineno = 0;
06380 __Pyx_RefNannySetupContext("toClifford", 0);
06381
06382 /* "PyClical.pyx":535
06383 *
06384 * cdef inline Clifford toClifford(obj):
06385 * return clifford(obj).instance[0] # ««««««««
06386 *
06387 * cdef class clifford:
06388 */
06389 __pyx_t_1 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 535, __pyx_L1_error)
06390 __Pyx_GOTREF(__pyx_t_1);
06391 __pyx_r = (((struct __pyx_obj_8PyClical_clifford *)__pyx_t_1)->instance[0]);
06392 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
06393 goto __pyx_L0;
06394
06395 /* "PyClical.pyx":534
06396 * cdef class clifford
06397 *
06398 * cdef inline Clifford toClifford(obj): # ««««««««
06399 * return clifford(obj).instance[0]
06400 *
06401 */
06402
06403 /* function exit code */
06404 __pyx_L1_error:;
06405 __Pyx_XDECREF(__pyx_t_1);
06406 __Pyx_WriteUnraisable("PyClical.toClifford", __pyx_clineno, __pyx_lineno, __pyx_filename, 1,
0);
06407 __Pyx_pretend_to_initialize(&__pyx_r);
06408 __pyx_L0:;
06409 __Pyx_RefNannyFinishContext();
06410 return __pyx_r;
06411 }
06412
06413 /* "PyClical.pyx":543
06414 * cdef Clifford *instance # Wrapped instance of C++ class Clifford.
06415 *
06416 * cdef inline wrap(clifford self, Clifford other): # ««««««««
06417 * """
06418 * Wrap an instance of the C++ class Clifford.
06419 */
06420
06421 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_8clifford_wrap(struct
__pyx_obj_8PyClical_clifford *__pyx_v_self, Clifford __pyx_v_other) {
06422 PyObject *__pyx_r = NULL;
06423 __Pyx_RefNannyDeclarations

```



```

06424 __Pyx_RefNannySetupContext("wrap", 0);
06425
06426 /* "PyClical.pyx":547
06427 * Wrap an instance of the C++ class Clifford.
06428 * """
06429 * self.instance[0] = other # ««««««««
06430 * return self
06431 *
06432 */
06433 (__pyx_v_self->instance[0]) = __pyx_v_other;
06434
06435 /* "PyClical.pyx":548
06436 * """
06437 * self.instance[0] = other
06438 * return self # ««««««««
06439 *
06440 * cdef inline Clifford unwrap(clifford self):
06441 */
06442 __Pyx_XDECREF(__pyx_r);
06443 __Pyx_INCREF((PyObject *)__pyx_v_self);
06444 __pyx_r = (PyObject *)__pyx_v_self;
06445 goto __pyx_L0;
06446
06447 /* "PyClical.pyx":543
06448 * cdef Clifford *instance # Wrapped instance of C++ class Clifford.
06449 *
06450 * cdef inline wrap(clifford self, Clifford other): # ««««««««
06451 * """
06452 * Wrap an instance of the C++ class Clifford.
06453 */
06454
06455 /* function exit code */
06456 __pyx_L0:;
06457 __Pyx_XGIVEREF(__pyx_r);
06458 __Pyx_RefNannyFinishContext();
06459 return __pyx_r;
06460 }
06461
06462 /* "PyClical.pyx":550
06463 * return self
06464 *
06465 * cdef inline Clifford unwrap(clifford self): # ««««««««
06466 * """
06467 * Return the wrapped C++ Clifford instance.
06468 */
06469
06470 static CYTHON_INLINE Clifford __pyx_f_8PyClical_8clifford_unwrap(struct
__pyx_obj_8PyClical_clifford *__pyx_v_self) {
06471 Clifford __pyx_r;
06472 __Pyx_RefNannyDeclarations
06473 __Pyx_RefNannySetupContext("unwrap", 0);
06474
06475 /* "PyClical.pyx":554
06476 * Return the wrapped C++ Clifford instance.
06477 * """
06478 * return self.instance[0] # ««««««««
06479 *
06480 * cpdef copy(clifford self):
06481 */
06482 __pyx_r = (__pyx_v_self->instance[0]);
06483 goto __pyx_L0;
06484
06485 /* "PyClical.pyx":550
06486 * return self
06487 *
06488 * cdef inline Clifford unwrap(clifford self): # ««««««««
06489 * """
06490 * Return the wrapped C++ Clifford instance.
06491 */
06492
06493 /* function exit code */
06494 __pyx_L0:;
06495 __Pyx_RefNannyFinishContext();
06496 return __pyx_r;
06497 }
06498
06499 /* "PyClical.pyx":556
06500 * return self.instance[0]
06501 *
06502 * cpdef copy(clifford self): # ««««««««
06503 * """
06504 * Copy this clifford object.
06505 */
06506
06507 static PyObject *__pyx_pw_8PyClical_8clifford_1copy(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
06508 static PyObject *__pyx_f_8PyClical_8clifford_copy(struct __pyx_obj_8PyClical_clifford

```

```

 *__pyx_v_self, int __pyx_skip_dispatch) {
06509 PyObject *__pyx_r = NULL;
06510 __Pyx_RefNannyDeclarations
06511 PyObject *__pyx_t_1 = NULL;
06512 PyObject *__pyx_t_2 = NULL;
06513 PyObject *__pyx_t_3 = NULL;
06514 PyObject *__pyx_t_4 = NULL;
06515 int __pyx_lineno = 0;
06516 const char *__pyx_filename = NULL;
06517 int __pyx_clineno = 0;
06518 __Pyx_RefNannySetupContext("copy", 0);
06519 /* Check if called by wrapper */
06520 if (unlikely(__pyx_skip_dispatch)) ;
06521 /* Check if overridden in Python */
06522 else if (unlikely((Py_TYPE((PyObject *)__pyx_v_self)->tp_dictoffset != 0) ||
(Py_TYPE((PyObject *)__pyx_v_self)->tp_flags & (Py_TPFLAGS_IS_ABSTRACT | Py_TPFLAGS_HEAPTYPE)))) {
06523 #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_PYTYPE_LOOKUP && CYTHON_USE_TYPE_SLOTS
06524 static PY_UINT64_T __pyx_tp_dict_version = __PYX_DICT_VERSION_INIT, __pyx_obj_dict_version
= __PYX_DICT_VERSION_INIT;
06525 if (unlikely(!__Pyx_object_dict_version_matches((PyObject *)__pyx_v_self),
__pyx_tp_dict_version, __pyx_obj_dict_version)) {
06526 PY_UINT64_T __pyx_type_dict_guard = __Pyx_get_tp_dict_version((PyObject
*)__pyx_v_self));
06527 #endif
06528 __pyx_t_1 = __Pyx_PyObject_GetAttrStr((PyObject *)__pyx_v_self, __pyx_n_s_copy); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 556, __pyx_L1_error)
06529 __Pyx_GOTREF(__pyx_t_1);
06530 if (!PyCFunction_Check(__pyx_t_1) || (PyCFunction_GET_FUNCTION(__pyx_t_1) !=
(PyCFunction)(void*)__pyx_pw_8PyClical_8clifford_lcopy)) {
06531 __Pyx_XDECREF(__pyx_r);
06532 __Pyx_INCREF(__pyx_t_1);
06533 __pyx_t_3 = __pyx_t_1; __pyx_t_4 = NULL;
06534 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_3))) {
06535 __pyx_t_4 = PyMethod_GET_SELF(__pyx_t_3);
06536 if (likely(__pyx_t_4)) {
06537 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
06538 __Pyx_INCREF(__pyx_t_4);
06539 __Pyx_INCREF(function);
06540 __Pyx_DECREF_SET(__pyx_t_3, function);
06541 }
06542 }
06543 __pyx_t_2 = (__pyx_t_4) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_4) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
06544 __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
06545 if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 556, __pyx_L1_error)
06546 __Pyx_GOTREF(__pyx_t_2);
06547 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
06548 __pyx_r = __pyx_t_2;
06549 __pyx_t_2 = 0;
06550 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
06551 goto __pyx_L0;
06552 }
06553 #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_PYTYPE_LOOKUP && CYTHON_USE_TYPE_SLOTS
06554 __pyx_tp_dict_version = __Pyx_get_tp_dict_version((PyObject *)__pyx_v_self);
06555 __pyx_obj_dict_version = __Pyx_get_object_dict_version((PyObject *)__pyx_v_self);
06556 if (unlikely(__pyx_type_dict_guard != __pyx_tp_dict_version)) {
06557 __pyx_tp_dict_version = __pyx_obj_dict_version = __PYX_DICT_VERSION_INIT;
06558 }
06559 #endif
06560 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
06561 #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_PYTYPE_LOOKUP && CYTHON_USE_TYPE_SLOTS
06562 }
06563 #endif
06564 }
06565
06566 /* "PyClical.pyx":563
06567 {2}
06568 """
06569 return clifford(self) # ««««««««
06570 */
06571 def __cinit__(self, other = 0, ixt = None):
06572 */
06573 __Pyx_XDECREF(__pyx_r);
06574 __pyx_t_1 = __Pyx_PyObject_CallOneArg((PyObject *)__pyx_ptype_8PyClical_clifford),
((PyObject *)__pyx_v_self); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 563, __pyx_L1_error)
06575 __Pyx_GOTREF(__pyx_t_1);
06576 __pyx_r = __pyx_t_1;
06577 __pyx_t_1 = 0;
06578 goto __pyx_L0;
06579
06580 /* "PyClical.pyx":556
06581 return self.instance[0]
06582 */
06583 cpdef copy(clifford self): # ««««««««
06584 """
06585 Copy this clifford object.
06586 */

```

```

06587
06588 /* function exit code */
06589 __pyx_L1_error++;
06590 __Pyx_XDECREF(__pyx_t_1);
06591 __Pyx_XDECREF(__pyx_t_2);
06592 __Pyx_XDECREF(__pyx_t_3);
06593 __Pyx_XDECREF(__pyx_t_4);
06594 __Pyx_AddTraceback("PyClical.clifford.copy", __pyx_clineno, __pyx_lineno, __pyx_filename);
06595 __pyx_r = 0;
06596 __pyx_L0++;
06597 __Pyx_XGIVEREF(__pyx_r);
06598 __Pyx_RefNannyFinishContext();
06599 return __pyx_r;
06600 }
06601
06602 /* Python wrapper */
06603 static PyObject *__pyx_pw_8PyClical_8clifford_1copy(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
06604 static char __pyx_doc_8PyClical_8clifford_copy[] = "\n Copy this clifford object.\n\n
>> x=clifford(\"1{2}\"); y=x.copy(); print(y)\n {2}\n ";
06605 static PyObject *__pyx_pw_8PyClical_8clifford_1copy(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
06606 PyObject *__pyx_r = 0;
06607 __Pyx_RefNannyDeclarations
06608 __Pyx_RefNannySetupContext("copy (wrapper)", 0);
06609 __pyx_r = __pyx_pf_8PyClical_8clifford_copy(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
06610
06611 /* function exit code */
06612 __Pyx_RefNannyFinishContext();
06613 return __pyx_r;
06614 }
06615
06616 static PyObject *__pyx_pf_8PyClical_8clifford_copy(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
06617 PyObject *__pyx_r = NULL;
06618 __Pyx_RefNannyDeclarations
06619 PyObject *__pyx_t_1 = NULL;
06620 int __pyx_lineno = 0;
06621 const char *__pyx_filename = NULL;
06622 int __pyx_clineno = 0;
06623 __Pyx_RefNannySetupContext("copy", 0);
06624 __Pyx_XDECREF(__pyx_r);
06625 __pyx_t_1 = __pyx_f_8PyClical_8clifford_copy(__pyx_v_self, 1); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 556, __pyx_L1_error)
06626 __Pyx_GOTREF(__pyx_t_1);
06627 __pyx_r = __pyx_t_1;
06628 __pyx_t_1 = 0;
06629 goto __pyx_L0;
06630
06631 /* function exit code */
06632 __pyx_L1_error++;
06633 __Pyx_XDECREF(__pyx_t_1);
06634 __Pyx_AddTraceback("PyClical.clifford.copy", __pyx_clineno, __pyx_lineno, __pyx_filename);
06635 __pyx_r = NULL;
06636 __pyx_L0++;
06637 __Pyx_XGIVEREF(__pyx_r);
06638 __Pyx_RefNannyFinishContext();
06639 return __pyx_r;
06640 }
06641
06642 /* "PyClical.pyx":565
06643 * return clifford(self)
06644 *
06645 * def __cinit__(self, other = 0, ixt = None): # ««««««««
06646 * """
06647 * Construct an object of type clifford.
06648 */
06649
06650 /* Python wrapper */
06651 static int __pyx_pw_8PyClical_8clifford_3__cinit__(PyObject *__pyx_v_self, PyObject
*__pyx_args, PyObject *__pyx_kwds); /*proto*/
06652 static int __pyx_pw_8PyClical_8clifford_3__cinit__(PyObject *__pyx_v_self, PyObject
*__pyx_args, PyObject *__pyx_kwds) {
06653 PyObject *__pyx_v_other = 0;
06654 PyObject *__pyx_v_ixt = 0;
06655 int __pyx_lineno = 0;
06656 const char *__pyx_filename = NULL;
06657 int __pyx_clineno = 0;
06658 int __pyx_r;
06659 __Pyx_RefNannyDeclarations
06660 __Pyx_RefNannySetupContext("__cinit__ (wrapper)", 0);
06661 {
06662 static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_other,&__pyx_n_s_ixt,0};
06663 PyObject* values[2] = {0,0};
06664 values[0] = ((PyObject *)__pyx_int_0);
06665 values[1] = ((PyObject *)Py_None);

```

```

06666 if (unlikely(__pyx_kwds)) {
06667 Py_ssize_t kw_args;
06668 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
06669 switch (pos_args) {
06670 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
06671 CYTHON_FALLTHROUGH;
06672 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
06673 CYTHON_FALLTHROUGH;
06674 case 0: break;
06675 default: goto __pyx_L5_argtuple_error;
06676 }
06677 kw_args = PyDict_Size(__pyx_kwds);
06678 switch (pos_args) {
06679 case 0:
06680 if (kw_args > 0) {
06681 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_other);
06682 if (value) { values[0] = value; kw_args--; }
06683 }
06684 CYTHON_FALLTHROUGH;
06685 case 1:
06686 if (kw_args > 0) {
06687 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_ixt);
06688 if (value) { values[1] = value; kw_args--; }
06689 }
06690 }
06691 if (unlikely(kw_args > 0)) {
06692 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values,
06693 pos_args, "__cinit__") < 0)) __PYX_ERR(0, 565, __pyx_L3_error)
06694 } else {
06695 switch (PyTuple_GET_SIZE(__pyx_args)) {
06696 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
06697 CYTHON_FALLTHROUGH;
06698 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
06699 CYTHON_FALLTHROUGH;
06700 case 0: break;
06701 default: goto __pyx_L5_argtuple_error;
06702 }
06703 }
06704 __pyx_v_other = values[0];
06705 __pyx_v_ixt = values[1];
06706 }
06707 goto __pyx_L4_argument_unpacking_done;
06708 __pyx_L5_argtuple_error:;
06709 __Pyx_RaiseArgtupleInvalid("__cinit__", 0, 0, 2, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0,
565, __pyx_L3_error)
06710 __pyx_L3_error:;
06711 __Pyx_AddTraceback("PyClical.clifford.__cinit__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
06712 __Pyx_RefNannyFinishContext();
06713 return -1;
06714 __pyx_L4_argument_unpacking_done:;
06715 __pyx_r = __pyx_pf_8PyClical_8clifford_2__cinit__((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self), __pyx_v_other, __pyx_v_ixt);
06716
06717 /* function exit code */
06718 __Pyx_RefNannyFinishContext();
06719 return __pyx_r;
06720 }
06721
06722 static int __pyx_pf_8PyClical_8clifford_2__cinit__(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self, PyObject *__pyx_v_other, PyObject *__pyx_v_ixt) {
06723 PyObject *__pyx_v_error_msg_prefix = NULL;
06724 PyObject *__pyx_v_bother = NULL;
06725 PyObject *__pyx_v_err = NULL;
06726 int __pyx_r;
06727 __Pyx_RefNannyDeclarations
06728 int __pyx_t_1;
06729 int __pyx_t_2;
06730 PyObject *__pyx_t_3 = NULL;
06731 PyObject *__pyx_t_4 = NULL;
06732 PyObject *__pyx_t_5 = NULL;
06733 Clifford *__pyx_t_6;
06734 PyObject *__pyx_t_7 = NULL;
06735 PyObject *__pyx_t_8 = NULL;
06736 scalar_t __pyx_t_9;
06737 PyObject *__pyx_t_10 = NULL;
06738 PyObject *__pyx_t_11 = NULL;
06739 PyObject *__pyx_t_12 = NULL;
06740 PyObject *__pyx_t_13 = NULL;
06741 char *__pyx_t_14;
06742 int __pyx_t_15;
06743 PyObject *__pyx_t_16 = NULL;
06744 PyObject *__pyx_t_17 = NULL;
06745 PyObject *__pyx_t_18 = NULL;
06746 int __pyx_t_19;
06747 char const *__pyx_t_20;

```

```

06748 PyObject *__pyx_t_21 = NULL;
06749 PyObject *__pyx_t_22 = NULL;
06750 PyObject *__pyx_t_23 = NULL;
06751 int __pyx_lineno = 0;
06752 const char *__pyx_filename = NULL;
06753 int __pyx_clineno = 0;
06754 __Pyx_RefNannySetupContext("__cinit__", 0);
06755
06756 /* "PyClical.pyx":588
06757 * 2{1}+3{2}
06758 * """
06759 * error_msg_prefix = "Cannot initialize clifford object from" # ««««««««
06760 * if ixt is None:
06761 * try:
06762 */
06763 __Pyx_INCREF(__pyx_kp_u_Cannot_initialize_clifford_objec);
06764 __pyx_v_error_msg_prefix = __pyx_kp_u_Cannot_initialize_clifford_objec;
06765
06766 /* "PyClical.pyx":589
06767 * """
06768 * error_msg_prefix = "Cannot initialize clifford object from"
06769 * if ixt is None: # ««««««««
06770 * try:
06771 * if isinstance(other, clifford):
06772 */
06773 __pyx_t_1 = (__pyx_v_ixt == Py_None);
06774 __pyx_t_2 = (__pyx_t_1 != 0);
06775 if (__pyx_t_2) {
06776
06777 /* "PyClical.pyx":590
06778 * error_msg_prefix = "Cannot initialize clifford object from"
06779 * if ixt is None:
06780 * try: # ««««««««
06781 * if isinstance(other, clifford):
06782 * self.instance = new Clifford((<clifford>other).unwrap())
06783 */
06784 {
06785 __Pyx_PyThreadState_declare
06786 __Pyx_PyThreadState_assign
06787 __Pyx_ExceptionSave(&__pyx_t_3, &__pyx_t_4, &__pyx_t_5);
06788 __Pyx_XGOTREF(__pyx_t_3);
06789 __Pyx_XGOTREF(__pyx_t_4);
06790 __Pyx_XGOTREF(__pyx_t_5);
06791 /*try:*/ {
06792
06793 /* "PyClical.pyx":591
06794 * if ixt is None:
06795 * try:
06796 * if isinstance(other, clifford): # ««««««««
06797 * self.instance = new Clifford((<clifford>other).unwrap())
06798 * elif isinstance(other, index_set):
06799 */
06800 __pyx_t_2 = __Pyx_TypeCheck(__pyx_v_other, __pyx_ptype_8PyClical_clifford);
06801 __pyx_t_1 = (__pyx_t_2 != 0);
06802 if (__pyx_t_1) {
06803
06804 /* "PyClical.pyx":592
06805 * try:
06806 * if isinstance(other, clifford):
06807 * self.instance = new Clifford((<clifford>other).unwrap()) # ««««««««
06808 * elif isinstance(other, index_set):
06809 * self.instance = new Clifford((<index_set>other).unwrap(), <scalar_t>1.0)
06810 */
06811 try {
06812 __pyx_t_6 = new Clifford(__pyx_f_8PyClical_8clifford_unwrap(((struct
__pyx_obj_8PyClical_clifford *)__pyx_v_other)));
06813 } catch (...) {
06814 __Pyx_CppExn2PyErr();
06815 __PYX_ERR(0, 592, __pyx_L4_error)
06816 }
06817 __pyx_v_self->instance = __pyx_t_6;
06818
06819 /* "PyClical.pyx":591
06820 * if ixt is None:
06821 * try:
06822 * if isinstance(other, clifford): # ««««««««
06823 * self.instance = new Clifford((<clifford>other).unwrap())
06824 * elif isinstance(other, index_set):
06825 */
06826 goto __pyx_L10;
06827 }
06828
06829 /* "PyClical.pyx":593
06830 * if isinstance(other, clifford):
06831 * self.instance = new Clifford((<clifford>other).unwrap())
06832 * elif isinstance(other, index_set): # ««««««««
06833 * self.instance = new Clifford((<index_set>other).unwrap(), <scalar_t>1.0)

```

```

06834 * elif isinstance(other, numbers.Real):
06835 */
06836 __pyx_t_1 = __Pyx_TypeCheck(__pyx_v_other, __pyx_ptype_8PyClical_index_set);
06837 __pyx_t_2 = (__pyx_t_1 != 0);
06838 if (__pyx_t_2) {
06839
06840 /* "PyClical.pyx":594
06841 * self.instance = new Clifford(<clifford>other).unwrap()
06842 * elif isinstance(other, index_set):
06843 * self.instance = new Clifford(<index_set>other).unwrap(), <scalar_t>1.0)
06844 # ««««««««
06845 * elif isinstance(other, numbers.Real):
06846 * self.instance = new Clifford(<scalar_t>other)
06847 */
06848 try {
06849 __pyx_t_6 = new Clifford(__pyx_f_8PyClical_9index_set_unwrap(((struct
__pyx_obj_8PyClical_index_set *)__pyx_v_other)), ((scalar_t)1.0));
06849 } catch(...) {
06850 __Pyx_CppExn2PyErr();
06851 __PYX_ERR(0, 594, __pyx_L4_error)
06852 }
06853 __pyx_v_self->instance = __pyx_t_6;
06854
06855 /* "PyClical.pyx":593
06856 * if isinstance(other, clifford):
06857 * self.instance = new Clifford(<clifford>other).unwrap()
06858 * elif isinstance(other, index_set):
06859 * self.instance = new Clifford(<index_set>other).unwrap(), <scalar_t>1.0)
06860 * elif isinstance(other, numbers.Real):
06861 */
06862 goto __pyx_L10;
06863 }
06864
06865 /* "PyClical.pyx":595
06866 * elif isinstance(other, index_set):
06867 * self.instance = new Clifford(<index_set>other).unwrap(), <scalar_t>1.0)
06868 * elif isinstance(other, numbers.Real):
06869 * self.instance = new Clifford(<scalar_t>other)
06870 * elif isinstance(other, str):
06871 */
06872 __Pyx_GetModuleGlobalName(__pyx_t_7, __pyx_n_s_numbers); if (unlikely(!__pyx_t_7))
__PYX_ERR(0, 595, __pyx_L4_error)
06873 __Pyx_GOTREF(__pyx_t_7);
06874 __pyx_t_8 = __Pyx_PyObject_GetAttrStr(__pyx_t_7, __pyx_n_s_Real); if
(unlikely(!__pyx_t_8)) __PYX_ERR(0, 595, __pyx_L4_error)
06875 __Pyx_GOTREF(__pyx_t_8);
06876 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
06877 __pyx_t_2 = PyObject_IsInstance(__pyx_v_other, __pyx_t_8); if (unlikely(__pyx_t_2 ==
((int)-1))) __PYX_ERR(0, 595, __pyx_L4_error)
06878 __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
06879 __pyx_t_1 = (__pyx_t_2 != 0);
06880 if (__pyx_t_1) {
06881
06882 /* "PyClical.pyx":596
06883 * self.instance = new Clifford(<index_set>other).unwrap(), <scalar_t>1.0)
06884 * elif isinstance(other, numbers.Real):
06885 * self.instance = new Clifford(<scalar_t>other)
06886 * elif isinstance(other, str):
06887 * try:
06888 */
06889 __pyx_t_9 = __pyx_PyFloat_AsDouble(__pyx_v_other); if (unlikely((__pyx_t_9 ==
((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 596, __pyx_L4_error)
06890 try {
06891 __pyx_t_6 = new Clifford(((scalar_t)__pyx_t_9));
06892 } catch(...) {
06893 __Pyx_CppExn2PyErr();
06894 __PYX_ERR(0, 596, __pyx_L4_error)
06895 }
06896 __pyx_v_self->instance = __pyx_t_6;
06897
06898 /* "PyClical.pyx":595
06899 * elif isinstance(other, index_set):
06900 * self.instance = new Clifford(<index_set>other).unwrap(), <scalar_t>1.0)
06901 * elif isinstance(other, numbers.Real):
06902 * self.instance = new Clifford(<scalar_t>other)
06903 * elif isinstance(other, str):
06904 */
06905 goto __pyx_L10;
06906 }
06907
06908 /* "PyClical.pyx":597
06909 * elif isinstance(other, numbers.Real):
06910 * self.instance = new Clifford(<scalar_t>other)
06911 * elif isinstance(other, str):
06912 * try:
06913 * bother = other.encode("UTF-8")
06914 */

```

```

06915 __pyx_t_1 = PyUnicode_Check(__pyx_v_other);
06916 __pyx_t_2 = (__pyx_t_1 != 0);
06917 if (likely(__pyx_t_2)) {
06918
06919 /* "PyClical.pyx":598
06920 * self.instance = new Clifford(<scalar_t>other)
06921 * elif isinstance(other, str):
06922 * try:
06923 * # ««««««««
06924 * bother = other.encode("UTF-8")
06925 * self.instance = new Clifford(<char *>bother)
06926 */
06927 {
06928 __Pyx_PyThreadState_declare
06929 __Pyx_PyThreadState_assign
06930 __Pyx_ExceptionSave(&__pyx_t_10, &__pyx_t_11, &__pyx_t_12);
06931 __Pyx_XGOTREF(__pyx_t_10);
06932 __Pyx_XGOTREF(__pyx_t_11);
06933 __Pyx_XGOTREF(__pyx_t_12);
06934 /*try:*/ {
06935
06936 /* "PyClical.pyx":599
06937 * elif isinstance(other, str):
06938 * try:
06939 * bother = other.encode("UTF-8")
06940 * self.instance = new Clifford(<char *>bother)
06941 * except RuntimeError:
06942 */
06943 __pyx_t_7 = __Pyx_PyObject_GetAttrStr(__pyx_v_other, __pyx_n_s_encode); if
(unlikely(!__pyx_t_7)) __PYX_ERR(0, 599, __pyx_L11_error)
06944 __Pyx_GOTREF(__pyx_t_7);
06945 __pyx_t_13 = NULL;
06946 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_7))) {
06947 __pyx_t_13 = PyMethod_GET_SELF(__pyx_t_7);
06948 if (likely(__pyx_t_13)) {
06949 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_7);
06950 __Pyx_INCREF(__pyx_t_13);
06951 __Pyx_INCREF(function);
06952 __Pyx_DECREF_SET(__pyx_t_7, function);
06953 }
06954 }
06955 __pyx_t_8 = (__pyx_t_13) ? __Pyx_PyObject_Call2Args(__pyx_t_7, __pyx_t_13,
__pyx_kp_u_UTF_8) : __Pyx_PyObject_CallOneArg(__pyx_t_7, __pyx_kp_u_UTF_8);
06956 __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
06957 if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 599, __pyx_L11_error)
06958 __Pyx_GOTREF(__pyx_t_8);
06959 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
06960 __pyx_v_bother = __pyx_t_8;
06961 __pyx_t_8 = 0;
06962
06963 /* "PyClical.pyx":600
06964 * try:
06965 * bother = other.encode("UTF-8")
06966 * self.instance = new Clifford(<char *>bother)
06967 * except RuntimeError:
06968 * raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
06969 */
06970 __pyx_t_14 = __Pyx_PyObject_AsWritableString(__pyx_v_bother); if
(unlikely((!__pyx_t_14) && PyErr_Occurred())) __PYX_ERR(0, 600, __pyx_L11_error)
06971 try {
06972 __pyx_t_6 = new Clifford(((char *)__pyx_t_14));
06973 } catch (...) {
06974 __Pyx_CppExn2PyErr();
06975 __PYX_ERR(0, 600, __pyx_L11_error)
06976 }
06977 __pyx_v_self->instance = __pyx_t_6;
06978
06979 /* "PyClical.pyx":598
06980 * self.instance = new Clifford(<scalar_t>other)
06981 * elif isinstance(other, str):
06982 * try:
06983 * # ««««««««
06984 * bother = other.encode("UTF-8")
06985 * self.instance = new Clifford(<char *>bother)
06986 */
06987 {
06988 __Pyx_XDECREF(__pyx_t_10); __pyx_t_10 = 0;
06989 __Pyx_XDECREF(__pyx_t_11); __pyx_t_11 = 0;
06990 __Pyx_XDECREF(__pyx_t_12); __pyx_t_12 = 0;
06991 goto __pyx_L16_try_end;
06992 }
06993 __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
06994 __Pyx_XDECREF(__pyx_t_7); __pyx_t_7 = 0;
06995 __Pyx_XDECREF(__pyx_t_8); __pyx_t_8 = 0;
06996
06997 /* "PyClical.pyx":601
06998 * bother = other.encode("UTF-8")
06999 * self.instance = new Clifford(<char *>bother)
07000 * except RuntimeError:
07001 * # ««««««««

```

```

06999 * raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
07000 * else:
07001 */
07002 __pyx_t_15 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_RuntimeError);
07003 if (__pyx_t_15) {
07004 __Pyx_AddTraceback("PyClical.clifford.__cinit__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
07005 if (__Pyx_GetException(&__pyx_t_8, &__pyx_t_7, &__pyx_t_13) < 0) __PYX_ERR(0,
601, __pyx_L13_except_error)
07006 __Pyx_GOTREF(__pyx_t_8);
07007 __Pyx_GOTREF(__pyx_t_7);
07008 __Pyx_GOTREF(__pyx_t_13);
07009
07010 /* "PyClical.pyx":602
07011 self.instance = new Clifford(<char *>bother)
07012 except RuntimeError:
07013 raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
07014 # ««««««««
07015 else:
07016 raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
07017 */
07018 __pyx_t_16 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix,
__pyx_kp_u_invalid_string); if (unlikely(!__pyx_t_16)) __PYX_ERR(0, 602, __pyx_L13_except_error)
07019 __Pyx_GOTREF(__pyx_t_16);
07020 __pyx_t_17 = PyObject_Repr(__pyx_v_other); if (unlikely(!__pyx_t_17))
__PYX_ERR(0, 602, __pyx_L13_except_error)
07021 __Pyx_GOTREF(__pyx_t_17);
07022 __pyx_t_18 = PyNumber_Add(__pyx_t_16, __pyx_t_17); if (unlikely(!__pyx_t_18))
__PYX_ERR(0, 602, __pyx_L13_except_error)
07023 __Pyx_GOTREF(__pyx_t_18);
07024 __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
07025 __Pyx_DECREF(__pyx_t_17); __pyx_t_17 = 0;
07026 __pyx_t_17 = PyNumber_Add(__pyx_t_18, __pyx_kp_u); if (unlikely(!__pyx_t_17))
__PYX_ERR(0, 602, __pyx_L13_except_error)
07027 __Pyx_GOTREF(__pyx_t_17);
07028 __Pyx_DECREF(__pyx_t_18); __pyx_t_18 = 0;
07029 __pyx_t_18 = __Pyx_PyObject_CallOneArg(__pyx_builtin_ValueError, __pyx_t_17); if
(unlikely(!__pyx_t_18)) __PYX_ERR(0, 602, __pyx_L13_except_error)
07030 __Pyx_GOTREF(__pyx_t_18);
07031 __Pyx_DECREF(__pyx_t_17); __pyx_t_17 = 0;
07032 __Pyx_Raise(__pyx_t_18, 0, 0, 0);
07033 __Pyx_DECREF(__pyx_t_18); __pyx_t_18 = 0;
07034 __PYX_ERR(0, 602, __pyx_L13_except_error)
07035 }
07036 goto __pyx_L13_except_error;
07037 __pyx_L13_except_error:;
07038
07039 /* "PyClical.pyx":598
07040 self.instance = new Clifford(<scalar_t>other)
07041 elif isinstance(other, str):
07042 try:
07043 # ««««««««
07044 bother = other.encode("UTF-8")
07045 self.instance = new Clifford(<char *>bother)
07046 */
07047 __Pyx_XGIVEREF(__pyx_t_10);
07048 __Pyx_XGIVEREF(__pyx_t_11);
07049 __Pyx_XGIVEREF(__pyx_t_12);
07050 __Pyx_ExceptionReset(__pyx_t_10, __pyx_t_11, __pyx_t_12);
07051 goto __pyx_L4_error;
07052 __pyx_L16_try_end:;
07053 }
07054
07055 /* "PyClical.pyx":597
07056 elif isinstance(other, numbers.Real):
07057 self.instance = new Clifford(<scalar_t>other)
07058 elif isinstance(other, str):
07059 # ««««««««
07060 try:
07061 bother = other.encode("UTF-8")
07062 */
07063 goto __pyx_L10;
07064 }
07065
07066 /* "PyClical.pyx":604
07067 raise ValueError(error_msg_prefix + " invalid string " + repr(other) + ".")
07068 else:
07069 raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
07070 #
07071 ««««««««
07072 except RuntimeError as err:
07073 raise ValueError(error_msg_prefix + " " + str(type(other))
07074 */
07075
07076 /*else*/ {
07077 __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_2); if
(unlikely(!__pyx_t_13)) __PYX_ERR(0, 604, __pyx_L4_error)
07078 __Pyx_GOTREF(__pyx_t_13);
07079 __pyx_t_7 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject
*)Py_TYPE(__pyx_v_other))); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 604, __pyx_L4_error)
07080 __Pyx_GOTREF(__pyx_t_7);

```



```

07075 __pyx_t_8 = __Pyx_PyUnicode_Concat(__pyx_t_13, __pyx_t_7); if (unlikely(!__pyx_t_8))
__PYX_ERR(0, 604, __pyx_L4_error)
07076 __Pyx_GOTREF(__pyx_t_8);
07077 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
07078 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
07079 __pyx_t_7 = __Pyx_PyUnicode_Concat(__pyx_t_8, __pyx_kp_u_); if
(unlikely(!__pyx_t_7)) __PYX_ERR(0, 604, __pyx_L4_error)
07080 __Pyx_GOTREF(__pyx_t_7);
07081 __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
07082 __pyx_t_8 = __PyxPyObject_CallOneArg(__pyx_builtin_TypeError, __pyx_t_7); if
(unlikely(!__pyx_t_8)) __PYX_ERR(0, 604, __pyx_L4_error)
07083 __Pyx_GOTREF(__pyx_t_8);
07084 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
07085 __Pyx_Raise(__pyx_t_8, 0, 0, 0);
07086 __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
07087 __PYX_ERR(0, 604, __pyx_L4_error)
07088 }
07089 __pyx_L10::
07090
07091 /* "PyClical.pyx":590
07092 * error_msg_prefix = "Cannot initialize clifford object from"
07093 * if ixt is None:
07094 * try:
07095 * if isinstance(other, clifford):
07096 * self.instance = new Clifford(<clifford>other).unwrap()
07097 */
07098 }
07099 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
07100 __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
07101 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
07102 goto __pyx_L9_try_end;
07103 __pyx_L4_error:;
07104 __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
07105 __Pyx_XDECREF(__pyx_t_16); __pyx_t_16 = 0;
07106 __Pyx_XDECREF(__pyx_t_17); __pyx_t_17 = 0;
07107 __Pyx_XDECREF(__pyx_t_18); __pyx_t_18 = 0;
07108 __Pyx_XDECREF(__pyx_t_7); __pyx_t_7 = 0;
07109 __Pyx_XDECREF(__pyx_t_8); __pyx_t_8 = 0;
07110
07111 /* "PyClical.pyx":605
07112 * else:
07113 * raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
07114 * except RuntimeError as err:
07115 * raise ValueError(error_msg_prefix + " " + str(type(other))
07116 * + " value " + repr(other) + ":")
07117 */
07118 __pyx_t_15 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_RuntimeError);
07119 if (__pyx_t_15) {
07120 __Pyx_AddTraceback("PyClical.clifford.__cinit__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
07121 if (__Pyx_GetException(&__pyx_t_8, &__pyx_t_7, &__pyx_t_13) < 0) __PYX_ERR(0, 605,
__pyx_L6_except_error)
07122 __Pyx_GOTREF(__pyx_t_8);
07123 __Pyx_GOTREF(__pyx_t_7);
07124 __Pyx_GOTREF(__pyx_t_13);
07125 __Pyx_INCREF(__pyx_t_7);
07126 __pyx_v_err = __pyx_t_7;
07127 /*try:*/ {
07128
07129 /* "PyClical.pyx":606
07130 * raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
07131 * except RuntimeError as err:
07132 * raise ValueError(error_msg_prefix + " " + str(type(other))
07133 * + " value " + repr(other) + ":")
07134 * + "\n\t" + str(err))
07135 */
07136 __pyx_t_18 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_2); if
(unlikely(!__pyx_t_18)) __PYX_ERR(0, 606, __pyx_L24_error)
07137 __Pyx_GOTREF(__pyx_t_18);
07138 __pyx_t_17 = __PyxPyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject
*)Py_TYPE(__pyx_v_other))); if (unlikely(!__pyx_t_17)) __PYX_ERR(0, 606, __pyx_L24_error)
07139 __Pyx_GOTREF(__pyx_t_17);
07140 __pyx_t_16 = __Pyx_PyUnicode_Concat(__pyx_t_18, __pyx_t_17); if
(unlikely(!__pyx_t_16)) __PYX_ERR(0, 606, __pyx_L24_error)
07141 __Pyx_GOTREF(__pyx_t_16);
07142 __Pyx_DECREF(__pyx_t_18); __pyx_t_18 = 0;
07143 __Pyx_DECREF(__pyx_t_17); __pyx_t_17 = 0;
07144
07145 /* "PyClical.pyx":607
07146 * except RuntimeError as err:
07147 * raise ValueError(error_msg_prefix + " " + str(type(other))
07148 * + " value " + repr(other) + ":")
07149 * + "\n\t" + str(err))
07150 * elif isinstance(ixt, index_set):
07151 */
07152 __pyx_t_17 = __Pyx_PyUnicode_Concat(__pyx_t_16, __pyx_kp_u_value); if

```

```

 (unlikely(!__pyx_t_17)) __PYX_ERR(0, 607, __pyx_L24_error)
07153 __Pyx_GOTREF(__pyx_t_17);
07154 __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
07155 __pyx_t_16 = PyObject_Repr(__pyx_v_other); if (unlikely(!__pyx_t_16)) __PYX_ERR(0,
607, __pyx_L24_error)
07156 __Pyx_GOTREF(__pyx_t_16);
07157 __pyx_t_18 = PyNumber_Add(__pyx_t_17, __pyx_t_16); if (unlikely(!__pyx_t_18))
__PYX_ERR(0, 607, __pyx_L24_error)
07158 __Pyx_GOTREF(__pyx_t_18);
07159 __Pyx_DECREF(__pyx_t_17); __pyx_t_17 = 0;
07160 __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
07161 __pyx_t_16 = PyNumber_Add(__pyx_t_18, __pyx_kp_u_5); if (unlikely(!__pyx_t_16))
__PYX_ERR(0, 607, __pyx_L24_error)
07162 __Pyx_GOTREF(__pyx_t_16);
07163 __Pyx_DECREF(__pyx_t_18); __pyx_t_18 = 0;
07164
07165 /* "PyCliclcal.pyx":608
07166 * raise ValueError(error_msg_prefix + " " + str(type(other))
07167 * + " value " + repr(other) + ":"
07168 * + "\n\t" + str(err)) # ««««««««
07169 * elif isinstance(ixt, index_set):
07170 * if isinstance(other, numbers.Real):
07171 */
07172 __pyx_t_18 = PyNumber_Add(__pyx_t_16, __pyx_kp_u_6); if (unlikely(!__pyx_t_18))
__PYX_ERR(0, 608, __pyx_L24_error)
07173 __Pyx_GOTREF(__pyx_t_18);
07174 __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
07175 __pyx_t_16 = __Pyx_PyObject_CallOneArg((PyObject *)(&PyUnicode_Type)),
__pyx_v_err); if (unlikely(!__pyx_t_16)) __PYX_ERR(0, 608, __pyx_L24_error)
07176 __Pyx_GOTREF(__pyx_t_16);
07177 __pyx_t_17 = PyNumber_Add(__pyx_t_18, __pyx_t_16); if (unlikely(!__pyx_t_17))
__PYX_ERR(0, 608, __pyx_L24_error)
07178 __Pyx_GOTREF(__pyx_t_17);
07179 __Pyx_DECREF(__pyx_t_18); __pyx_t_18 = 0;
07180 __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
07181
07182 /* "PyCliclcal.pyx":606
07183 * raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
07184 * except RuntimeError as err:
07185 * raise ValueError(error_msg_prefix + " " + str(type(other)) # ««««««««
07186 * + " value " + repr(other) + ":"
07187 * + "\n\t" + str(err))
07188 */
07189 __pyx_t_16 = __Pyx_PyObject_CallOneArg(__pyx_builtin_ValueError, __pyx_t_17); if
(unlikely(!__pyx_t_16)) __PYX_ERR(0, 606, __pyx_L24_error)
07190 __Pyx_GOTREF(__pyx_t_16);
07191 __Pyx_DECREF(__pyx_t_17); __pyx_t_17 = 0;
07192 __Pyx_Raise(__pyx_t_16, 0, 0, 0);
07193 __Pyx_DECREF(__pyx_t_16); __pyx_t_16 = 0;
07194 __PYX_ERR(0, 606, __pyx_L24_error)
07195 }
07196
07197 /* "PyCliclcal.pyx":605
07198 * else:
07199 * raise TypeError(error_msg_prefix + " " + str(type(other)) + ".")
07200 * except RuntimeError as err: # ««««««««
07201 * raise ValueError(error_msg_prefix + " " + str(type(other))
07202 * + " value " + repr(other) + ":"
07203 */
07204 /*finally:*/ {
07205 __pyx_L24_error:;
07206 /*exception exit:*/{
07207 __Pyx_PyThreadState_declare
07208 __Pyx_PyThreadState_assign
07209 __pyx_t_12 = 0; __pyx_t_11 = 0; __pyx_t_10 = 0; __pyx_t_21 = 0; __pyx_t_22 = 0;
__pyx_t_23 = 0;
07210 __Pyx_XDECREF(__pyx_t_16); __pyx_t_16 = 0;
07211 __Pyx_XDECREF(__pyx_t_17); __pyx_t_17 = 0;
07212 __Pyx_XDECREF(__pyx_t_18); __pyx_t_18 = 0;
07213 if (PY_MAJOR_VERSION >= 3) __Pyx_ExceptionSwap(&__pyx_t_21, &__pyx_t_22,
&__pyx_t_23);
07214 if ((PY_MAJOR_VERSION < 3) || unlikely(__Pyx_GetException(&__pyx_t_12,
&__pyx_t_11, &__pyx_t_10) < 0)) __Pyx_ErrFetch(&__pyx_t_12, &__pyx_t_11, &__pyx_t_10);
07215 __Pyx_XGOTREF(__pyx_t_12);
07216 __Pyx_XGOTREF(__pyx_t_11);
07217 __Pyx_XGOTREF(__pyx_t_10);
07218 __Pyx_XGOTREF(__pyx_t_21);
07219 __Pyx_XGOTREF(__pyx_t_22);
07220 __Pyx_XGOTREF(__pyx_t_23);
07221 __pyx_t_15 = __pyx_lineno; __pyx_t_19 = __pyx_clineno; __pyx_t_20 =
__pyx_filename;
07222 {
07223 __Pyx_DECREF(__pyx_v_err);
07224 __pyx_v_err = NULL;
07225 }
07226 if (PY_MAJOR_VERSION >= 3) {
07227 __Pyx_XGIVEREF(__pyx_t_21);

```

```

07228 __Pyx_XGIVEREF(__pyx_t_22);
07229 __Pyx_XGIVEREF(__pyx_t_23);
07230 __Pyx_ExceptionReset(__pyx_t_21, __pyx_t_22, __pyx_t_23);
07231 }
07232 __Pyx_XGIVEREF(__pyx_t_12);
07233 __Pyx_XGIVEREF(__pyx_t_11);
07234 __Pyx_XGIVEREF(__pyx_t_10);
07235 __Pyx_ErrRestore(__pyx_t_12, __pyx_t_11, __pyx_t_10);
07236 __pyx_t_12 = 0; __pyx_t_11 = 0; __pyx_t_10 = 0; __pyx_t_21 = 0; __pyx_t_22 = 0;
__pyx_t_23 = 0;
07237 __pyx_lineno = __pyx_t_15; __pyx_clineno = __pyx_t_19; __pyx_filename =
__pyx_t_20;
07238 goto __pyx_L6_except_error;
07239 }
07240 }
07241 }
07242 goto __pyx_L6_except_error;
07243 __pyx_L6_except_error;;
07244
07245 /* "PyClical.pyx":590
07246 * error_msg_prefix = "Cannot initialize clifford object from"
07247 * if ixt is None:
07248 * try:
07249 * # ««««««««
07250 * if isinstance(other, clifford):
07251 * self.instance = new Clifford((<clifford>other).unwrap())
07252 */
07252 __Pyx_XGIVEREF(__pyx_t_3);
07253 __Pyx_XGIVEREF(__pyx_t_4);
07254 __Pyx_XGIVEREF(__pyx_t_5);
07255 __Pyx_ExceptionReset(__pyx_t_3, __pyx_t_4, __pyx_t_5);
07256 goto __pyx_L1_error;
07257 __pyx_L9_try_end;;
07258 }
07259
07260 /* "PyClical.pyx":589
07261 * """
07262 * error_msg_prefix = "Cannot initialize clifford object from"
07263 * if ixt is None:
07264 * try:
07265 * # ««««««««
07266 * if isinstance(other, clifford):
07267 */
07267 goto __pyx_L3;
07268 }
07269
07270 /* "PyClical.pyx":609
07271 *
07272 * + " value " + repr(other) + ":"
07273 * + "\n\t" + str(err))
07274 * # ««««««««
07275 * elif isinstance(ixt, index_set):
07276 * if isinstance(other, numbers.Real):
07277 * self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
07278 */
07278 __pyx_t_2 = __Pyx_TypeCheck(__pyx_v_ixt, __pyx_ptype_8PyClical_index_set);
07279 __pyx_t_1 = (__pyx_t_2 != 0);
07280 if (likely(__pyx_t_1)) {
07281 /* "PyClical.pyx":610
07282 *
07283 * + "\n\t" + str(err))
07284 * elif isinstance(ixt, index_set):
07285 * if isinstance(other, numbers.Real):
07286 * self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
07287 * elif isinstance(other, collections.abc.Sequence):
07288 */
07288 __Pyx_GetModuleGlobalName(__pyx_t_13, __pyx_n_s_numbers); if (unlikely(!__pyx_t_13))
__PYX_ERR(0, 610, __pyx_L1_error)
07289 __Pyx_GOTREF(__pyx_t_13);
07290 __pyx_t_7 = __Pyx_PyObject_GetAttrStr(__pyx_t_13, __pyx_n_s_Real); if
(unlikely(!__pyx_t_7)) __PYX_ERR(0, 610, __pyx_L1_error)
07291 __Pyx_GOTREF(__pyx_t_7);
07292 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
07293 __pyx_t_1 = PyObject_IsInstance(__pyx_v_other, __pyx_t_7); if (unlikely(__pyx_t_1 ==
((int)-1))) __PYX_ERR(0, 610, __pyx_L1_error)
07294 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
07295 __pyx_t_2 = (__pyx_t_1 != 0);
07296 if (__pyx_t_2) {
07297
07298 /* "PyClical.pyx":611
07299 * elif isinstance(ixt, index_set):
07300 * if isinstance(other, numbers.Real):
07301 * self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
07302 *
07303 * # ««««««««
07304 * elif isinstance(other, collections.abc.Sequence):
07305 * self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
07306 */
07306 __pyx_t_9 = __pyx_PyFloat_AsDouble(__pyx_v_other); if (unlikely((__pyx_t_9 ==
((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 611, __pyx_L1_error)
07307 try {
07308 __pyx_t_6 = new Clifford(__pyx_f_8PyClical_9index_set_unwrap((struct

```

```

__pyx_obj_8PyClical_index_set *)__pyx_v_ixt)), ((scalar_t)__pyx_t_9));
77308 } catch(...) {
77309 __Pyx_CppExn2PyErr();
77310 __PYX_ERR(0, 611, __pyx_L1_error)
77311 }
77312 __pyx_v_self->instance = __pyx_t_6;
77313
77314 /* "PyClical.pyx":610
77315 *
77316 * elif isinstance(ixt, index_set):
77317 * if isinstance(other, numbers.Real):
77318 * self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
77319 * elif isinstance(other, collections.abc.Sequence):
77320 */
77321 goto __pyx_L30;
77322 }
77323
77324 /* "PyClical.pyx":612
77325 * if isinstance(other, numbers.Real):
77326 * self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
77327 * elif isinstance(other, collections.abc.Sequence):
77328 * self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
77329 * else:
77330 */
77331 __Pyx_GetModuleGlobalName(__pyx_t_7, __pyx_n_s_collections); if (unlikely(!__pyx_t_7))
__PYX_ERR(0, 612, __pyx_L1_error)
77332 __Pyx_GOTREF(__pyx_t_7);
77333 __pyx_t_13 = __Pyx_PyObject_GetAttrStr(__pyx_t_7, __pyx_n_s_abc); if
(unlikely(!__pyx_t_13)) __PYX_ERR(0, 612, __pyx_L1_error)
77334 __Pyx_GOTREF(__pyx_t_13);
77335 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
77336 __pyx_t_7 = __Pyx_PyObject_GetAttrStr(__pyx_t_13, __pyx_n_s_Sequence); if
(unlikely(!__pyx_t_7)) __PYX_ERR(0, 612, __pyx_L1_error)
77337 __Pyx_GOTREF(__pyx_t_7);
77338 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
77339 __pyx_t_2 = PyObject_IsInstance(__pyx_v_other, __pyx_t_7); if (unlikely(__pyx_t_2 ==
((int)-1))) __PYX_ERR(0, 612, __pyx_L1_error)
77340 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
77341 __pyx_t_1 = (__pyx_t_2 != 0);
77342 if (likely(__pyx_t_1)) {
77343
77344 /* "PyClical.pyx":613
77345 * self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
77346 * elif isinstance(other, collections.abc.Sequence):
77347 * self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
77348 * else:
77349 * raise TypeError(error_msg_prefix + " (" + str(type(other))
77350 */
77351 try {
77352 __pyx_t_6 = new Clifford(__pyx_f_8PyClical_list_to_vector(__pyx_v_other),
__pyx_f_8PyClical_9index_set_unwrap(((struct __pyx_obj_8PyClical_index_set *)__pyx_v_ixt)));
77353 } catch(...) {
77354 __Pyx_CppExn2PyErr();
77355 __PYX_ERR(0, 613, __pyx_L1_error)
77356 }
77357 __pyx_v_self->instance = __pyx_t_6;
77358
77359 /* "PyClical.pyx":612
77360 * if isinstance(other, numbers.Real):
77361 * self.instance = new Clifford((<index_set>ixt).unwrap(), <scalar_t>other)
77362 * elif isinstance(other, collections.abc.Sequence):
77363 * self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
77364 * else:
77365 */
77366 goto __pyx_L30;
77367 }
77368
77369 /* "PyClical.pyx":615
77370 * self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
77371 * else:
77372 * raise TypeError(error_msg_prefix + " (" + str(type(other))
77373 * + ", " + repr(ixt) + ").")
77374 *
77375 */
77376 /*else*/ {
77377 __pyx_t_7 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_7); if
(unlikely(!__pyx_t_7)) __PYX_ERR(0, 615, __pyx_L1_error)
77378 __Pyx_GOTREF(__pyx_t_7);
77379 __pyx_t_13 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject
*)Py_TYPE(__pyx_v_other))); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 615, __pyx_L1_error)
77380 __Pyx_GOTREF(__pyx_t_13);
77381 __pyx_t_8 = __Pyx_PyUnicode_Concat(__pyx_t_7, __pyx_t_13); if (unlikely(!__pyx_t_8))
__PYX_ERR(0, 615, __pyx_L1_error)
77382 __Pyx_GOTREF(__pyx_t_8);
77383 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
77384 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;

```

```

07385
07386 /* "PyClical.pyx":616
07387 * else:
07388 * raise TypeError(error_msg_prefix + " (" + str(type(other))
07389 * + ", " + repr(ixt) + ").") # ««««««««
07390 * else:
07391 * raise TypeError(error_msg_prefix + " (" + str(type(other))
07392 */
07393 __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_t_8, __pyx_kp_u_8); if
(unlikely(!__pyx_t_13)) __PYX_ERR(0, 616, __pyx_L1_error)
07394 __Pyx_GOTREF(__pyx_t_13);
07395 __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
07396 __pyx_t_8 = PyObject_Repr(__pyx_v_ixt); if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 616,
__pyx_L1_error)
07397 __Pyx_GOTREF(__pyx_t_8);
07398 __pyx_t_7 = PyNumber_Add(__pyx_t_13, __pyx_t_8); if (unlikely(!__pyx_t_7)) __PYX_ERR(0,
616, __pyx_L1_error)
07399 __Pyx_GOTREF(__pyx_t_7);
07400 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
07401 __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
07402 __pyx_t_8 = PyNumber_Add(__pyx_t_7, __pyx_kp_u_9); if (unlikely(!__pyx_t_8))
__PYX_ERR(0, 616, __pyx_L1_error)
07403 __Pyx_GOTREF(__pyx_t_8);
07404 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
07405
07406 /* "PyClical.pyx":615
07407 * self.instance = new Clifford(list_to_vector(other), (<index_set>ixt).unwrap())
07408 * else:
07409 * raise TypeError(error_msg_prefix + " (" + str(type(other))
07410 * + ", " + repr(ixt) + ").") # ««««««««
07411 * else:
07412 */
07413 __pyx_t_7 = __Pyx_PyObject_CallOneArg(__pyx_builtin_TypeError, __pyx_t_8); if
(unlikely(!__pyx_t_7)) __PYX_ERR(0, 615, __pyx_L1_error)
07414 __Pyx_GOTREF(__pyx_t_7);
07415 __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
07416 __Pyx_Raise(__pyx_t_7, 0, 0, 0);
07417 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
07418 __PYX_ERR(0, 615, __pyx_L1_error)
07419 }
07420 __pyx_L30:;
07421
07422 /* "PyClical.pyx":609
07423 * + " value " + repr(other) + ":"
07424 * + "\n\t" + str(err))
07425 * elif isinstance(ixt, index_set): # ««««««««
07426 * if isinstance(other, numbers.Real):
07427 * self.instance = new Clifford(<index_set>ixt).unwrap(), <scalar_t>other)
07428 */
07429 goto __pyx_L3;
07430 }
07431
07432 /* "PyClical.pyx":618
07433 * + ", " + repr(ixt) + ").")
07434 * else:
07435 * raise TypeError(error_msg_prefix + " (" + str(type(other))
07436 * + ", " + str(type(ixt)) + ").") # ««««««««
07437 *
07438 */
07439 /*else*/ {
07440 __pyx_t_7 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_7); if
(unlikely(!__pyx_t_7)) __PYX_ERR(0, 618, __pyx_L1_error)
07441 __Pyx_GOTREF(__pyx_t_7);
07442 __pyx_t_8 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject
*)Py_TYPE(__pyx_v_other))); if (unlikely(!__pyx_t_8)) __PYX_ERR(0, 618, __pyx_L1_error)
07443 __Pyx_GOTREF(__pyx_t_8);
07444 __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_t_7, __pyx_t_8); if (unlikely(!__pyx_t_13))
__PYX_ERR(0, 618, __pyx_L1_error)
07445 __Pyx_GOTREF(__pyx_t_13);
07446 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
07447 __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
07448
07449 /* "PyClical.pyx":619
07450 * else:
07451 * raise TypeError(error_msg_prefix + " (" + str(type(other))
07452 * + ", " + str(type(ixt)) + ").") # ««««««««
07453 *
07454 * def __dealloc__(self):
07455 */
07456 __pyx_t_8 = __Pyx_PyUnicode_Concat(__pyx_t_13, __pyx_kp_u_8); if (unlikely(!__pyx_t_8))
__PYX_ERR(0, 619, __pyx_L1_error)
07457 __Pyx_GOTREF(__pyx_t_8);
07458 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
07459 __pyx_t_13 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject
*)Py_TYPE(__pyx_v_ixt))); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 619, __pyx_L1_error)
07460 __Pyx_GOTREF(__pyx_t_13);
07461 __pyx_t_7 = __Pyx_PyUnicode_Concat(__pyx_t_8, __pyx_t_13); if (unlikely(!__pyx_t_7))

```

```

__PYX_ERR(0, 619, __pyx_L1_error)
07462 __Pyx_GOTREF(__pyx_t_7);
07463 __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
07464 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
07465 __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_t_7, __pyx_kp_u_9); if (unlikely(!__pyx_t_13))
__PYX_ERR(0, 619, __pyx_L1_error)
07466 __Pyx_GOTREF(__pyx_t_13);
07467 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
07468
07469 /* "PyClical.pyx":618
07470 * + ", " + repr(ixt) + ").")
07471 * else:
07472 * raise TypeError(error_msg_prefix + " (" + str(type(other)) # ««««««««
07473 * + ", " + str(type(ixt)) + ").")
07474 *
07475 */
07476 __pyx_t_7 = __Pyx_PyObject_CallOneArg(__pyx_builtin_TypeError, __pyx_t_13); if
(unlikely(!__pyx_t_7)) __PYX_ERR(0, 618, __pyx_L1_error)
07477 __Pyx_GOTREF(__pyx_t_7);
07478 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
07479 __Pyx_Raise(__pyx_t_7, 0, 0, 0);
07480 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
07481 __PYX_ERR(0, 618, __pyx_L1_error)
07482 }
07483 __pyx_L3:;
07484
07485 /* "PyClical.pyx":565
07486 * return clifford(self)
07487 *
07488 * def __cinit__(self, other = 0, ixt = None): # ««««««««
07489 * """
07490 * Construct an object of type clifford.
07491 */
07492
07493 /* function exit code */
07494 __pyx_r = 0;
07495 goto __pyx_L0;
07496 __pyx_L1_error:;
07497 __Pyx_XDECREF(__pyx_t_7);
07498 __Pyx_XDECREF(__pyx_t_8);
07499 __Pyx_XDECREF(__pyx_t_13);
07500 __Pyx_XDECREF(__pyx_t_16);
07501 __Pyx_XDECREF(__pyx_t_17);
07502 __Pyx_XDECREF(__pyx_t_18);
07503 __Pyx_AddTraceback("PyClical.clifford.__cinit__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
07504 __pyx_r = -1;
07505 __pyx_L0:;
07506 __Pyx_XDECREF(__pyx_v_error_msg_prefix);
07507 __Pyx_XDECREF(__pyx_v_bother);
07508 __Pyx_XDECREF(__pyx_v_err);
07509 __Pyx_RefNannyFinishContext();
07510 return __pyx_r;
07511 }
07512
07513 /* "PyClical.pyx":621
07514 * + ", " + str(type(ixt)) + ").")
07515 *
07516 * def __dealloc__(self): # ««««««««
07517 * """
07518 * Clean up by deallocating the instance of C++ class Clifford.
07519 */
07520
07521 /* Python wrapper */
07522 static void __pyx_pw_8PyClical_8clifford_5__dealloc__(PyObject *__pyx_v_self); /*proto*/
07523 static void __pyx_pw_8PyClical_8clifford_5__dealloc__(PyObject *__pyx_v_self) {
07524 __Pyx_RefNannyDeclarations
07525 __Pyx_RefNannySetupContext("__dealloc__ (wrapper)", 0);
07526 __pyx_pf_8PyClical_8clifford_4__dealloc__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
07527
07528 /* function exit code */
07529 __Pyx_RefNannyFinishContext();
07530 }
07531
07532 static void __pyx_pf_8PyClical_8clifford_4__dealloc__(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
07533 __Pyx_RefNannyDeclarations
07534 __Pyx_RefNannySetupContext("__dealloc__", 0);
07535
07536 /* "PyClical.pyx":625
07537 * Clean up by deallocating the instance of C++ class Clifford.
07538 * """
07539 * del self.instance # ««««««««
07540 *
07541 * def __contains__(self, x):
07542 */

```

```

07543 delete __pyx_v_self->instance;
07544
07545 /* "PyClical.pyx":621
07546 * + ", " + str(type(ixt)) + ").")
07547 *
07548 * def __dealloc__(self): # ««««««««
07549 * """
07550 * Clean up by deallocating the instance of C++ class Clifford.
07551 */
07552
07553 /* function exit code */
07554 __Pyx_RefNannyFinishContext();
07555 }
07556
07557 /* "PyClical.pyx":627
07558 * del self.instance
07559 *
07560 * def __contains__(self, x): # ««««««««
07561 * """
07562 * Not applicable.
07563 */
07564
07565 /* Python wrapper */
07566 static int __pyx_pw_8PyClical_8clifford_7__contains__(PyObject *__pyx_v_self, PyObject
07567 *__pyx_v_x); /*proto*/
07567 static char __pyx_doc_8PyClical_8clifford_6__contains__[] = "\n Not applicable.\n\n
>> x=clifford(index_set({-3,4,7})); -3 in x\n Traceback (most recent call last):\n
...\n TypeError: Not applicable.\n ";
07568 #if CYTHON_COMPILING_IN_CPYTHON
07569 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_6__contains__;
07570 #endif
07571 static int __pyx_pw_8PyClical_8clifford_7__contains__(PyObject *__pyx_v_self, PyObject
07572 *__pyx_v_x) {
07572 int __pyx_r;
07573 __Pyx_RefNannyDeclarations
07574 __Pyx_RefNannySetupContext("__contains__ (wrapper)", 0);
07575 __pyx_r = __pyx_pf_8PyClical_8clifford_6__contains__(((struct __pyx_obj_8PyClical_clifford
07576 *)__pyx_v_self), ((PyObject *)__pyx_v_x));
07577
07577 /* function exit code */
07578 __Pyx_RefNannyFinishContext();
07579 return __pyx_r;
07580 }
07581
07582 static int __pyx_pf_8PyClical_8clifford_6__contains__(CYTHON_UNUSED struct
__pyx_obj_8PyClical_clifford *__pyx_v_self, CYTHON_UNUSED PyObject *__pyx_v_x) {
07583 int __pyx_r;
07584 __Pyx_RefNannyDeclarations
07585 PyObject *__pyx_t_1 = NULL;
07586 int __pyx_lineno = 0;
07587 const char *__pyx_filename = NULL;
07588 int __pyx_clineno = 0;
07589 __Pyx_RefNannySetupContext("__contains__", 0);
07590
07591 /* "PyClical.pyx":636
07592 * TypeError: Not applicable.
07593 * """
07594 * raise TypeError("Not applicable.") # ««««««««
07595 *
07596 * def __iter__(self):
07597 */
07598 __pyx_t_1 = __Pyx_PyObject_Call(__pyx_builtin_TypeError, __pyx_tuple__10, NULL); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 636, __pyx_L1_error)
07599 __Pyx_GOTREF(__pyx_t_1);
07600 __Pyx_Raise(__pyx_t_1, 0, 0, 0);
07601 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
07602 __PYX_ERR(0, 636, __pyx_L1_error)
07603
07604 /* "PyClical.pyx":627
07605 * del self.instance
07606 *
07607 * def __contains__(self, x): # ««««««««
07608 * """
07609 * Not applicable.
07610 */
07611
07612 /* function exit code */
07613 __pyx_L1_error:;
07614 __Pyx_XDECREF(__pyx_t_1);
07615 __Pyx_AddTraceback("PyClical.clifford.__contains__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
07616 __pyx_r = -1;
07617 __Pyx_RefNannyFinishContext();
07618 return __pyx_r;
07619 }
07620
07621 /* "PyClical.pyx":638

```

```

07622 * raise TypeError("Not applicable.")
07623 *
07624 * def __iter__(self): # ««««««««
07625 * """
07626 * Not applicable.
07627 */
07628
07629 /* Python wrapper */
07630 static PyObject *__pyx_pw_8PyClical_8clifford_9__iter__(PyObject *__pyx_v_self); /*proto*/
07631 static char __pyx_doc_8PyClical_8clifford_8__iter__[] = "\n Not applicable.\n\n
>> for a in clifford(index_set({-3,4,7})):print(a, end=',')\n Traceback (most recent call
last):\n ... \n TypeError: Not applicable.\n ";
07632 #if CYTHON_COMPILING_IN_CPYTHON
07633 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_8__iter__;
07634 #endif
07635 static PyObject *__pyx_pw_8PyClical_8clifford_9__iter__(PyObject *__pyx_v_self) {
07636 PyObject *__pyx_r = 0;
07637 __Pyx_RefNannyDeclarations
07638 __Pyx_RefNannySetupContext("__iter__ (wrapper)", 0);
07639 __pyx_r = __pyx_pf_8PyClical_8clifford_8__iter__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
07640
07641 /* function exit code */
07642 __Pyx_RefNannyFinishContext();
07643 return __pyx_r;
07644 }
07645
07646 static PyObject *__pyx_pf_8PyClical_8clifford_8__iter__(CYTHON_UNUSED struct
__pyx_obj_8PyClical_clifford *__pyx_v_self) {
07647 PyObject *__pyx_r = NULL;
07648 __Pyx_RefNannyDeclarations
07649 PyObject *__pyx_t_1 = NULL;
07650 int __pyx_lineno = 0;
07651 const char *__pyx_filename = NULL;
07652 int __pyx_clineno = 0;
07653 __Pyx_RefNannySetupContext("__iter__", 0);
07654
07655 /* "PyClical.pyx":647
07656 * TypeError: Not applicable.
07657 * """
07658 * raise TypeError("Not applicable.") # ««««««««
07659 *
07660 * def reframe(self, ixt):
07661 */
07662 __pyx_t_1 = __Pyx_PyObject_Call(__pyx_builtin_TypeError, __pyx_tuple__10, NULL); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 647, __pyx_L1_error)
07663 __Pyx_GOTREF(__pyx_t_1);
07664 __Pyx_Raise(__pyx_t_1, 0, 0, 0);
07665 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
07666 __PYX_ERR(0, 647, __pyx_L1_error)
07667
07668 /* "PyClical.pyx":638
07669 * raise TypeError("Not applicable.")
07670 *
07671 * def __iter__(self): # ««««««««
07672 * """
07673 * Not applicable.
07674 */
07675
07676 /* function exit code */
07677 __pyx_L1_error:;
07678 __Pyx_XDECREF(__pyx_t_1);
07679 __Pyx_AddTraceback("PyClical.clifford.__iter__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
07680 __pyx_r = NULL;
07681 __Pyx_XGIVEREF(__pyx_r);
07682 __Pyx_RefNannyFinishContext();
07683 return __pyx_r;
07684 }
07685
07686 /* "PyClical.pyx":649
07687 * raise TypeError("Not applicable.")
07688 *
07689 * def reframe(self, ixt): # ««««««««
07690 * """
07691 * Put self into a larger frame, containing the union of self.frame() and index set ixt.
07692 */
07693
07694 /* Python wrapper */
07695 static PyObject *__pyx_pw_8PyClical_8clifford_11reframe(PyObject *__pyx_v_self, PyObject
*__pyx_v_ixt); /*proto*/
07696 static char __pyx_doc_8PyClical_8clifford_10reframe[] = "\n Put self into a larger
frame, containing the union of self.frame() and index set ixt.\n This can be used to make
multiplication faster, by multiplying within a common frame.\n\n >>
clifford(\"2+3{1}\").reframe(index_set({1,2,3}))\n clifford(\"2+3{1}\")\n >>
s=index_set({1,2,3});t=index_set({-3,-2,-1});x=random_clifford(s); x.reframe(t).frame() == (s|t);\n
True\n ";

```



```

07697 static PyObject *__pyx_pw_8PyClical_8clifford_1lreframe(PyObject *__pyx_v_self, PyObject
07698 *__pyx_v_ixt) {
07698 PyObject *__pyx_r = 0;
07699 __Pyx_RefNannyDeclarations
07700 __Pyx_RefNannySetupContext("reframe (wrapper)", 0);
07701 __pyx_r = __pyx_pf_8PyClical_8clifford_1lreframe(((struct __pyx_obj_8PyClical_clifford
07702 *)__pyx_v_self), ((PyObject *)__pyx_v_ixt));
07703
07703 /* function exit code */
07704 __Pyx_RefNannyFinishContext();
07705 return __pyx_r;
07706 }
07707
07708 static PyObject *__pyx_pf_8PyClical_8clifford_1lreframe(struct __pyx_obj_8PyClical_clifford
07709 *__pyx_v_self, PyObject *__pyx_v_ixt) {
07709 PyObject *__pyx_v_error_msg_prefix = NULL;
07710 struct __pyx_obj_8PyClical_clifford *__pyx_v_result = NULL;
07711 PyObject *__pyx_v_err = NULL;
07712 PyObject *__pyx_r = NULL;
07713 __Pyx_RefNannyDeclarations
07714 int __pyx_t_1;
07715 int __pyx_t_2;
07716 PyObject *__pyx_t_3 = NULL;
07717 PyObject *__pyx_t_4 = NULL;
07718 PyObject *__pyx_t_5 = NULL;
07719 PyObject *__pyx_t_6 = NULL;
07720 Clifford *__pyx_t_7;
07721 int __pyx_t_8;
07722 PyObject *__pyx_t_9 = NULL;
07723 PyObject *__pyx_t_10 = NULL;
07724 PyObject *__pyx_t_11 = NULL;
07725 PyObject *__pyx_t_12 = NULL;
07726 PyObject *__pyx_t_13 = NULL;
07727 int __pyx_t_14;
07728 char const *__pyx_t_15;
07729 PyObject *__pyx_t_16 = NULL;
07730 PyObject *__pyx_t_17 = NULL;
07731 PyObject *__pyx_t_18 = NULL;
07732 PyObject *__pyx_t_19 = NULL;
07733 PyObject *__pyx_t_20 = NULL;
07734 PyObject *__pyx_t_21 = NULL;
07735 int __pyx_lineno = 0;
07736 const char *__pyx_filename = NULL;
07737 int __pyx_clineno = 0;
07738 __Pyx_RefNannySetupContext("reframe", 0);
07739
07740 /* "PyClical.pyx":659
07741 * True
07742 * """
07743 * error_msg_prefix = "Cannot reframe" # ««««««««
07744 * if isinstance(ixt, index_set):
07745 * try:
07746 */
07747 __Pyx_INCREF(__pyx_kp_u_Cannot_reframe);
07748 __pyx_v_error_msg_prefix = __pyx_kp_u_Cannot_reframe;
07749
07750 /* "PyClical.pyx":660
07751 * """
07752 * error_msg_prefix = "Cannot reframe"
07753 * if isinstance(ixt, index_set): # ««««««««
07754 * try:
07755 * result = clifford()
07756 */
07757 __pyx_t_1 = __Pyx_TypeCheck(__pyx_v_ixt, __pyx_ptype_8PyClical_index_set);
07758 __pyx_t_2 = (__pyx_t_1 != 0);
07759 if (likely(__pyx_t_2)) {
07760
07761 /* "PyClical.pyx":661
07762 * error_msg_prefix = "Cannot reframe"
07763 * if isinstance(ixt, index_set):
07764 * try: # ««««««««
07765 * result = clifford()
07766 * result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
07767 */
07768 {
07769 __Pyx_PyThreadState_declare
07770 __Pyx_PyThreadState_assign
07771 __Pyx_ExceptionSave(&__pyx_t_3, &__pyx_t_4, &__pyx_t_5);
07772 __Pyx_XGOTREF(__pyx_t_3);
07773 __Pyx_XGOTREF(__pyx_t_4);
07774 __Pyx_XGOTREF(__pyx_t_5);
07775 /*try:*/ {
07776
07777 /* "PyClical.pyx":662
07778 * if isinstance(ixt, index_set):
07779 * try:
07780 * result = clifford() # ««««««««

```

```

07781 * result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
07782 * except RuntimeError as err:
07783 */
07784 __pyx_t_6 = __Pyx_PyObject_CallNoArg(((PyObject *) __pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_6)) __PYX_ERR(0, 662, __pyx_L4_error)
07785 __Pyx_GOTREF(__pyx_t_6);
07786 __pyx_v_result = ((struct __pyx_obj_8PyClical_clifford *) __pyx_t_6);
07787 __pyx_t_6 = 0;
07788
07789 /* "PyClical.pyx":663
07790 * try:
07791 * result = clifford()
07792 * result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
07793 # ««««««««
07793 * except RuntimeError as err:
07794 * raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
07795 */
07796 try {
07797 __pyx_t_7 = new Clifford(__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self),
__pyx_f_8PyClical_9index_set_unwrap(((struct __pyx_obj_8PyClical_index_set *) __pyx_v_ixt)));
07798 } catch (...) {
07799 __Pyx_CppExn2PyErr();
07800 __PYX_ERR(0, 663, __pyx_L4_error)
07801 }
07802 __pyx_v_result->instance = __pyx_t_7;
07803
07804 /* "PyClical.pyx":661
07805 * error_msg_prefix = "Cannot reframe"
07806 * if isinstance(ixt, index_set):
07807 * try:
07808 * # ««««««««
07808 * result = clifford()
07809 * result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
07810 */
07811 }
07812 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
07813 __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
07814 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
07815 goto __pyx_L9_try_end;
07816 __pyx_L4_error:;
07817 __Pyx_XDECREF(__pyx_t_6); __pyx_t_6 = 0;
07818
07819 /* "PyClical.pyx":664
07820 * result = clifford()
07821 * result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
07822 * except RuntimeError as err:
07823 * # ««««««««
07823 * raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
07824 * + str(ixt) + ":"
07825 */
07826 __pyx_t_8 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_RuntimeError);
07827 if (__pyx_t_8) {
07828 __Pyx_AddTraceback("PyClical.clifford.reframe", __pyx_clineno, __pyx_lineno,
__pyx_filename);
07829 if (__Pyx_GetException(&__pyx_t_6, &__pyx_t_9, &__pyx_t_10) < 0) __PYX_ERR(0, 664,
__pyx_L6_except_error)
07830 __Pyx_GOTREF(__pyx_t_6);
07831 __Pyx_GOTREF(__pyx_t_9);
07832 __Pyx_GOTREF(__pyx_t_10);
07833 __Pyx_INCREF(__pyx_t_9);
07834 __pyx_v_err = __pyx_t_9;
07835 /*try:*/ {
07836
07837 /* "PyClical.pyx":665
07838 * result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
07839 * except RuntimeError as err:
07840 * raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
07841 # ««««««««
07841 * + str(ixt) + ":"
07842 * + "\n\t" + str(err))
07843 */
07844 __pyx_t_11 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_from); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 665, __pyx_L15_error)
07845 __Pyx_GOTREF(__pyx_t_11);
07846 __pyx_t_12 = __Pyx_PyObject_CallOneArg(((PyObject *) (&PyUnicode_Type)), ((PyObject
*) __pyx_v_self)); if (unlikely(!__pyx_t_12)) __PYX_ERR(0, 665, __pyx_L15_error)
07847 __Pyx_GOTREF(__pyx_t_12);
07848 __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_t_11, __pyx_t_12); if
(unlikely(!__pyx_t_13)) __PYX_ERR(0, 665, __pyx_L15_error)
07849 __Pyx_GOTREF(__pyx_t_13);
07850 __Pyx_DECREF(__pyx_t_11); __pyx_t_11 = 0;
07851 __Pyx_DECREF(__pyx_t_12); __pyx_t_12 = 0;
07852 __pyx_t_12 = __Pyx_PyUnicode_Concat(__pyx_t_13, __pyx_kp_u_to_frame); if
(unlikely(!__pyx_t_12)) __PYX_ERR(0, 665, __pyx_L15_error)
07853 __Pyx_GOTREF(__pyx_t_12);
07854 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
07855
07856 /* "PyClical.pyx":666
07857 * except RuntimeError as err:

```

```

07858 * raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
07859 * + str(ixt) + ":", # ««««««««
07860 * + "\n\t" + str(err))
07861 * else:
07862 */
07863 __pyx_t_13 = __Pyx_PyObject_CallOneArg((PyObject *)(&PyUnicode_Type)),
__pyx_v_ixt); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 666, __pyx_L15_error)
07864 __Pyx_GOTREF(__pyx_t_13);
07865 __pyx_t_11 = __Pyx_PyUnicode_Concat(__pyx_t_12, __pyx_t_13); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 666, __pyx_L15_error)
07866 __Pyx_GOTREF(__pyx_t_11);
07867 __Pyx_DECREF(__pyx_t_12); __pyx_t_12 = 0;
07868 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
07869 __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_t_11, __pyx_kp_u_5); if
(unlikely(!__pyx_t_13)) __PYX_ERR(0, 666, __pyx_L15_error)
07870 __Pyx_GOTREF(__pyx_t_13);
07871 __Pyx_DECREF(__pyx_t_11); __pyx_t_11 = 0;
07872
07873 /* "PyClical.pyx":667
07874 * raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
07875 * + str(ixt) + ":", #
07876 * + "\n\t" + str(err))
07877 * else:
07878 * raise TypeError(error_msg_prefix + " using (" + str(type(ixt)) + ").")
07879 */
07880 __pyx_t_11 = __Pyx_PyUnicode_Concat(__pyx_t_13, __pyx_kp_u_6); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 667, __pyx_L15_error)
07881 __Pyx_GOTREF(__pyx_t_11);
07882 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
07883 __pyx_t_13 = __Pyx_PyObject_CallOneArg((PyObject *)(&PyUnicode_Type)),
__pyx_v_err); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 667, __pyx_L15_error)
07884 __Pyx_GOTREF(__pyx_t_13);
07885 __pyx_t_12 = __Pyx_PyUnicode_Concat(__pyx_t_11, __pyx_t_13); if
(unlikely(!__pyx_t_12)) __PYX_ERR(0, 667, __pyx_L15_error)
07886 __Pyx_GOTREF(__pyx_t_12);
07887 __Pyx_DECREF(__pyx_t_11); __pyx_t_11 = 0;
07888 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
07889
07890 /* "PyClical.pyx":665
07891 * result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
07892 * except RuntimeError as err:
07893 * raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
07894 * + str(ixt) + ":", # ««««««««
07895 * + "\n\t" + str(err))
07896 */
07897 __pyx_t_13 = __Pyx_PyObject_CallOneArg(__pyx_builtin_ValueError, __pyx_t_12); if
(unlikely(!__pyx_t_13)) __PYX_ERR(0, 665, __pyx_L15_error)
07898 __Pyx_GOTREF(__pyx_t_13);
07899 __Pyx_DECREF(__pyx_t_12); __pyx_t_12 = 0;
07900 __Pyx_Raise(__pyx_t_13, 0, 0, 0);
07901 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
07902 __PYX_ERR(0, 665, __pyx_L15_error)
07903 }
07904
07905 /* "PyClical.pyx":664
07906 * result = clifford()
07907 * result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
07908 * except RuntimeError as err: # ««««««««
07909 * raise ValueError(error_msg_prefix + " from " + str(self) + " to frame "
07910 * + str(ixt) + ":", #
07911 */
07912 /*finally:*/ {
07913 __pyx_L15_error;;
07914 /*exception exit:*/{
07915 __Pyx_PyThreadState_declare
07916 __Pyx_PyThreadState_assign
07917 __pyx_t_16 = 0; __pyx_t_17 = 0; __pyx_t_18 = 0; __pyx_t_19 = 0; __pyx_t_20 = 0;
__pyx_t_21 = 0;
07918 __Pyx_XDECREF(__pyx_t_11); __pyx_t_11 = 0;
07919 __Pyx_XDECREF(__pyx_t_12); __pyx_t_12 = 0;
07920 __Pyx_XDECREF(__pyx_t_13); __pyx_t_13 = 0;
07921 if (PY_MAJOR_VERSION >= 3) __Pyx_ExceptionSwap(&__pyx_t_19, &__pyx_t_20,
&__pyx_t_21);
07922 if ((PY_MAJOR_VERSION < 3) || unlikely(__Pyx_GetException(&__pyx_t_16,
&__pyx_t_17, &__pyx_t_18) < 0)) __Pyx_ErrFetch(&__pyx_t_16, &__pyx_t_17, &__pyx_t_18);
07923 __Pyx_XGOTREF(__pyx_t_16);
07924 __Pyx_XGOTREF(__pyx_t_17);
07925 __Pyx_XGOTREF(__pyx_t_18);
07926 __Pyx_XGOTREF(__pyx_t_19);
07927 __Pyx_XGOTREF(__pyx_t_20);
07928 __Pyx_XGOTREF(__pyx_t_21);
07929 __pyx_t_8 = __pyx_lineno; __pyx_t_14 = __pyx_clineno; __pyx_t_15 = __pyx_filename;
07930 {
07931 __Pyx_DECREF(__pyx_v_err);
07932 __pyx_v_err = NULL;

```

```

07933 }
07934 if (PY_MAJOR_VERSION >= 3) {
07935 __Pyx_XGIVEREF(__pyx_t_19);
07936 __Pyx_XGIVEREF(__pyx_t_20);
07937 __Pyx_XGIVEREF(__pyx_t_21);
07938 __Pyx_ExceptionReset(__pyx_t_19, __pyx_t_20, __pyx_t_21);
07939 }
07940 __Pyx_XGIVEREF(__pyx_t_16);
07941 __Pyx_XGIVEREF(__pyx_t_17);
07942 __Pyx_XGIVEREF(__pyx_t_18);
07943 __Pyx_ErrRestore(__pyx_t_16, __pyx_t_17, __pyx_t_18);
07944 __pyx_t_16 = 0; __pyx_t_17 = 0; __pyx_t_18 = 0; __pyx_t_19 = 0; __pyx_t_20 = 0;
__pyx_t_21 = 0;
07945 __pyx_lineno = __pyx_t_8; __pyx_clineno = __pyx_t_14; __pyx_filename = __pyx_t_15;
07946 goto __pyx_L6_except_error;
07947 }
07948 }
07949 }
07950 goto __pyx_L6_except_error;
07951 __pyx_L6_except_error;
07952
07953 /* "PyClical.pyx":661
07954 * error_msg_prefix = "Cannot reframe"
07955 * if isinstance(ixt, index_set):
07956 * try:
07957 * # ««««««««
07958 * result = clifford()
07959 * result.instance = new Clifford(self.unwrap(), (<index_set>ixt).unwrap())
07960 */
07961 __Pyx_XGIVEREF(__pyx_t_3);
07962 __Pyx_XGIVEREF(__pyx_t_4);
07963 __Pyx_XGIVEREF(__pyx_t_5);
07964 __Pyx_ExceptionReset(__pyx_t_3, __pyx_t_4, __pyx_t_5);
07965 goto __pyx_L1_error;
07966 __pyx_L9_try_end;
07967 }
07968
07969 /* "PyClical.pyx":660
07970 * """
07971 * error_msg_prefix = "Cannot reframe"
07972 * if isinstance(ixt, index_set):
07973 * try:
07974 * # ««««««««
07975 * result = clifford()
07976 */
07977 goto __pyx_L3;
07978 }
07979
07980 /* "PyClical.pyx":669
07981 *
07982 * + "\n\t" + str(err))
07983 *
07984 * else:
07985 * raise TypeError(error_msg_prefix + " using (" + str(type(ixt)) + ").")
07986 * #
07987 * return result
07988 *
07989 */
07990 /*else*/ {
07991 __pyx_t_10 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_kp_u_using); if
(unlikely(!__pyx_t_10)) __PYX_ERR(0, 669, __pyx_L1_error)
07992 __Pyx_GOTREF(__pyx_t_10);
07993 __pyx_t_9 = __Pyx_PyObject_CallOneArg((PyObject *)(&PyUnicode_Type)), ((PyObject
*)Py_TYPE(__pyx_v_ixt)); if (unlikely(!__pyx_t_9)) __PYX_ERR(0, 669, __pyx_L1_error)
07994 __Pyx_GOTREF(__pyx_t_9);
07995 __pyx_t_6 = __Pyx_PyUnicode_Concat(__pyx_t_10, __pyx_t_9); if (unlikely(!__pyx_t_6))
__PYX_ERR(0, 669, __pyx_L1_error)
07996 __Pyx_GOTREF(__pyx_t_6);
07997 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
07998 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
07999 __pyx_t_9 = __Pyx_PyUnicode_Concat(__pyx_t_6, __pyx_kp_u_9); if (unlikely(!__pyx_t_9))
__PYX_ERR(0, 669, __pyx_L1_error)
08000 __Pyx_GOTREF(__pyx_t_9);
08001 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
08002 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
08003 __Pyx_Raise(__pyx_t_6, 0, 0, 0);
08004 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
08005 __PYX_ERR(0, 669, __pyx_L1_error)
08006 }
08007 __pyx_L3;
08008
08009 /* "PyClical.pyx":670
08010 *
08011 * else:
08012 * raise TypeError(error_msg_prefix + " using (" + str(type(ixt)) + ").")
08013 *
08014 * return result
08015 * # ««««««««
08016 *
08017 * def __richcmp__(lhs, rhs, int op):
08018 */

```

```

08013 __Pyx_XDECREF(__pyx_r);
08014 __Pyx_INCREF((PyObject *)__pyx_v_result);
08015 __pyx_r = ((PyObject *)__pyx_v_result);
08016 goto __pyx_L0;
08017
08018 /* "PyClical.pyx":649
08019 * raise TypeError("Not applicable.")
08020 *
08021 * def reframe(self, ixt): # ««««««««
08022 * """
08023 * Put self into a larger frame, containing the union of self.frame() and index set ixt.
08024 */
08025
08026 /* function exit code */
08027 __pyx_L1_error:;
08028 __Pyx_XDECREF(__pyx_t_6);
08029 __Pyx_XDECREF(__pyx_t_9);
08030 __Pyx_XDECREF(__pyx_t_10);
08031 __Pyx_XDECREF(__pyx_t_11);
08032 __Pyx_XDECREF(__pyx_t_12);
08033 __Pyx_XDECREF(__pyx_t_13);
08034 __Pyx_AddTraceback("PyClical.clifford.reframe", __pyx_clineno, __pyx_lineno,
__pyx_filename);
08035 __pyx_r = NULL;
08036 __pyx_L0:;
08037 __Pyx_XDECREF(__pyx_v_error_msg_prefix);
08038 __Pyx_XDECREF((PyObject *)__pyx_v_result);
08039 __Pyx_XDECREF(__pyx_v_err);
08040 __Pyx_XGIVEREF(__pyx_r);
08041 __Pyx_RefNannyFinishContext();
08042 return __pyx_r;
08043 }
08044
08045 /* "PyClical.pyx":672
08046 * return result
08047 *
08048 * def __richcmp__(lhs, rhs, int op): # ««««««««
08049 * """
08050 * Compare objects of type clifford.
08051 */
08052
08053 /* Python wrapper */
08054 static PyObject *__pyx_pw_8PyClical_8clifford_13__richcmp__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs, int __pyx_v_op); /*proto*/
08055 static PyObject *__pyx_pw_8PyClical_8clifford_13__richcmp__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs, int __pyx_v_op) {
08056 PyObject *__pyx_r = 0;
08057 __Pyx_RefNannyDeclarations
08058 __Pyx_RefNannySetupContext("__richcmp__ (wrapper)", 0);
08059 __pyx_r = __pyx_pf_8PyClical_8clifford_12__richcmp__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_lhs), ((PyObject *)__pyx_v_rhs), ((int)__pyx_v_op));
08060
08061 /* function exit code */
08062 __Pyx_RefNannyFinishContext();
08063 return __pyx_r;
08064 }
08065
08066 static PyObject *__pyx_pf_8PyClical_8clifford_12__richcmp__(struct
__pyx_obj_8PyClical_clifford *__pyx_v_lhs, PyObject *__pyx_v_rhs, int __pyx_v_op) {
08067 PyObject *__pyx_r = NULL;
08068 __Pyx_RefNannyDeclarations
08069 int __pyx_t_1;
08070 int __pyx_t_2;
08071 int __pyx_t_3;
08072 PyObject *__pyx_t_4 = NULL;
08073 PyObject *__pyx_t_5 = NULL;
08074 PyObject *__pyx_t_6 = NULL;
08075 int __pyx_lineno = 0;
08076 const char *__pyx_filename = NULL;
08077 int __pyx_clineno = 0;
08078 __Pyx_RefNannySetupContext("__richcmp__", 0);
08079
08080 /* "PyClical.pyx":691
08081 * True
08082 * """
08083 * if op == 2: # == # ««««««««
08084 * if (lhs is None) or (rhs is None):
08085 * return bool(lhs is rhs)
08086 */
08087 __pyx_t_1 = ((__pyx_v_op == 2) != 0);
08088 if (__pyx_t_1) {
08089
08090 /* "PyClical.pyx":692
08091 * """
08092 * if op == 2: # ==
08093 * if (lhs is None) or (rhs is None): # ««««««««
08094 * return bool(lhs is rhs)

```

```

08095 * else:
08096 */
08097 __pyx_t_2 = ((PyObject *)__pyx_v_lhs) == Py_None);
08098 __pyx_t_3 = (__pyx_t_2 != 0);
08099 if (!__pyx_t_3) {
08100 } else {
08101 __pyx_t_1 = __pyx_t_3;
08102 goto __pyx_L5_bool_binop_done;
08103 }
08104 __pyx_t_3 = (__pyx_v_rhs == Py_None);
08105 __pyx_t_2 = (__pyx_t_3 != 0);
08106 __pyx_t_1 = __pyx_t_2;
08107 __pyx_L5_bool_binop_done;
08108 if (__pyx_t_1) {
08109
08110 /* "PyClicl.pyx":693
08111 * if op == 2: # ==
08112 * if (lhs is None) or (rhs is None):
08113 * return bool(lhs is rhs) # ««««««««
08114 * else:
08115 * return bool(toClifford(lhs) == toClifford(rhs))
08116 */
08117 __Pyx_XDECREF(__pyx_r);
08118 __pyx_t_1 = ((PyObject *)__pyx_v_lhs) == __pyx_v_rhs);
08119 __pyx_t_4 = __Pyx_PyBool_FromLong((!(!__pyx_t_1))); if (unlikely(!__pyx_t_4))
__PYX_ERR(0, 693, __pyx_L1_error)
08120 __Pyx_GOTREF(__pyx_t_4);
08121 __pyx_r = __pyx_t_4;
08122 __pyx_t_4 = 0;
08123 goto __pyx_L0;
08124
08125 /* "PyClicl.pyx":692
08126 * """
08127 * if op == 2: # ==
08128 * if (lhs is None) or (rhs is None): # ««««««««
08129 * return bool(lhs is rhs)
08130 * else:
08131 */
08132 }
08133
08134 /* "PyClicl.pyx":695
08135 * return bool(lhs is rhs)
08136 * else:
08137 * return bool(toClifford(lhs) == toClifford(rhs)) # ««««««««
08138 * elif op == 3: # !=
08139 * if (lhs is None) or (rhs is None):
08140 */
08141 /*else*/ {
08142 __Pyx_XDECREF(__pyx_r);
08143 __pyx_t_1 = (__pyx_f_8PyClicl_toClifford((PyObject *)__pyx_v_lhs)) ==
__pyx_f_8PyClicl_toClifford(__pyx_v_rhs));
08144 __pyx_t_4 = __Pyx_PyBool_FromLong((!(!__pyx_t_1))); if (unlikely(!__pyx_t_4))
__PYX_ERR(0, 695, __pyx_L1_error)
08145 __Pyx_GOTREF(__pyx_t_4);
08146 __pyx_r = __pyx_t_4;
08147 __pyx_t_4 = 0;
08148 goto __pyx_L0;
08149 }
08150
08151 /* "PyClicl.pyx":691
08152 * True
08153 * """
08154 * if op == 2: # == # ««««««««
08155 * if (lhs is None) or (rhs is None):
08156 * return bool(lhs is rhs)
08157 */
08158 }
08159
08160 /* "PyClicl.pyx":696
08161 * else:
08162 * return bool(toClifford(lhs) == toClifford(rhs))
08163 * elif op == 3: # != # ««««««««
08164 * if (lhs is None) or (rhs is None):
08165 * return not bool(lhs is rhs)
08166 */
08167 __pyx_t_1 = ((__pyx_v_op == 3) != 0);
08168 if (__pyx_t_1) {
08169
08170 /* "PyClicl.pyx":697
08171 * return bool(toClifford(lhs) == toClifford(rhs))
08172 * elif op == 3: # !=
08173 * if (lhs is None) or (rhs is None): # ««««««««
08174 * return not bool(lhs is rhs)
08175 * else:
08176 */
08177 __pyx_t_2 = ((PyObject *)__pyx_v_lhs) == Py_None);
08178 __pyx_t_3 = (__pyx_t_2 != 0);

```

```

08179 if (!__pyx_t_3) {
08180 } else {
08181 __pyx_t_1 = __pyx_t_3;
08182 goto __pyx_L8_bool_binop_done;
08183 }
08184 __pyx_t_3 = (__pyx_v_rhs == Py_None);
08185 __pyx_t_2 = (__pyx_t_3 != 0);
08186 __pyx_t_1 = __pyx_t_2;
08187 __pyx_L8_bool_binop_done;
08188 if (__pyx_t_1) {
08189
08190 /* "PyClical.pyx":698
08191 *
08192 * if (lhs is None) or (rhs is None):
08193 * return not bool(lhs is rhs) # ««««««««
08194 *
08195 * else:
08196 * return bool(toClifford(lhs) != toClifford(rhs))
08197 */
08198 __Pyx_XDECREF(__pyx_r);
08199 __pyx_t_1 = (((PyObject *) __pyx_v_lhs) == __pyx_v_rhs);
08200 __pyx_t_4 = __Pyx_PyBool_FromLong((((!(!__pyx_t_1)) != 0))); if (unlikely(!__pyx_t_4))
__PYX_ERR(0, 698, __pyx_L1_error)
08201 __Pyx_GOTREF(__pyx_t_4);
08202 __pyx_r = __pyx_t_4;
08203 __pyx_t_4 = 0;
08204 goto __pyx_L0;
08205
08206 /* "PyClical.pyx":697
08207 *
08208 * return bool(toClifford(lhs) == toClifford(rhs))
08209 *
08210 * elif op == 3: # !=
08211 * if (lhs is None) or (rhs is None):
08212 * return not bool(lhs is rhs) # ««««««««
08213 *
08214 * else:
08215 */
08216 }
08217
08218 /* "PyClical.pyx":700
08219 *
08220 * return not bool(lhs is rhs)
08221 *
08222 * else:
08223 * return bool(toClifford(lhs) != toClifford(rhs)) # ««««««««
08224 *
08225 * elif isinstance(lhs, clifford) or isinstance(rhs, clifford):
08226 * raise TypeError("This comparison operator is not implemented for "
08227 */
08228 /*else*/ {
08229 __Pyx_XDECREF(__pyx_r);
08230 __pyx_t_1 = (__pyx_f_8PyClical_toClifford(((PyObject *) __pyx_v_lhs)) !=
__pyx_f_8PyClical_toClifford(__pyx_v_rhs));
08231 __pyx_t_4 = __Pyx_PyBool_FromLong((((!(!__pyx_t_1)) != 0))); if (unlikely(!__pyx_t_4))
__PYX_ERR(0, 700, __pyx_L1_error)
08232 __Pyx_GOTREF(__pyx_t_4);
08233 __pyx_r = __pyx_t_4;
08234 __pyx_t_4 = 0;
08235 goto __pyx_L0;
08236 }
08237
08238 /* "PyClical.pyx":696
08239 *
08240 * else:
08241 * return bool(toClifford(lhs) == toClifford(rhs))
08242 *
08243 * elif op == 3: # !=
08244 * if (lhs is None) or (rhs is None):
08245 * return not bool(lhs is rhs) # ««««««««
08246 */
08247 }
08248
08249 /* "PyClical.pyx":701
08250 *
08251 * else:
08252 * return bool(toClifford(lhs) != toClifford(rhs))
08253 *
08254 * elif isinstance(lhs, clifford) or isinstance(rhs, clifford):
08255 * raise TypeError("This comparison operator is not implemented for "
08256 * + str(type(lhs)) + ", " + str(type(rhs)) + ".")
08257 */
08258 __pyx_t_2 = __Pyx_TypeCheck(((PyObject *) __pyx_v_lhs), __pyx_ptype_8PyClical_clifford);
08259 __pyx_t_3 = (__pyx_t_2 != 0);
08260 if (!__pyx_t_3) {
08261 } else {
08262 __pyx_t_1 = __pyx_t_3;
08263 goto __pyx_L10_bool_binop_done;
08264 }
08265 __pyx_t_3 = __Pyx_TypeCheck(__pyx_v_rhs, __pyx_ptype_8PyClical_clifford);
08266 __pyx_t_2 = (__pyx_t_3 != 0);
08267 __pyx_t_1 = __pyx_t_2;
08268 __pyx_L10_bool_binop_done;
08269 if (unlikely(__pyx_t_1)) {
08270
08271 /* "PyClical.pyx":703
08272 *
08273 * elif isinstance(lhs, clifford) or isinstance(rhs, clifford):
08274 * raise TypeError("This comparison operator is not implemented for "

```

```

08263 * + str(type(lhs)) + ", " + str(type(rhs)) + ".") # ««««««««
08264 * else:
08265 * return NotImplemented
08266 */
08267 __pyx_t_4 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject
*)Py_TYPE(((PyObject *)__pyx_v_lhs))))); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 703, __pyx_L1_error)
08268 __Pyx_GOTREF(__pyx_t_4);
08269 __pyx_t_5 = __Pyx_PyUnicode_Concat(__pyx_kp_u_This_comparison_operator_is_not, __pyx_t_4);
08270 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 703, __pyx_L1_error)
08271 __Pyx_GOTREF(__pyx_t_5);
08272 __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
08273 __pyx_t_4 = __Pyx_PyUnicode_Concat(__pyx_t_5, __pyx_kp_u_8); if (unlikely(!__pyx_t_4))
__PYX_ERR(0, 703, __pyx_L1_error)
08274 __Pyx_GOTREF(__pyx_t_4);
08275 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
08276 __pyx_t_5 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject
*)Py_TYPE(__pyx_v_rhs))); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 703, __pyx_L1_error)
08277 __Pyx_GOTREF(__pyx_t_5);
08278 __pyx_t_6 = __Pyx_PyUnicode_Concat(__pyx_t_4, __pyx_t_5); if (unlikely(!__pyx_t_6))
__PYX_ERR(0, 703, __pyx_L1_error)
08279 __Pyx_GOTREF(__pyx_t_6);
08280 __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
08281 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
08282 __pyx_t_5 = __Pyx_PyUnicode_Concat(__pyx_t_6, __pyx_kp_u_); if (unlikely(!__pyx_t_5))
__PYX_ERR(0, 703, __pyx_L1_error)
08283 __Pyx_GOTREF(__pyx_t_5);
08284 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
08285 /* "PyClicl.pyx":702
08286 * return bool(toClifford(lhs) != toClifford(rhs))
08287 * elif isinstance(lhs, clifford) or isinstance(rhs, clifford):
08288 * raise TypeError("This comparison operator is not implemented for " #
««««««««
08289 * + str(type(lhs)) + ", " + str(type(rhs)) + ".")
08290 * else:
08291 */
08292 __pyx_t_6 = __Pyx_PyObject_CallOneArg(__pyx_builtin_TypeError, __pyx_t_5); if
(unlikely(!__pyx_t_6)) __PYX_ERR(0, 702, __pyx_L1_error)
08293 __Pyx_GOTREF(__pyx_t_6);
08294 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
08295 __Pyx_Raise(__pyx_t_6, 0, 0, 0);
08296 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
08297 __PYX_ERR(0, 702, __pyx_L1_error)
08298 /* "PyClicl.pyx":701
08299 * else:
08300 * return bool(toClifford(lhs) != toClifford(rhs))
08301 * elif isinstance(lhs, clifford) or isinstance(rhs, clifford): # ««««««««
08302 * raise TypeError("This comparison operator is not implemented for "
08303 * + str(type(lhs)) + ", " + str(type(rhs)) + ".")
08304 *
08305 */
08306 }
08307
08308 /* "PyClicl.pyx":705
08309 * + str(type(lhs)) + ", " + str(type(rhs)) + ".")
08310 * else:
08311 * return NotImplemented # ««««««««
08312 *
08313 def __getitem__(self, ixt):
08314 */
08315 /*else*/ {
08316 __Pyx_XDECREF(__pyx_r);
08317 __Pyx_INCREF(__pyx_builtin_NotImplemented);
08318 __pyx_r = __pyx_builtin_NotImplemented;
08319 goto __pyx_L0;
08320 }
08321
08322 /* "PyClicl.pyx":672
08323 * return result
08324 *
08325 def __richcmp__(lhs, rhs, int op): # ««««««««
08326 * """
08327 * Compare objects of type clifford.
08328 */
08329
08330 /* function exit code */
08331 __pyx_L1_error:;
08332 __Pyx_XDECREF(__pyx_t_4);
08333 __Pyx_XDECREF(__pyx_t_5);
08334 __Pyx_XDECREF(__pyx_t_6);
08335 __Pyx_AddTraceback("PyClicl.clifford.__richcmp__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
08336 __pyx_r = NULL;
08337 __pyx_L0:;
08338 __Pyx_XGIVEREF(__pyx_r);
08339 __Pyx_RefNannyFinishContext();
08340 return __pyx_r;

```



```

08341 }
08342
08343 /* "PyClical.pyx":707
08344 * return NotImplemented
08345 *
08346 * def __getitem__(self, ixt): # ««««««««
08347 * """
08348 * Subscripting: map from index set to scalar coordinate.
08349 */
08350
08351 /* Python wrapper */
08352 static PyObject *__pyx_pw_8PyClical_8clifford_15__getitem__(PyObject *__pyx_v_self, PyObject
__pyx_v_ixt); /*proto*/
08353 static char __pyx_doc_8PyClical_8clifford_14__getitem__[] = "\n Subscripting: map from
index set to scalar coordinate.\n\n >> clifford(\"{1}\") [index_set(1)]\n 1.0\nclifford(\"{1}\") [index_set({1})]\n 1.0\n >> clifford(\"{1}\") [index_set({1,2})]\n 0.0\n >> clifford(\"2{1,2}\") [index_set({1,2})]\n 2.0\n ";
08354 #if CYTHON_COMPILING_IN_CPYTHON
08355 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_14__getitem__;
08356 #endif
08357 static PyObject *__pyx_pw_8PyClical_8clifford_15__getitem__(PyObject *__pyx_v_self, PyObject
__pyx_v_ixt) {
08358 PyObject *__pyx_r = 0;
08359 __Pyx_RefNannyDeclarations
08360 __Pyx_RefNannySetupContext("__getitem__ (wrapper)", 0);
08361 __pyx_r = __pyx_pf_8PyClical_8clifford_14__getitem__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_ixt));
08362
08363 /* function exit code */
08364 __Pyx_RefNannyFinishContext();
08365 return __pyx_r;
08366 }
08367
08368 static PyObject *__pyx_pf_8PyClical_8clifford_14__getitem__(struct
__pyx_obj_8PyClical_clifford *__pyx_v_self, PyObject *__pyx_v_ixt) {
08369 PyObject *__pyx_r = NULL;
08370 __Pyx_RefNannyDeclarations
08371 PyObject *__pyx_t_1 = NULL;
08372 int __pyx_lineno = 0;
08373 const char *__pyx_filename = NULL;
08374 int __pyx_clineno = 0;
08375 __Pyx_RefNannySetupContext("__getitem__", 0);
08376
08377 /* "PyClical.pyx":720
08378 * 2.0
08379 * """
08380 * return self.instance.getitem(toIndexSet(ixt)) # ««««««««
08381 *
08382 * def __neg__(self):
08383 */
08384 __Pyx_XDECREF(__pyx_r);
08385 __pyx_t_1 =
PyFloat_FromDouble(__pyx_v_self->instance->operator[](__pyx_f_8PyClical_toIndexSet(__pyx_v_ixt))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 720, __pyx_L1_error)
08386 __Pyx_GOTREF(__pyx_t_1);
08387 __pyx_r = __pyx_t_1;
08388 __pyx_t_1 = 0;
08389 goto __pyx_L0;
08390
08391 /* "PyClical.pyx":707
08392 * return NotImplemented
08393 *
08394 * def __getitem__(self, ixt): # ««««««««
08395 * """
08396 * Subscripting: map from index set to scalar coordinate.
08397 */
08398
08399 /* function exit code */
08400 __pyx_L1_error;
08401 __Pyx_XDECREF(__pyx_t_1);
08402 __Pyx_AddTraceback("PyClical.clifford.__getitem__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
08403 __pyx_r = NULL;
08404 __pyx_L0;
08405 __Pyx_XGIVEREF(__pyx_r);
08406 __Pyx_RefNannyFinishContext();
08407 return __pyx_r;
08408 }
08409
08410 /* "PyClical.pyx":722
08411 * return self.instance.getitem(toIndexSet(ixt))
08412 *
08413 * def __neg__(self): # ««««««««
08414 * """
08415 * Unary -.
08416 */
08417

```

```

08418 /* Python wrapper */
08419 static PyObject *__pyx_pw_8PyClical_8clifford_17__neg__(PyObject *__pyx_v_self); /*proto*/
08420 static char __pyx_doc_8PyClical_8clifford_16__neg__[] = "\n Unary -. \n\n »>
print(-clifford("{1}"))\n -{1}\n ";
08421 #if CYTHON_COMPILING_IN_CPYTHON
08422 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_16__neg__;
08423 #endif
08424 static PyObject *__pyx_pw_8PyClical_8clifford_17__neg__(PyObject *__pyx_v_self) {
08425 PyObject *__pyx_r = 0;
08426 __Pyx_RefNannyDeclarations
08427 __Pyx_RefNannySetupContext("__neg__ (wrapper)", 0);
08428 __pyx_r = __pyx_pf_8PyClical_8clifford_16__neg__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
08429
08430 /* function exit code */
08431 __Pyx_RefNannyFinishContext();
08432 return __pyx_r;
08433 }
08434
08435 static PyObject *__pyx_pf_8PyClical_8clifford_16__neg__(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
08436 PyObject *__pyx_r = NULL;
08437 __Pyx_RefNannyDeclarations
08438 PyObject *__pyx_t_1 = NULL;
08439 PyObject *__pyx_t_2 = NULL;
08440 int __pyx_lineno = 0;
08441 const char *__pyx_filename = NULL;
08442 int __pyx_clineno = 0;
08443 __Pyx_RefNannySetupContext("__neg__", 0);
08444
08445 /* "PyClical.pyx":729
08446 * -{1}
08447 * """
08448 * return clifford().wrap(self.instance.neg()) # ««««««««
08449 *
08450 * def __pos__(self):
08451 */
08452 __Pyx_XDECREF(__pyx_r);
08453 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 729, __pyx_L1_error)
08454 __Pyx_GOTREF(__pyx_t_1);
08455 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), __pyx_v_self->instance->operator-()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 729,
__pyx_L1_error)
08456 __Pyx_GOTREF(__pyx_t_2);
08457 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
08458 __pyx_r = __pyx_t_2;
08459 __pyx_t_2 = 0;
08460 goto __pyx_L0;
08461
08462 /* "PyClical.pyx":722
08463 * return self.instance.getitem(toIndexSet(ixt))
08464 *
08465 * def __neg__(self): # ««««««««
08466 * """
08467 * Unary -.
08468 */
08469
08470 /* function exit code */
08471 __pyx_L1_error:;
08472 __Pyx_XDECREF(__pyx_t_1);
08473 __Pyx_XDECREF(__pyx_t_2);
08474 __Pyx_AddTraceback("PyClical.clifford.__neg__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
08475 __pyx_r = NULL;
08476 __pyx_L0:;
08477 __Pyx_XGIVEREF(__pyx_r);
08478 __Pyx_RefNannyFinishContext();
08479 return __pyx_r;
08480 }
08481
08482 /* "PyClical.pyx":731
08483 * return clifford().wrap(self.instance.neg())
08484 *
08485 * def __pos__(self): # ««««««««
08486 * """
08487 * Unary +.
08488 */
08489
08490 /* Python wrapper */
08491 static PyObject *__pyx_pw_8PyClical_8clifford_19__pos__(PyObject *__pyx_v_self); /*proto*/
08492 static char __pyx_doc_8PyClical_8clifford_18__pos__[] = "\n Unary +. \n\n »>
print(+clifford("{1}"))\n {1}\n ";
08493 #if CYTHON_COMPILING_IN_CPYTHON
08494 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_18__pos__;
08495 #endif
08496 static PyObject *__pyx_pw_8PyClical_8clifford_19__pos__(PyObject *__pyx_v_self) {

```

```

08497 PyObject *__pyx_r = 0;
08498 __Pyx_RefNannyDeclarations
08499 __Pyx_RefNannySetupContext("__pos__ (wrapper)", 0);
08500 __pyx_r = __pyx_pf_8PyClical_8clifford_18__pos__((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
08501
08502 /* function exit code */
08503 __Pyx_RefNannyFinishContext();
08504 return __pyx_r;
08505 }
08506
08507 static PyObject *__pyx_pf_8PyClical_8clifford_18__pos__(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
08508 PyObject *__pyx_r = NULL;
08509 __Pyx_RefNannyDeclarations
08510 PyObject *__pyx_t_1 = NULL;
08511 int __pyx_lineno = 0;
08512 const char *__pyx_filename = NULL;
08513 int __pyx_clineno = 0;
08514 __Pyx_RefNannySetupContext("__pos__", 0);
08515
08516 /* "PyClical.pyx":738
08517 * {1}
08518 * """
08519 * return clifford(self) # ««««««««
08520 *
08521 * def __add__(lhs, rhs):
08522 */
08523 __Pyx_XDECREF(__pyx_r);
08524 __pyx_t_1 = __Pyx_PyObject_CallOneArg((PyObject *)__pyx_ptype_8PyClical_clifford),
((PyObject *)__pyx_v_self)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 738, __pyx_L1_error)
08525 __Pyx_GOTREF(__pyx_t_1);
08526 __pyx_r = __pyx_t_1;
08527 __pyx_t_1 = 0;
08528 goto __pyx_L0;
08529
08530 /* "PyClical.pyx":731
08531 * return clifford().wrap(self.instance.neg())
08532 *
08533 * def __pos__(self):
08534 * """
08535 * Unary +.
08536 */
08537
08538 /* function exit code */
08539 __pyx_L1_error:;
08540 __Pyx_XDECREF(__pyx_t_1);
08541 __Pyx_AddTraceback("PyClical.clifford.__pos__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
08542 __pyx_r = NULL;
08543 __pyx_L0:;
08544 __Pyx_XGIVEREF(__pyx_r);
08545 __Pyx_RefNannyFinishContext();
08546 return __pyx_r;
08547 }
08548
08549 /* "PyClical.pyx":740
08550 * return clifford(self)
08551 *
08552 * def __add__(lhs, rhs):
08553 * """
08554 * Geometric sum.
08555 */
08556
08557 /* Python wrapper */
08558 static PyObject *__pyx_pw_8PyClical_8clifford_21__add__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs); /*proto*/
08559 static char __pyx_doc_8PyClical_8clifford_20__add__[] = "\n Geometric sum.\n\n
>> print(clifford(1) + clifford(\"{2}\"))\n 1+{2}\n >> print(clifford(\"{1}\") +
clifford(\"{2}\"))\n {1}+{2}\n ";
08560 #if CYTHON_COMPILING_IN_CPYTHON
08561 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_20__add__;
08562 #endif
08563 static PyObject *__pyx_pw_8PyClical_8clifford_21__add__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs) {
08564 PyObject *__pyx_r = 0;
08565 __Pyx_RefNannyDeclarations
08566 __Pyx_RefNannySetupContext("__add__ (wrapper)", 0);
08567 __pyx_r = __pyx_pf_8PyClical_8clifford_20__add__((PyObject *)__pyx_v_lhs), ((PyObject
*)__pyx_v_rhs));
08568
08569 /* function exit code */
08570 __Pyx_RefNannyFinishContext();
08571 return __pyx_r;
08572 }
08573
08574 static PyObject *__pyx_pf_8PyClical_8clifford_20__add__(PyObject *__pyx_v_lhs, PyObject

```

```

__pyx_v_rhs) {
08575 PyObject *__pyx_r = NULL;
08576 __Pyx_RefNannyDeclarations
08577 PyObject *__pyx_t_1 = NULL;
08578 PyObject *__pyx_t_2 = NULL;
08579 int __pyx_lineno = 0;
08580 const char *__pyx_filename = NULL;
08581 int __pyx_clineno = 0;
08582 __Pyx_RefNannySetupContext("__add__", 0);
08583
08584 /* "PyClicl.pyx":749
08585 * {1}+{2}
08586 * """
08587 * return clifford().wrap(toClifford(lhs) + toClifford(rhs)) # ««««««««
08588 *
08589 * def __iadd__(self, rhs):
08590 */
08591 __Pyx_XDECREF(__pyx_r);
08592 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClicl_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 749, __pyx_L1_error)
08593 __Pyx_GOTREF(__pyx_t_1);
08594 __pyx_t_2 = __pyx_f_8PyClicl_8clifford_wrap(((struct __pyx_obj_8PyClicl_clifford
*)__pyx_t_1), (__pyx_f_8PyClicl_toClifford(__pyx_v_lhs) +
__pyx_f_8PyClicl_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 749,
__pyx_L1_error)
08595 __Pyx_GOTREF(__pyx_t_2);
08596 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
08597 __pyx_r = __pyx_t_2;
08598 __pyx_t_2 = 0;
08599 goto __pyx_L0;
08600
08601 /* "PyClicl.pyx":740
08602 * return clifford(self)
08603 *
08604 * def __add__(lhs, rhs): # ««««««««
08605 * """
08606 * Geometric sum.
08607 */
08608
08609 /* function exit code */
08610 __pyx_L1_error:;
08611 __Pyx_XDECREF(__pyx_t_1);
08612 __Pyx_XDECREF(__pyx_t_2);
08613 __Pyx_AddTraceback("PyClicl.clifford.__add__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
08614 __pyx_r = NULL;
08615 __pyx_L0:;
08616 __Pyx_XGIVEREF(__pyx_r);
08617 __Pyx_RefNannyFinishContext();
08618 return __pyx_r;
08619 }
08620
08621 /* "PyClicl.pyx":751
08622 * return clifford().wrap(toClifford(lhs) + toClifford(rhs))
08623 *
08624 * def __iadd__(self, rhs): # ««««««««
08625 * """
08626 * Geometric sum.
08627 */
08628
08629 /* Python wrapper */
08630 static PyObject *__pyx_pw_8PyClicl_8clifford_23__iadd__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
08631 static char __pyx_doc_8PyClicl_8clifford_22__iadd__[] = "\n Geometric sum.\n\n
>> x = clifford(1); x += clifford('{2}'); print(x)\n 1+{2}\n ";
08632 #if CYTHON_COMPILING_IN_CPYTHON
08633 struct wrapperbase __pyx_wrapperbase_8PyClicl_8clifford_22__iadd__;
08634 #endif
08635 static PyObject *__pyx_pw_8PyClicl_8clifford_23__iadd__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
08636 PyObject *__pyx_r = 0;
08637 __Pyx_RefNannyDeclarations
08638 __Pyx_RefNannySetupContext("__iadd__ (wrapper)", 0);
08639 __pyx_r = __pyx_pf_8PyClicl_8clifford_22__iadd__(((struct __pyx_obj_8PyClicl_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));
08640
08641 /* function exit code */
08642 __Pyx_RefNannyFinishContext();
08643 return __pyx_r;
08644 }
08645
08646 static PyObject *__pyx_pf_8PyClicl_8clifford_22__iadd__(struct __pyx_obj_8PyClicl_clifford
*__pyx_v_self, PyObject *__pyx_v_rhs) {
08647 PyObject *__pyx_r = NULL;
08648 __Pyx_RefNannyDeclarations
08649 PyObject *__pyx_t_1 = NULL;
08650 int __pyx_lineno = 0;

```

```

08651 const char *__pyx_filename = NULL;
08652 int __pyx_clineno = 0;
08653 __Pyx_RefNannySetupContext("__iadd__", 0);
08654
08655 /* "PyClical.pyx":758
08656 * 1+{2}
08657 * """
08658 * return self.wrap(self.unwrap() + toClifford(rhs)) # ««««««««
08659 *
08660 * def __sub__(lhs, rhs):
08661 */
08662 __Pyx_XDECREF(__pyx_r);
08663 __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self,
08664 (__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) + __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if
08665 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 758, __pyx_L1_error)
08666 __Pyx_GOTREF(__pyx_t_1);
08667 __pyx_r = __pyx_t_1;
08668 __pyx_t_1 = 0;
08669 goto __pyx_L0;
08670
08671 /* "PyClical.pyx":751
08672 * return clifford().wrap(toClifford(lhs) + toClifford(rhs))
08673 *
08674 * def __iadd__(self, rhs): # ««««««««
08675 * """
08676 * Geometric sum.
08677 */
08678 /* function exit code */
08679 __pyx_L1_error++;
08680 __Pyx_XDECREF(__pyx_t_1);
08681 __Pyx_AddTraceback("PyClical.clifford.__iadd__", __pyx_clineno, __pyx_lineno,
08682 __pyx_filename);
08683 __pyx_r = NULL;
08684 __pyx_L0++;
08685 __Pyx_XGIVEREF(__pyx_r);
08686 __Pyx_RefNannyFinishContext();
08687 return __pyx_r;
08688 }
08689
08690 /* "PyClical.pyx":760
08691 * return self.wrap(self.unwrap() + toClifford(rhs))
08692 *
08693 * def __sub__(lhs, rhs): # ««««««««
08694 * """
08695 * Geometric difference.
08696 */
08697 /* Python wrapper */
08698 static PyObject *__pyx_pw_8PyClical_8clifford_25__sub__(PyObject *__pyx_v_lhs, PyObject
08699 *__pyx_v_rhs); /*proto*/
08700 static char __pyx_doc_8PyClical_8clifford_24__sub__[] = "\n Geometric difference.\n\n
08701 >> print(clifford(1) - clifford(\{2\})\n\n 1-2}\n >> print(clifford(\{1\}) -
08702 clifford(\{2\})\n\n {1}-{2}\n ";
08703 #if CYTHON_COMPILING_IN_CPYTHON
08704 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_24__sub__;
08705 #endif
08706 static PyObject *__pyx_pw_8PyClical_8clifford_25__sub__(PyObject *__pyx_v_lhs, PyObject
08707 *__pyx_v_rhs) {
08708 PyObject *__pyx_r = 0;
08709 __Pyx_RefNannyDeclarations
08710 __Pyx_RefNannySetupContext("__sub__ (wrapper)", 0);
08711 __pyx_r = __pyx_pf_8PyClical_8clifford_24__sub__(((PyObject *)__pyx_v_lhs), ((PyObject
08712 *)__pyx_v_rhs));
08713
08714 /* function exit code */
08715 __Pyx_RefNannyFinishContext();
08716 return __pyx_r;
08717 }
08718
08719 static PyObject *__pyx_pf_8PyClical_8clifford_24__sub__(PyObject *__pyx_v_lhs, PyObject
08720 *__pyx_v_rhs) {
08721 PyObject *__pyx_r = NULL;
08722 __Pyx_RefNannyDeclarations
08723 PyObject *__pyx_t_1 = NULL;
08724 PyObject *__pyx_t_2 = NULL;
08725 int __pyx_lineno = 0;
08726 const char *__pyx_filename = NULL;
08727 int __pyx_clineno = 0;
08728 __Pyx_RefNannySetupContext("__sub__", 0);
08729
08730 /* "PyClical.pyx":769
08731 * {1}-{2}
08732 * """
08733 * return clifford().wrap(toClifford(lhs) - toClifford(rhs)) # ««««««««
08734 *
08735 * def __isub__(self, rhs):

```

```

08729 */
08730 __Pyx_XDECREF(__pyx_r);
08731 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 769, __pyx_L1_error)
08732 __Pyx_GOTREF(__pyx_t_1);
08733 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), (__pyx_f_8PyClical_toClifford(__pyx_v_lhs) -
__pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 769,
__pyx_L1_error)
08734 __Pyx_GOTREF(__pyx_t_2);
08735 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
08736 __pyx_r = __pyx_t_2;
08737 __pyx_t_2 = 0;
08738 goto __pyx_L0;
08739
08740 /* "PyClical.pyx":760
08741 * return self.wrap(self.unwrap() + toClifford(rhs))
08742 *
08743 * def __sub__(lhs, rhs): # ««««««««
08744 * """
08745 * Geometric difference.
08746 */
08747
08748 /* function exit code */
08749 __pyx_L1_error:;
08750 __Pyx_XDECREF(__pyx_t_1);
08751 __Pyx_XDECREF(__pyx_t_2);
08752 __Pyx_AddTraceback("PyClical.clifford.__sub__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
08753 __pyx_r = NULL;
08754 __pyx_L0:;
08755 __Pyx_XGIVEREF(__pyx_r);
08756 __Pyx_RefNannyFinishContext();
08757 return __pyx_r;
08758 }
08759
08760 /* "PyClical.pyx":771
08761 * return clifford().wrap(toClifford(lhs) - toClifford(rhs))
08762 *
08763 * def __isub__(self, rhs): # ««««««««
08764 * """
08765 * Geometric difference.
08766 */
08767
08768 /* Python wrapper */
08769 static PyObject *__pyx_pw_8PyClical_8clifford_27__isub__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
08770 static char __pyx_doc_8PyClical_8clifford_26__isub__[] = "\n Geometric difference.\n\n
>> x = clifford(1); x -= clifford(\'{2}\'); print(x)\n 1-{2}\n ";
08771 #if CYTHON_COMPILING_IN_CPYTHON
08772 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_26__isub__;
08773 #endif
08774 static PyObject *__pyx_pw_8PyClical_8clifford_27__isub__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
08775 PyObject *__pyx_r = 0;
08776 __Pyx_RefNannyDeclarations
08777 __Pyx_RefNannySetupContext("__isub__ (wrapper)", 0);
08778 __pyx_r = __pyx_pf_8PyClical_8clifford_26__isub__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));
08779
08780 /* function exit code */
08781 __Pyx_RefNannyFinishContext();
08782 return __pyx_r;
08783 }
08784
08785 static PyObject *__pyx_pf_8PyClical_8clifford_26__isub__(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self, PyObject *__pyx_v_rhs) {
08786 PyObject *__pyx_r = NULL;
08787 __Pyx_RefNannyDeclarations
08788 PyObject *__pyx_t_1 = NULL;
08789 int __pyx_lineno = 0;
08790 const char *__pyx_filename = NULL;
08791 int __pyx_clineno = 0;
08792 __Pyx_RefNannySetupContext("__isub__", 0);
08793
08794 /* "PyClical.pyx":778
08795 * 1-{2}
08796 * """
08797 * return self.wrap(self.unwrap() - toClifford(rhs)) # ««««««««
08798 *
08799 * def __mul__(lhs, rhs):
08800 */
08801 __Pyx_XDECREF(__pyx_r);
08802 __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self,
(__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) - __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 778, __pyx_L1_error)
08803 __Pyx_GOTREF(__pyx_t_1);

```

```

08804 __pyx_r = __pyx_t_1;
08805 __pyx_t_1 = 0;
08806 goto __pyx_L0;
08807
08808 /* "PyClical.pyx":771
08809 * return clifford().wrap(toClifford(lhs) - toClifford(rhs))
08810 *
08811 * def __isub__(self, rhs): # ««««««««
08812 * """
08813 * Geometric difference.
08814 */
08815
08816 /* function exit code */
08817 __pyx_L1_error++;
08818 __Pyx_XDECREF(__pyx_t_1);
08819 __Pyx_AddTraceback("PyClical.clifford.__isub__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
08820 __pyx_r = NULL;
08821 __pyx_L0++;
08822 __Pyx_XGIVEREF(__pyx_r);
08823 __Pyx_RefNannyFinishContext();
08824 return __pyx_r;
08825 }
08826
08827 /* "PyClical.pyx":780
08828 * return self.wrap(self.unwrap() - toClifford(rhs))
08829 *
08830 * def __mul__(lhs, rhs): # ««««««««
08831 * """
08832 * Geometric product.
08833 */
08834
08835 /* Python wrapper */
08836 static PyObject * __pyx_pw_8PyClical_8clifford_29__mul__(PyObject * __pyx_v_lhs, PyObject
__pyx_v_rhs); /*proto*/
08837 static char __pyx_doc_8PyClical_8clifford_28__mul__[] = "\n Geometric product.\n\n
>> print(clifford(\"{1}\") * clifford(\"{2}\"))\n {1,2}\n >> print(clifford(2) *
clifford(\"{2}\"))\n 2{2}\n >> print(clifford(\"{1}\") * clifford(\"{1,2}\"))\n
{2}\n ";
08838 #if CYTHON_COMPILING_IN_CPYTHON
08839 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_28__mul__;
08840 #endif
08841 static PyObject * __pyx_pw_8PyClical_8clifford_29__mul__(PyObject * __pyx_v_lhs, PyObject
__pyx_v_rhs) {
08842 PyObject * __pyx_r = 0;
08843 __Pyx_RefNannyDeclarations
08844 __Pyx_RefNannySetupContext("__mul__ (wrapper)", 0);
08845 __pyx_r = __pyx_pf_8PyClical_8clifford_28__mul__(((PyObject *) __pyx_v_lhs), ((PyObject
*) __pyx_v_rhs));
08846
08847 /* function exit code */
08848 __Pyx_RefNannyFinishContext();
08849 return __pyx_r;
08850 }
08851
08852 static PyObject * __pyx_pf_8PyClical_8clifford_28__mul__(PyObject * __pyx_v_lhs, PyObject
__pyx_v_rhs) {
08853 PyObject * __pyx_r = NULL;
08854 __Pyx_RefNannyDeclarations
08855 PyObject * __pyx_t_1 = NULL;
08856 PyObject * __pyx_t_2 = NULL;
08857 int __pyx_lineno = 0;
08858 const char * __pyx_filename = NULL;
08859 int __pyx_clineno = 0;
08860 __Pyx_RefNannySetupContext("__mul__", 0);
08861
08862 /* "PyClical.pyx":791
08863 * {2}
08864 * """
08865 * return clifford().wrap(toClifford(lhs) * toClifford(rhs))
08866 *
08867 * def __imul__(self, rhs):
08868 */
08869 __Pyx_XDECREF(__pyx_r);
08870 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *) __pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 791, __pyx_L1_error)
08871 __Pyx_GOTREF(__pyx_t_1);
08872 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*) __pyx_t_1), (__pyx_f_8PyClical_toClifford(__pyx_v_lhs) *
__pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 791,
__pyx_L1_error)
08873 __Pyx_GOTREF(__pyx_t_2);
08874 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
08875 __pyx_r = __pyx_t_2;
08876 __pyx_t_2 = 0;
08877 goto __pyx_L0;
08878

```

```

08879 /* "PyClicl.pyx":780
08880 * return self.wrap(self.unwrap() - toClifford(rhs))
08881 *
08882 * def __mul__(lhs, rhs): # ««««««««
08883 * """
08884 * Geometric product.
08885 */
08886
08887 /* function exit code */
08888 __pyx_L1_error:;
08889 __Pyx_XDECREF(__pyx_t_1);
08890 __Pyx_XDECREF(__pyx_t_2);
08891 __Pyx_AddTraceback("PyClicl.clifford.__mul__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
08892 __pyx_r = NULL;
08893 __pyx_L0:;
08894 __Pyx_XGIVEREF(__pyx_r);
08895 __Pyx_RefNannyFinishContext();
08896 return __pyx_r;
08897 }
08898
08899 /* "PyClicl.pyx":793
08900 * return clifford().wrap(toClifford(lhs) * toClifford(rhs))
08901 *
08902 * def __imul__(self, rhs): # ««««««««
08903 * """
08904 * Geometric product.
08905 */
08906
08907 /* Python wrapper */
08908 static PyObject *__pyx_pw_8PyClicl_8clifford_31__imul__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
08909 static char __pyx_doc_8PyClicl_8clifford_30__imul__[] = "\n Geometric product.\n\n
>> x = clifford(2); x *= clifford('{2}'); print(x)\n 2{2}\n >> x = clifford('{1}');
x *= clifford('{2}'); print(x)\n {1,2}\n >> x = clifford('{1}'); x *=
clifford('{1,2}'); print(x)\n {2}\n ";
08910 #if CYTHON_COMPILING_IN_CPYTHON
08911 struct wrapperbase __pyx_wrapperbase_8PyClicl_8clifford_30__imul__;
08912 #endif
08913 static PyObject *__pyx_pw_8PyClicl_8clifford_31__imul__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
08914 PyObject *__pyx_r = 0;
08915 __Pyx_RefNannyDeclarations
08916 __Pyx_RefNannySetupContext("__imul__ (wrapper)", 0);
08917 __pyx_r = __pyx_pf_8PyClicl_8clifford_30__imul__(((struct __pyx_obj_8PyClicl_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));
08918
08919 /* function exit code */
08920 __Pyx_RefNannyFinishContext();
08921 return __pyx_r;
08922 }
08923
08924 static PyObject *__pyx_pf_8PyClicl_8clifford_30__imul__(struct __pyx_obj_8PyClicl_clifford
*__pyx_v_self, PyObject *__pyx_v_rhs) {
08925 PyObject *__pyx_r = NULL;
08926 __Pyx_RefNannyDeclarations
08927 PyObject *__pyx_t_1 = NULL;
08928 int __pyx_lineno = 0;
08929 const char *__pyx_filename = NULL;
08930 int __pyx_clineno = 0;
08931 __Pyx_RefNannySetupContext("__imul__", 0);
08932
08933 /* "PyClicl.pyx":804
08934 * {2}
08935 * """
08936 * return self.wrap(self.unwrap() * toClifford(rhs)) # ««««««««
08937 *
08938 * def __mod__(lhs, rhs):
08939 */
08940 __Pyx_XDECREF(__pyx_r);
08941 __pyx_t_1 = __pyx_f_8PyClicl_8clifford_wrap(__pyx_v_self,
(__pyx_f_8PyClicl_8clifford_unwrap(__pyx_v_self) * __pyx_f_8PyClicl_toClifford(__pyx_v_rhs))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 804, __pyx_L1_error)
08942 __Pyx_GOTREF(__pyx_t_1);
08943 __pyx_r = __pyx_t_1;
08944 __pyx_t_1 = 0;
08945 goto __pyx_L0;
08946
08947 /* "PyClicl.pyx":793
08948 * return clifford().wrap(toClifford(lhs) * toClifford(rhs))
08949 *
08950 * def __imul__(self, rhs): # ««««««««
08951 * """
08952 * Geometric product.
08953 */
08954
08955 /* function exit code */

```



```

08956 __pyx_L1_error++;
08957 __Pyx_XDECREF(__pyx_t_1);
08958 __Pyx_AddTraceback("PyClicl.clifford.__imul__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
08959 __pyx_r = NULL;
08960 __pyx_L0++;
08961 __Pyx_XGIVEREF(__pyx_r);
08962 __Pyx_RefNannyFinishContext();
08963 return __pyx_r;
08964 }
08965
08966 /* "PyClicl.pyx":806
08967 * return self.wrap(self.unwrap() * toClifford(rhs))
08968 *
08969 * def __mod__(lhs, rhs): # ««««««««
08970 * """
08971 * Contraction.
08972 */
08973
08974 /* Python wrapper */
08975 static PyObject * __pyx_pw_8PyClicl_8clifford_33__mod__(PyObject * __pyx_v_lhs, PyObject
* __pyx_v_rhs); /*proto*/
08976 static char __pyx_doc_8PyClicl_8clifford_32__mod__[] = "\n Contraction.\n\n >>
print(clifford(\"{1}\") % clifford(\"{2}\"))\n 0\n >> print(clifford(2) %
clifford(\"{2}\"))\n 2{2}\n >> print(clifford(\"{1}\") % clifford(\"{1}\"))\n
1\n >> print(clifford(\"{1}\") % clifford(\"{1,2}\"))\n {2}\n ";
08977 #if CYTHON_COMPILING_IN_CPYTHON
08978 struct wrapperbase __pyx_wrapperbase_8PyClicl_8clifford_32__mod__;
08979 #endif
08980 static PyObject * __pyx_pw_8PyClicl_8clifford_33__mod__(PyObject * __pyx_v_lhs, PyObject
* __pyx_v_rhs) {
08981 PyObject * __pyx_r = 0;
08982 __Pyx_RefNannyDeclarations
08983 __Pyx_RefNannySetupContext("__mod__ (wrapper)", 0);
08984 __pyx_r = __pyx_pf_8PyClicl_8clifford_32__mod__(((PyObject *) __pyx_v_lhs), ((PyObject
*) __pyx_v_rhs));
08985
08986 /* function exit code */
08987 __Pyx_RefNannyFinishContext();
08988 return __pyx_r;
08989 }
08990
08991 static PyObject * __pyx_pf_8PyClicl_8clifford_32__mod__(PyObject * __pyx_v_lhs, PyObject
* __pyx_v_rhs) {
08992 PyObject * __pyx_r = NULL;
08993 __Pyx_RefNannyDeclarations
08994 PyObject * __pyx_t_1 = NULL;
08995 PyObject * __pyx_t_2 = NULL;
08996 int __pyx_lineno = 0;
08997 const char * __pyx_filename = NULL;
08998 int __pyx_clineno = 0;
08999 __Pyx_RefNannySetupContext("__mod__", 0);
09000
09001 /* "PyClicl.pyx":819
09002 * {2}
09003 * """
09004 * return clifford().wrap(toClifford(lhs) % toClifford(rhs)) # ««««««««
09005 *
09006 * def __imod__(self, rhs):
09007 */
09008 __Pyx_XDECREF(__pyx_r);
09009 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *) __pyx_ptype_8PyClicl_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 819, __pyx_L1_error)
09010 __Pyx_GOTREF(__pyx_t_1);
09011 __pyx_t_2 = __pyx_f_8PyClicl_8clifford_wrap(((struct __pyx_obj_8PyClicl_clifford
*) __pyx_t_1), (__pyx_f_8PyClicl_toClifford(__pyx_v_lhs) %
__pyx_f_8PyClicl_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 819,
__pyx_L1_error)
09012 __Pyx_GOTREF(__pyx_t_2);
09013 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
09014 __pyx_r = __pyx_t_2;
09015 __pyx_t_2 = 0;
09016 goto __pyx_L0;
09017
09018 /* "PyClicl.pyx":806
09019 * return self.wrap(self.unwrap() * toClifford(rhs))
09020 *
09021 * def __mod__(lhs, rhs): # ««««««««
09022 * """
09023 * Contraction.
09024 */
09025
09026 /* function exit code */
09027 __pyx_L1_error++;
09028 __Pyx_XDECREF(__pyx_t_1);
09029 __Pyx_XDECREF(__pyx_t_2);
09030 __Pyx_AddTraceback("PyClicl.clifford.__mod__", __pyx_clineno, __pyx_lineno,

```

```

__pyx_filename);
09031 __pyx_r = NULL;
09032 __pyx_L0:;
09033 __Pyx_XGIVEREF(__pyx_r);
09034 __Pyx_RefNannyFinishContext();
09035 return __pyx_r;
09036 }
09037
09038 /* "PyClicl.pyx":821
09039 * return clifford().wrap(toClifford(lhs) % toClifford(rhs))
09040 *
09041 * def __imod__(self, rhs): # ««««««««
09042 * """
09043 * Contraction.
09044 */
09045
09046 /* Python wrapper */
09047 static PyObject * __pyx_pw_8PyClicl_8clifford_35__imod__(PyObject * __pyx_v_self, PyObject
* __pyx_v_rhs); /*proto*/
09048 static char __pyx_doc_8PyClicl_8clifford_34__imod__[] = "\n Contraction.\n\n >>
x = clifford(\"{1}\"); x %= clifford(\"{2}\"); print(x)\n 0\n >> x = clifford(2); x
%= clifford(\"{2}\"); print(x)\n 2{2}\n >> x = clifford(\"{1}\"); x %= clifford(\"{1}\");
print(x)\n 1\n >> x = clifford(\"{1}\"); x %= clifford(\"{1,2}\"); print(x)\n
{2}\n ";
09049 #if CYTHON_COMPILING_IN_CPYTHON
09050 struct wrapperbase __pyx_wrapperbase_8PyClicl_8clifford_34__imod__;
09051 #endif
09052 static PyObject * __pyx_pw_8PyClicl_8clifford_35__imod__(PyObject * __pyx_v_self, PyObject
* __pyx_v_rhs) {
09053 PyObject * __pyx_r = 0;
09054 __Pyx_RefNannyDeclarations
09055 __Pyx_RefNannySetupContext("__imod__ (wrapper)", 0);
09056 __pyx_r = __pyx_pf_8PyClicl_8clifford_34__imod__(((struct __pyx_obj_8PyClicl_clifford
*) __pyx_v_self), ((PyObject *) __pyx_v_rhs));
09057
09058 /* function exit code */
09059 __Pyx_RefNannyFinishContext();
09060 return __pyx_r;
09061 }
09062
09063 static PyObject * __pyx_pf_8PyClicl_8clifford_34__imod__(struct __pyx_obj_8PyClicl_clifford
* __pyx_v_self, PyObject * __pyx_v_rhs) {
09064 PyObject * __pyx_r = NULL;
09065 __Pyx_RefNannyDeclarations
09066 PyObject * __pyx_t_1 = NULL;
09067 int __pyx_lineno = 0;
09068 const char * __pyx_filename = NULL;
09069 int __pyx_clineno = 0;
09070 __Pyx_RefNannySetupContext("__imod__", 0);
09071
09072 /* "PyClicl.pyx":834
09073 * {2}
09074 * """
09075 * return self.wrap(self.unwrap() % toClifford(rhs)) # ««««««««
09076 *
09077 * def __and__(lhs, rhs):
09078 */
09079 __Pyx_XDECREF(__pyx_r);
09080 __pyx_t_1 = __pyx_f_8PyClicl_8clifford_wrap(__pyx_v_self,
(__pyx_f_8PyClicl_8clifford_unwrap(__pyx_v_self) % __pyx_f_8PyClicl_toClifford(__pyx_v_rhs))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 834, __pyx_L1_error)
09081 __Pyx_GOTREF(__pyx_t_1);
09082 __pyx_r = __pyx_t_1;
09083 __pyx_t_1 = 0;
09084 goto __pyx_L0;
09085
09086 /* "PyClicl.pyx":821
09087 * return clifford().wrap(toClifford(lhs) % toClifford(rhs))
09088 *
09089 * def __imod__(self, rhs): # ««««««««
09090 * """
09091 * Contraction.
09092 */
09093
09094 /* function exit code */
09095 __pyx_L1_error:;
09096 __Pyx_XDECREF(__pyx_t_1);
09097 __Pyx_AddTraceback("PyClicl.clifford.__imod__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
09098 __pyx_r = NULL;
09099 __pyx_L0:;
09100 __Pyx_XGIVEREF(__pyx_r);
09101 __Pyx_RefNannyFinishContext();
09102 return __pyx_r;
09103 }
09104
09105 /* "PyClicl.pyx":836

```

Generated by Doxygen

```

09181 * """
09182 * Inner product.
09183 */
09184
09185 /* Python wrapper */
09186 static PyObject *__pyx_pw_8PyClical_8clifford_39__iand__(PyObject *__pyx_v_self, PyObject
09187 *__pyx_v_rhs); /*proto*/
09188 static char __pyx_doc_8PyClical_8clifford_38__iand__[] = "\n Inner product.\n\n
>> x = clifford(\"{1}\"); x &= clifford(\"{2}\"); print(x)\n 0\n >> x = clifford(2); x
&= clifford(\"{2}\"); print(x)\n 0\n >> x = clifford(\"{1}\"); x &= clifford(\"{1}\");
print(x)\n 1\n >> x = clifford(\"{1}\"); x &= clifford(\"{1,2}\"); print(x)\n
{2}\n ";
09189 #if CYTHON_COMPILING_IN_CPYTHON
09190 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_38__iand__;
09191 #endif
09192 static PyObject *__pyx_pw_8PyClical_8clifford_39__iand__(PyObject *__pyx_v_self, PyObject
09193 *__pyx_v_rhs) {
09194 PyObject *__pyx_r = 0;
09195 __Pyx_RefNannyDeclarations
09196 __Pyx_RefNannySetupContext("__iand__ (wrapper)", 0);
09197 __pyx_r = __pyx_pf_8PyClical_8clifford_38__iand__((struct __pyx_obj_8PyClical_clifford
09198 *)__pyx_v_self), ((PyObject *)__pyx_v_rhs));
09199
09200 /* function exit code */
09201 __Pyx_RefNannyFinishContext();
09202 return __pyx_r;
09203 }
09204
09205 static PyObject *__pyx_pf_8PyClical_8clifford_38__iand__(struct __pyx_obj_8PyClical_clifford
09206 *__pyx_v_self, PyObject *__pyx_v_rhs) {
09207 PyObject *__pyx_r = NULL;
09208 __Pyx_RefNannyDeclarations
09209 PyObject *__pyx_t_1 = NULL;
09210 int __pyx_lineno = 0;
09211 const char *__pyx_filename = NULL;
09212 int __pyx_clineno = 0;
09213 __Pyx_RefNannySetupContext("__iand__", 0);
09214
09215 /* "PyClical.pyx":864
09216 {
09217 """
09218 return self.wrap(self.unwrap() & toClifford(rhs)) # ««««««««
09219
09220 def __xor__(lhs, rhs):
09221 */
09222 __Pyx_XDECREF(__pyx_r);
09223 __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self,
09224 (__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) & __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if
09225 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 864, __pyx_L1_error)
09226 __Pyx_GOTREF(__pyx_t_1);
09227 __pyx_r = __pyx_t_1;
09228 __pyx_t_1 = 0;
09229 goto __pyx_L0;
09230
09231 /* "PyClical.pyx":851
09232 return clifford().wrap(toClifford(lhs) & toClifford(rhs))
09233
09234 def __iand__(self, rhs): # ««««««««
09235 """
09236 Inner product.
09237 */
09238
09239 /* function exit code */
09240 __pyx_L1_error:;
09241 __Pyx_XDECREF(__pyx_t_1);
09242 __Pyx_AddTraceback("PyClical.clifford.__iand__", __pyx_clineno, __pyx_lineno,
09243 __pyx_filename);
09244 __pyx_r = NULL;
09245 __pyx_L0:;
09246 __Pyx_XGIVEREF(__pyx_r);
09247 __Pyx_RefNannyFinishContext();
09248 return __pyx_r;
09249 }
09250
09251 /* "PyClical.pyx":866
09252 return self.wrap(self.unwrap() & toClifford(rhs))
09253
09254 def __xor__(lhs, rhs): # ««««««««
09255 """
09256 Outer product.
09257 */
09258
09259 /* Python wrapper */
09260 static PyObject *__pyx_pw_8PyClical_8clifford_41__xor__(PyObject *__pyx_v_lhs, PyObject
09261 *__pyx_v_rhs); /*proto*/
09262 static char __pyx_doc_8PyClical_8clifford_40__xor__[] = "\n Outer product.\n\n
>> print(clifford(\"{1}\") ^ clifford(\"{2}\"))\n {1,2}\n >> print(clifford(2) ^

```

```

 clifford("\{2}\")\n 2{2}\n >> print(clifford("\{1}\") ^ clifford("\{1}\"))\n 0\n
>> print(clifford("\{1}\") ^ clifford("\{1,2}\"))\n 0\n ";
09255 #if CYTHON_COMPILING_IN_CPYTHON
09256 struct wrapperbase __pyx_wrapperbase_8PyClicl_8clifford_40__xor__;
09257 #endif
09258 static PyObject *__pyx_pw_8PyClicl_8clifford_41__xor__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs) {
09259 PyObject *__pyx_r = 0;
09260 __Pyx_RefNannyDeclarations
09261 __Pyx_RefNannySetupContext("__xor__ (wrapper)", 0);
09262 __pyx_r = __pyx_pf_8PyClicl_8clifford_40__xor__((PyObject *)__pyx_v_lhs), ((PyObject
*)__pyx_v_rhs));
09263
09264 /* function exit code */
09265 __Pyx_RefNannyFinishContext();
09266 return __pyx_r;
09267 }
09268
09269 static PyObject *__pyx_pf_8PyClicl_8clifford_40__xor__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs) {
09270 PyObject *__pyx_r = NULL;
09271 __Pyx_RefNannyDeclarations
09272 PyObject *__pyx_t_1 = NULL;
09273 PyObject *__pyx_t_2 = NULL;
09274 int __pyx_lineno = 0;
09275 const char *__pyx_filename = NULL;
09276 int __pyx_clineno = 0;
09277 __Pyx_RefNannySetupContext("__xor__", 0);
09278
09279 /* "PyClicl.pyx":879
09280 *
09281 *
09282 * return clifford().wrap(toClifford(lhs) ^ toClifford(rhs))
09283 *
09284 * def __ixor__(self, rhs):
09285 */
09286 __Pyx_XDECREF(__pyx_r);
09287 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClicl_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 879, __pyx_L1_error)
09288 __Pyx_GOTREF(__pyx_t_1);
09289 __pyx_t_2 = __pyx_f_8PyClicl_8clifford_wrap(((struct __pyx_obj_8PyClicl_clifford
*)__pyx_t_1), (__pyx_f_8PyClicl_toClifford(__pyx_v_lhs) ^
__pyx_f_8PyClicl_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 879,
__pyx_L1_error)
09290 __Pyx_GOTREF(__pyx_t_2);
09291 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
09292 __pyx_r = __pyx_t_2;
09293 __pyx_t_2 = 0;
09294 goto __pyx_L0;
09295
09296 /* "PyClicl.pyx":866
09297 * return self.wrap(self.unwrap() & toClifford(rhs))
09298 *
09299 * def __xor__(lhs, rhs):
09300 *
09301 * Outer product.
09302 */
09303
09304 /* function exit code */
09305 __pyx_L1_error;
09306 __Pyx_XDECREF(__pyx_t_1);
09307 __Pyx_XDECREF(__pyx_t_2);
09308 __Pyx_AddTraceback("PyClicl.clifford.__xor__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
09309 __pyx_r = NULL;
09310 __pyx_L0;
09311 __Pyx_XGIVEREF(__pyx_r);
09312 __Pyx_RefNannyFinishContext();
09313 return __pyx_r;
09314 }
09315
09316 /* "PyClicl.pyx":881
09317 * return clifford().wrap(toClifford(lhs) ^ toClifford(rhs))
09318 *
09319 * def __ixor__(self, rhs):
09320 *
09321 * Outer product.
09322 */
09323
09324 /* Python wrapper */
09325 static PyObject *__pyx_pw_8PyClicl_8clifford_43__ixor__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
09326 static char __pyx_doc_8PyClicl_8clifford_42__ixor__[] = "\n Outer product.\n\n
>> x = clifford("\{1}\"); x ^= clifford("\{2}\"); print(x)\n {1,2}\n >> x = clifford(2);
x ^= clifford("\{2}\"); print(x)\n 2{2}\n >> x = clifford("\{1}\"); x ^=
clifford("\{1}\"); print(x)\n 0\n >> x = clifford("\{1}\"); x ^= clifford("\{1,2}\");
print(x)\n 0\n ";

```

```

09327 #if CYTHON_COMPILING_IN_CPYTHON
09328 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_42__ixor__;
09329 #endif
09330 static PyObject *__pyx_pw_8PyClical_8clifford_43__ixor__(PyObject *__pyx_v_self, PyObject
09331 *__pyx_v_rhs) {
09332 PyObject *__pyx_r = 0;
09333 __Pyx_RefNannyDeclarations
09334 __Pyx_RefNannySetupContext("__ixor__ (wrapper)", 0);
09335 __pyx_r = __pyx_pf_8PyClical_8clifford_42__ixor__(((struct __pyx_obj_8PyClical_clifford
09336 *)__pyx_v_self), ((PyObject *)__pyx_v_rhs));
09337
09338 /* function exit code */
09339 __Pyx_RefNannyFinishContext();
09340 return __pyx_r;
09341 }
09342
09343 static PyObject *__pyx_pf_8PyClical_8clifford_42__ixor__(struct __pyx_obj_8PyClical_clifford
09344 *__pyx_v_self, PyObject *__pyx_v_rhs) {
09345 PyObject *__pyx_r = NULL;
09346 __Pyx_RefNannyDeclarations
09347 PyObject *__pyx_t_1 = NULL;
09348 int __pyx_lineno = 0;
09349 const char *__pyx_filename = NULL;
09350 int __pyx_clineno = 0;
09351 __Pyx_RefNannySetupContext("__ixor__", 0);
09352
09353 /* "PyClical.pyx":894
09354 *
09355 * return self.wrap(self.unwrap() ^ toClifford(rhs)) # ««««««««
09356 *
09357 * def __truediv__(lhs, rhs):
09358 */
09359 __Pyx_XDECREF(__pyx_r);
09360 __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self,
09361 (__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) ^ __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if
09362 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 894, __pyx_L1_error)
09363 __Pyx_GOTREF(__pyx_t_1);
09364 __pyx_r = __pyx_t_1;
09365 __pyx_t_1 = 0;
09366 goto __pyx_L0;
09367
09368 /* "PyClical.pyx":881
09369 * return clifford().wrap(toClifford(lhs) ^ toClifford(rhs))
09370 *
09371 * def __ixor__(self, rhs): # ««««««««
09372 * """
09373 * Outer product.
09374 */
09375 /* function exit code */
09376 __pyx_L1_error:;
09377 __Pyx_XDECREF(__pyx_t_1);
09378 __Pyx_AddTraceback("PyClical.clifford.__ixor__", __pyx_clineno, __pyx_lineno,
09379 __pyx_filename);
09380 __pyx_r = NULL;
09381 __pyx_L0:;
09382 __Pyx_XGIVEREF(__pyx_r);
09383 __Pyx_RefNannyFinishContext();
09384 return __pyx_r;
09385 }
09386
09387 /* "PyClical.pyx":896
09388 * return self.wrap(self.unwrap() ^ toClifford(rhs))
09389 *
09390 * def __truediv__(lhs, rhs): # ««««««««
09391 * """
09392 * Geometric quotient.
09393 */
09394 /* Python wrapper */
09395 static PyObject *__pyx_pw_8PyClical_8clifford_45__truediv__(PyObject *__pyx_v_lhs, PyObject
09396 *__pyx_v_rhs); /*proto*/
09397 static char __pyx_doc_8PyClical_8clifford_44__truediv__[] = "\n Geometric quotient.\n\n
09398 >> print(clifford(\\"{1}\\") / clifford(\\"{2}\\"))\n {1,2}\n >> print(clifford(2) /
09399 clifford(\\"{2}\\"))\n 2{2}\n >> print(clifford(\\"{1}\\") / clifford(\\"{1}\\"))\n 1\n
09400 >> print(clifford(\\"{1}\\") / clifford(\\"{1,2}\\"))\n -{2}\n ";
09401 #if CYTHON_COMPILING_IN_CPYTHON
09402 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_44__truediv__;
09403 #endif
09404 static PyObject *__pyx_pw_8PyClical_8clifford_45__truediv__(PyObject *__pyx_v_lhs, PyObject
09405 *__pyx_v_rhs) {
09406 PyObject *__pyx_r = 0;
09407 __Pyx_RefNannyDeclarations
09408 __Pyx_RefNannySetupContext("__truediv__ (wrapper)", 0);
09409 __pyx_r = __pyx_pf_8PyClical_8clifford_44__truediv__(((PyObject *)__pyx_v_lhs), ((PyObject
09410 *)__pyx_v_rhs));

```

```

09402
09403 /* function exit code */
09404 __Pyx_RefNannyFinishContext();
09405 return __pyx_r;
09406 }
09407
09408 static PyObject *__pyx_pf_8PyClical_8clifford_44__truediv__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs) {
09409 PyObject *__pyx_r = NULL;
09410 __Pyx_RefNannyDeclarations
09411 PyObject *__pyx_t_1 = NULL;
09412 PyObject *__pyx_t_2 = NULL;
09413 int __pyx_lineno = 0;
09414 const char *__pyx_filename = NULL;
09415 int __pyx_clineno = 0;
09416 __Pyx_RefNannySetupContext("__truediv__", 0);
09417
09418 /* "PyClical.pyx":909
09419 *
09420 *
09421 * return clifford().wrap(toClifford(lhs) / toClifford(rhs))
09422 *
09423 * def __idiv__(self, rhs):
09424 */
09425 __Pyx_XDECREF(__pyx_r);
09426 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 909, __pyx_L1_error)
09427 __Pyx_GOTREF(__pyx_t_1);
09428 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), (__pyx_f_8PyClical_toClifford(__pyx_v_lhs) /
__pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 909,
__pyx_L1_error)
09429 __Pyx_GOTREF(__pyx_t_2);
09430 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
09431 __pyx_r = __pyx_t_2;
09432 __pyx_t_2 = 0;
09433 goto __pyx_L0;
09434
09435 /* "PyClical.pyx":896
09436 *
09437 * return self.wrap(self.unwrap() ^ toClifford(rhs))
09438 *
09439 * def __truediv__(lhs, rhs):
09440 *
09441 * Geometric quotient.
09442 */
09443 /* function exit code */
09444 __pyx_L1_error:;
09445 __Pyx_XDECREF(__pyx_t_1);
09446 __Pyx_XDECREF(__pyx_t_2);
09447 __Pyx_AddTraceback("PyClical.clifford.__truediv__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
09448 __pyx_r = NULL;
09449 __pyx_L0:;
09450 __Pyx_XGIVEREF(__pyx_r);
09451 __Pyx_RefNannyFinishContext();
09452 return __pyx_r;
09453 }
09454
09455 /* "PyClical.pyx":911
09456 *
09457 * return clifford().wrap(toClifford(lhs) / toClifford(rhs))
09458 *
09459 * def __idiv__(self, rhs):
09460 *
09461 * Geometric quotient.
09462 */
09463 /* Python wrapper */
09464 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
09465 static PyObject *__pyx_pw_8PyClical_8clifford_47__idiv__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
09466 static char __pyx_doc_8PyClical_8clifford_46__idiv__[] = "\n Geometric quotient.\n\n
>> x = clifford('{1}'); x /= clifford('{2}'); print(x)\n {1,2}\n >> x = clifford(2);
x /= clifford('{2}'); print(x)\n 2{2}\n >> x = clifford('{1}'); x /=
clifford('{1}'); print(x)\n 1\n >> x = clifford('{1}'); x /= clifford('{1,2}');
print(x)\n -{2}\n ";
09467 #if CYTHON_COMPILING_IN_CPYTHON
09468 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_46__idiv__;
09469 #endif
09470 static PyObject *__pyx_pw_8PyClical_8clifford_47__idiv__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
09471 PyObject *__pyx_r = 0;
09472 __Pyx_RefNannyDeclarations
09473 __Pyx_RefNannySetupContext("__idiv__ (wrapper)", 0);
09474 __pyx_r = __pyx_pf_8PyClical_8clifford_46__idiv__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));
09475

```

```

09476 /* function exit code */
09477 __Pyx_RefNannyFinishContext();
09478 return __pyx_r;
09479 }
09480 #endif
09481 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
09482 static PyObject *__pyx_pf_8PyClical_8clifford_46_idiv__(struct __pyx_obj_8PyClical_clifford
09483 *__pyx_v_self, PyObject *__pyx_v_rhs) {
09484 PyObject *__pyx_r = NULL;
09485 __Pyx_RefNannyDeclarations
09486 PyObject *__pyx_t_1 = NULL;
09487 int __pyx_lineno = 0;
09488 const char *__pyx_filename = NULL;
09489 int __pyx_clineno = 0;
09490 __Pyx_RefNannySetupContext("__idiv__", 0);
09491
09492 /* "PyClical.pyx":924
09493 *
09494 *
09495 * return self.wrap(self.unwrap() / toClifford(rhs))
09496 *
09497 * def inv(self):
09498 */
09499 __Pyx_XDECREF(__pyx_r);
09500 __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self,
09501 (__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) / __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if
09502 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 924, __pyx_L1_error)
09503 __Pyx_GOTREF(__pyx_t_1);
09504 __pyx_r = __pyx_t_1;
09505 __pyx_t_1 = 0;
09506 goto __pyx_L0;
09507
09508 /* "PyClical.pyx":911
09509 *
09510 * return clifford().wrap(toClifford(lhs) / toClifford(rhs))
09511 *
09512 * def __idiv__(self, rhs):
09513 *
09514 * """
09515 * Geometric quotient.
09516 *
09517 */
09518 /* function exit code */
09519 __pyx_L1_error:;
09520 __Pyx_XDECREF(__pyx_t_1);
09521 __Pyx_AddTraceback("PyClical.clifford.__idiv__", __pyx_clineno, __pyx_lineno,
09522 __pyx_filename);
09523 __pyx_r = NULL;
09524 __pyx_L0:;
09525 __Pyx_XGIVEREF(__pyx_r);
09526 __Pyx_RefNannyFinishContext();
09527 return __pyx_r;
09528 }
09529 #endif
09530 /* "PyClical.pyx":926
09531 *
09532 * return self.wrap(self.unwrap() / toClifford(rhs))
09533 *
09534 * def inv(self):
09535 *
09536 * """
09537 * Geometric multiplicative inverse.
09538 *
09539 */
09540 /* Python wrapper */
09541 static PyObject *__pyx_pw_8PyClical_8clifford_49inv(PyObject *__pyx_v_self, CYTHON_UNUSED
09542 PyObject *unused); /*proto*/
09543 static char __pyx_doc_8PyClical_8clifford_48inv[] = "\n Geometric multiplicative
09544 inverse.\n\n >> x = clifford('{1}'); print(x.inv())\n {1}\n >> x = clifford(2);
09545 print(x.inv())\n 0.5\n >> x = clifford('{1,2}'); print(x.inv())\n -{1,2}\n
09546 ";
09547 static PyObject *__pyx_pw_8PyClical_8clifford_49inv(PyObject *__pyx_v_self, CYTHON_UNUSED
09548 PyObject *unused) {
09549 PyObject *__pyx_r = 0;
09550 __Pyx_RefNannyDeclarations
09551 __Pyx_RefNannySetupContext("inv (wrapper)", 0);
09552 __pyx_r = __pyx_pf_8PyClical_8clifford_48inv(((struct __pyx_obj_8PyClical_clifford
09553 *)__pyx_v_self));
09554
09555 /* function exit code */
09556 __Pyx_RefNannyFinishContext();
09557 return __pyx_r;
09558 }
09559
09560 static PyObject *__pyx_pf_8PyClical_8clifford_48inv(struct __pyx_obj_8PyClical_clifford
09561 *__pyx_v_self) {
09562 PyObject *__pyx_r = NULL;
09563 __Pyx_RefNannyDeclarations
09564 PyObject *__pyx_t_1 = NULL;
09565 PyObject *__pyx_t_2 = NULL;
09566 int __pyx_lineno = 0;

```



```

09554 const char *__pyx_filename = NULL;
09555 int __pyx_clineno = 0;
09556 __Pyx_RefNannySetupContext("inv", 0);
09557
09558 /* "PyClical.pyx":937
09559 * -{1,2}
09560 * """
09561 * return clifford().wrap(self.instance.inv()) # ««««««««
09562 *
09563 * def __or__(lhs, rhs):
09564 */
09565 __Pyx_XDECREF(__pyx_r);
09566 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 937, __pyx_L1_error)
09567 __Pyx_GOTREF(__pyx_t_1);
09568 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), __pyx_v_self->instance->inv()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 937,
__pyx_L1_error)
09569 __Pyx_GOTREF(__pyx_t_2);
09570 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
09571 __pyx_r = __pyx_t_2;
09572 __pyx_t_2 = 0;
09573 goto __pyx_L0;
09574
09575 /* "PyClical.pyx":926
09576 * return self.wrap(self.unwrap() / toClifford(rhs))
09577 *
09578 * def inv(self): # ««««««««
09579 * """
09580 * Geometric multiplicative inverse.
09581 */
09582
09583 /* function exit code */
09584 __pyx_L1_error:;
09585 __Pyx_XDECREF(__pyx_t_1);
09586 __Pyx_XDECREF(__pyx_t_2);
09587 __Pyx_AddTraceback("PyClical.clifford.inv", __pyx_clineno, __pyx_lineno, __pyx_filename);
09588 __pyx_r = NULL;
09589 __pyx_L0:;
09590 __Pyx_XGIVEREF(__pyx_r);
09591 __Pyx_RefNannyFinishContext();
09592 return __pyx_r;
09593 }
09594
09595 /* "PyClical.pyx":939
09596 * return clifford().wrap(self.instance.inv())
09597 *
09598 * def __or__(lhs, rhs): # ««««««««
09599 * """
09600 * Transform left hand side, using right hand side as a transformation.
09601 */
09602
09603 /* Python wrapper */
09604 static PyObject *__pyx_pw_8PyClical_8clifford_51_or__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs); /*proto*/
09605 static char __pyx_doc_8PyClical_8clifford_50_or__[] = "\n Transform left hand side,
using right hand side as a transformation.\n\n >> x=clifford(\"{1,2}\") * pi/2;
y=clifford(\"{1}\"); print(y|x)\n -{1}\n >> x=clifford(\"{1,2}\") * pi/2;
y=clifford(\"{1}\"); print(y|exp(x))\n -{1}\n ";
09606 #if CYTHON_COMPILING_IN_CPYTHON
09607 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_50_or__;
09608 #endif
09609 static PyObject *__pyx_pw_8PyClical_8clifford_51_or__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs) {
09610 PyObject *__pyx_r = 0;
09611 __Pyx_RefNannyDeclarations
09612 __Pyx_RefNannySetupContext("__or__ (wrapper)", 0);
09613 __pyx_r = __pyx_pf_8PyClical_8clifford_50_or__(((PyObject *)__pyx_v_lhs), ((PyObject
*)__pyx_v_rhs));
09614
09615 /* function exit code */
09616 __Pyx_RefNannyFinishContext();
09617 return __pyx_r;
09618 }
09619
09620 static PyObject *__pyx_pf_8PyClical_8clifford_50_or__(PyObject *__pyx_v_lhs, PyObject
*__pyx_v_rhs) {
09621 PyObject *__pyx_r = NULL;
09622 __Pyx_RefNannyDeclarations
09623 PyObject *__pyx_t_1 = NULL;
09624 PyObject *__pyx_t_2 = NULL;
09625 int __pyx_lineno = 0;
09626 const char *__pyx_filename = NULL;
09627 int __pyx_clineno = 0;
09628 __Pyx_RefNannySetupContext("__or__", 0);
09629
09630 /* "PyClical.pyx":948

```

```

09631 * -{1}
09632 * """
09633 * return clifford().wrap(toClifford(lhs) | toClifford(rhs)) # ««««««««
09634 *
09635 * def __ior__(self, rhs):
09636 */
09637 __Pyx_XDECREF(__pyx_r);
09638 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 948, __pyx_L1_error)
09639 __Pyx_GOTREF(__pyx_t_1);
09640 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), (__pyx_f_8PyClical_toClifford(__pyx_v_lhs) |
__pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 948,
__pyx_L1_error)
09641 __Pyx_GOTREF(__pyx_t_2);
09642 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
09643 __pyx_r = __pyx_t_2;
09644 __pyx_t_2 = 0;
09645 goto __pyx_L0;
09646
09647 /* "PyClical.pyx":939
09648 * return clifford().wrap(self.instance.inv())
09649 *
09650 * def __or__(lhs, rhs): # ««««««««
09651 * """
09652 * Transform left hand side, using right hand side as a transformation.
09653 */
09654
09655 /* function exit code */
09656 __pyx_L1_error;;
09657 __Pyx_XDECREF(__pyx_t_1);
09658 __Pyx_XDECREF(__pyx_t_2);
09659 __Pyx_AddTraceback("PyClical.clifford.__or__", __pyx_clineno, __pyx_lineno, __pyx_filename);
09660 __pyx_r = NULL;
09661 __pyx_L0;;
09662 __Pyx_XGIVEREF(__pyx_r);
09663 __Pyx_RefNannyFinishContext();
09664 return __pyx_r;
09665 }
09666
09667 /* "PyClical.pyx":950
09668 * return clifford().wrap(toClifford(lhs) | toClifford(rhs))
09669 *
09670 * def __ior__(self, rhs): # ««««««««
09671 * """
09672 * Transform left hand side, using right hand side as a transformation.
09673 */
09674
09675 /* Python wrapper */
09676 static PyObject *__pyx_pw_8PyClical_8clifford_53__ior__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs); /*proto*/
09677 static char __pyx_doc_8PyClical_8clifford_52__ior__[] = "\n Transform left hand side,
using right hand side as a transformation.\n\n >>> x=clifford('{1,2}\n') * pi/2;
y=clifford('{1}\n'); y|=x; print(y)\n -{1}\n >>> x=clifford('{1,2}\n') * pi/2;
y=clifford('{1}\n'); y|=exp(x); print(y)\n -{1}\n ";
09678 #if CYTHON_COMPILING_IN_CPYTHON
09679 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_52__ior__;
09680 #endif
09681 static PyObject *__pyx_pw_8PyClical_8clifford_53__ior__(PyObject *__pyx_v_self, PyObject
*__pyx_v_rhs) {
09682 PyObject *__pyx_r = 0;
09683 __Pyx_RefNannyDeclarations
09684 __Pyx_RefNannySetupContext("__ior__ (wrapper)", 0);
09685 __pyx_r = __pyx_pf_8PyClical_8clifford_52__ior__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_rhs));
09686
09687 /* function exit code */
09688 __Pyx_RefNannyFinishContext();
09689 return __pyx_r;
09690 }
09691
09692 static PyObject *__pyx_pf_8PyClical_8clifford_52__ior__(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self, PyObject *__pyx_v_rhs) {
09693 PyObject *__pyx_r = NULL;
09694 __Pyx_RefNannyDeclarations
09695 PyObject *__pyx_t_1 = NULL;
09696 int __pyx_lineno = 0;
09697 const char *__pyx_filename = NULL;
09698 int __pyx_clineno = 0;
09699 __Pyx_RefNannySetupContext("__ior__", 0);
09700
09701 /* "PyClical.pyx":959
09702 * -{1}
09703 * """
09704 * return self.wrap(self.unwrap() | toClifford(rhs)) # ««««««««
09705 *
09706 * def __pow__(self, m, dummy):

```

```

09707 */
09708 __Pyx_XDECREF(__pyx_r);
09709 __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(__pyx_v_self,
 (__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self) | __pyx_f_8PyClical_toClifford(__pyx_v_rhs))); if
 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 959, __pyx_L1_error)
09710 __Pyx_GOTREF(__pyx_t_1);
09711 __pyx_r = __pyx_t_1;
09712 __pyx_t_1 = 0;
09713 goto __pyx_L0;
09714
09715 /* "PyClical.pyx":950
09716 * return clifford().wrap(toClifford(lhs) | toClifford(rhs))
09717 *
09718 * def __ior__(self, rhs): # ««««««««
09719 * """
09720 * Transform left hand side, using right hand side as a transformation.
09721 */
09722
09723 /* function exit code */
09724 __pyx_L1_error:;
09725 __Pyx_XDECREF(__pyx_t_1);
09726 __Pyx_AddTraceback("PyClical.clifford.__ior__", __pyx_clineno, __pyx_lineno,
 __pyx_filename);
09727 __pyx_r = NULL;
09728 __pyx_L0:;
09729 __Pyx_XGIVEREF(__pyx_r);
09730 __Pyx_RefNannyFinishContext();
09731 return __pyx_r;
09732 }
09733
09734 /* "PyClical.pyx":961
09735 * return self.wrap(self.unwrap() | toClifford(rhs))
09736 *
09737 * def __pow__(self, m, dummy): # ««««««««
09738 * """
09739 * Power: self to the m.
09740 */
09741
09742 /* Python wrapper */
09743 static PyObject *__pyx_pw_8PyClical_8clifford_55__pow__(PyObject *__pyx_v_self, PyObject
 *__pyx_v_m, PyObject *__pyx_v_dummy); /*proto*/
09744 static char __pyx_doc_8PyClical_8clifford_54__pow__[] = "\n Power: self to the m.\n\n
 >> x=clifford(\"{1}\"); print(x ** 2)\n 1\n >> x=clifford(\"2\"); print(x ** 2)\n
 4\n >> x=clifford(\"2+{1}\"); print(x ** 0)\n 1\n >> x=clifford(\"2+{1}\");
 print(x ** 1)\n 2+{1}\n >> x=clifford(\"2+{1}\"); print(x ** 2)\n 5+4{1}\n
 >> i=clifford(\"{1,2}\"); print(exp(pi/2) * (i ** i))\n 1\n ";
09745 #if CYTHON_COMPILING_IN_CPYTHON
09746 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_54__pow__;
09747 #endif
09748 static PyObject *__pyx_pw_8PyClical_8clifford_55__pow__(PyObject *__pyx_v_self, PyObject
 *__pyx_v_m, PyObject *__pyx_v_dummy) {
09749 PyObject *__pyx_r = 0;
09750 __Pyx_RefNannyDeclarations
09751 __Pyx_RefNannySetupContext("__pow__ (wrapper)", 0);
09752 __pyx_r = __pyx_pf_8PyClical_8clifford_54__pow__((PyObject *)__pyx_v_self), ((PyObject
 *)__pyx_v_m), ((PyObject *)__pyx_v_dummy));
09753
09754 /* function exit code */
09755 __Pyx_RefNannyFinishContext();
09756 return __pyx_r;
09757 }
09758
09759 static PyObject *__pyx_pf_8PyClical_8clifford_54__pow__(PyObject *__pyx_v_self, PyObject
 *__pyx_v_m, CYTHON_UNUSED PyObject *__pyx_v_dummy) {
09760 PyObject *__pyx_r = NULL;
09761 __Pyx_RefNannyDeclarations
09762 PyObject *__pyx_t_1 = NULL;
09763 int __pyx_lineno = 0;
09764 const char *__pyx_filename = NULL;
09765 int __pyx_clineno = 0;
09766 __Pyx_RefNannySetupContext("__pow__", 0);
09767
09768 /* "PyClical.pyx":978
09769 * 1
09770 * """
09771 * return pow(self, m) # ««««««««
09772 *
09773 * def pow(self, m):
09774 */
09775 __Pyx_XDECREF(__pyx_r);
09776 __pyx_t_1 = __pyx_f_8PyClical_pow(__pyx_v_self, __pyx_v_m, 0); if (unlikely(!__pyx_t_1))
 __PYX_ERR(0, 978, __pyx_L1_error)
09777 __Pyx_GOTREF(__pyx_t_1);
09778 __pyx_r = __pyx_t_1;
09779 __pyx_t_1 = 0;
09780 goto __pyx_L0;
09781

```

```

09782 /* "PyClical.pyx":961
09783 * return self.wrap(self.unwrap() | toClifford(rhs))
09784 *
09785 * def __pow__(self, m, dummy): # ««««««««
09786 * """
09787 * Power: self to the m.
09788 */
09789
09790 /* function exit code */
09791 __pyx_L1_error++;
09792 __Pyx_XDECREF(__pyx_t_1);
09793 __Pyx_AddTraceback("PyClical.clifford.__pow__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
09794 __pyx_r = NULL;
09795 __pyx_L0++;
09796 __Pyx_XGIVEREF(__pyx_r);
09797 __Pyx_RefNannyFinishContext();
09798 return __pyx_r;
09799 }
09800
09801 /* "PyClical.pyx":980
09802 * return pow(self, m)
09803 *
09804 * def pow(self, m): # ««««««««
09805 * """
09806 * Power: self to the m.
09807 */
09808
09809 /* Python wrapper */
09810 static PyObject *__pyx_pw_8PyClical_8clifford_57pow(PyObject *__pyx_v_self, PyObject
*__pyx_v_m); /*proto*/
09811 static char __pyx_doc_8PyClical_8clifford_56pow[] = "\n Power: self to the m.\n\n
>> x=clifford(\"{1}\"); print(x.pow(2))\n 1\n >> x=clifford(\"2\"); print(x.pow(2))\n
4\n >> x=clifford(\"2+{1}\"); print(x.pow(0))\n 1\n >> x=clifford(\"2+{1}\");
print(x.pow(1))\n 2+{1}\n >> x=clifford(\"2+{1}\"); print(x.pow(2))\n 5+4{1}\n
>> print(clifford(\"1+{1}+{1,2}\").pow(3))\n 1+3{1}+3{1,2}\n >> i=clifford(\"{1,2}\");
print(exp(pi/2) * i.pow(i))\n 1\n ";
09812 static PyObject *__pyx_pw_8PyClical_8clifford_57pow(PyObject *__pyx_v_self, PyObject
*__pyx_v_m) {
09813 PyObject *__pyx_r = 0;
09814 __Pyx_RefNannyDeclarations
09815 __Pyx_RefNannySetupContext("pow (wrapper)", 0);
09816 __pyx_r = __pyx_pf_8PyClical_8clifford_56pow(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_m));
09817
09818 /* function exit code */
09819 __Pyx_RefNannyFinishContext();
09820 return __pyx_r;
09821 }
09822
09823 static PyObject *__pyx_pf_8PyClical_8clifford_56pow(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self, PyObject *__pyx_v_m) {
09824 PyObject *__pyx_r = NULL;
09825 __Pyx_RefNannyDeclarations
09826 PyObject *__pyx_t_1 = NULL;
09827 PyObject *__pyx_t_2 = NULL;
09828 int __pyx_t_3;
09829 int __pyx_t_4;
09830 int __pyx_t_5;
09831 int __pyx_lineno = 0;
09832 const char *__pyx_filename = NULL;
09833 int __pyx_clineno = 0;
09834 __Pyx_RefNannySetupContext("pow", 0);
09835
09836 /* "PyClical.pyx":999
09837 * 1
09838 * """
09839 * if isinstance(m, numbers.Integral): # ««««««««
09840 * return clifford().wrap(self.instance.pow(m))
09841 * else:
09842 */
09843 __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_numbers); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 999, __pyx_L1_error)
09844 __Pyx_GOTREF(__pyx_t_1);
09845 __pyx_t_2 = __Pyx_PyObject_GetAttrStr(__pyx_t_1, __pyx_n_s_Integral); if
(unlikely(!__pyx_t_2)) __PYX_ERR(0, 999, __pyx_L1_error)
09846 __Pyx_GOTREF(__pyx_t_2);
09847 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
09848 __pyx_t_3 = PyObject_IsInstance(__pyx_v_m, __pyx_t_2); if (unlikely(__pyx_t_3 == ((int)-1)))
__PYX_ERR(0, 999, __pyx_L1_error)
09849 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
09850 __pyx_t_4 = (__pyx_t_3 != 0);
09851 if (__pyx_t_4) {
09852
09853 /* "PyClical.pyx":1000
09854 * """
09855 * if isinstance(m, numbers.Integral):

```

```

09856 * return clifford().wrap(self.instance.pow(m)) # ««««««««
09857 * else:
09858 * return exp(m * log(self))
09859 */
09860 __Pyx_XDECREF(__pyx_r);
09861 __pyx_t_2 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_2)) __PYX_ERR(0, 1000, __pyx_L1_error)
09862 __Pyx_GOTREF(__pyx_t_2);
09863 __pyx_t_5 = __Pyx_PyInt_As_int(__pyx_v_m); if (unlikely((__pyx_t_5 == (int)-1) &&
PyErr_Occurred())) __PYX_ERR(0, 1000, __pyx_L1_error)
09864 __pyx_t_1 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_2), __pyx_v_self->instance->pow(__pyx_t_5)); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1000,
__pyx_L1_error)
09865 __Pyx_GOTREF(__pyx_t_1);
09866 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
09867 __pyx_r = __pyx_t_1;
09868 __pyx_t_1 = 0;
09869 goto __pyx_L0;
09870
09871 /* "PyClical.pyx":999
09872 * 1
09873 * """
09874 * if isinstance(m, numbers.Integral): # ««««««««
09875 * return clifford().wrap(self.instance.pow(m))
09876 * else:
09877 */
09878 }
09879
09880 /* "PyClical.pyx":1002
09881 * return clifford().wrap(self.instance.pow(m))
09882 * else:
09883 * return exp(m * log(self)) # ««««««««
09884 *
09885 * def outer_pow(self, m):
09886 */
09887 /*else*/ {
09888 __Pyx_XDECREF(__pyx_r);
09889 __pyx_t_1 = __pyx_f_8PyClical_log(((PyObject *)__pyx_v_self), 0, NULL); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1002, __pyx_L1_error)
09890 __Pyx_GOTREF(__pyx_t_1);
09891 __pyx_t_2 = PyNumber_Multiply(__pyx_v_m, __pyx_t_1); if (unlikely(!__pyx_t_2))
__PYX_ERR(0, 1002, __pyx_L1_error)
09892 __Pyx_GOTREF(__pyx_t_2);
09893 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
09894 __pyx_t_1 = __pyx_f_8PyClical_exp(__pyx_t_2, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0,
1002, __pyx_L1_error)
09895 __Pyx_GOTREF(__pyx_t_1);
09896 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
09897 __pyx_r = __pyx_t_1;
09898 __pyx_t_1 = 0;
09899 goto __pyx_L0;
09900 }
09901
09902 /* "PyClical.pyx":980
09903 * return pow(self, m)
09904 *
09905 * def pow(self, m): # ««««««««
09906 * """
09907 * Power: self to the m.
09908 */
09909
09910 /* function exit code */
09911 __pyx_L1_error:;
09912 __Pyx_XDECREF(__pyx_t_1);
09913 __Pyx_XDECREF(__pyx_t_2);
09914 __Pyx_AddTraceback("PyClical.clifford.pow", __pyx_clineno, __pyx_lineno, __pyx_filename);
09915 __pyx_r = NULL;
09916 __pyx_L0:;
09917 __Pyx_XGIVEREF(__pyx_r);
09918 __Pyx_RefNannyFinishContext();
09919 return __pyx_r;
09920 }
09921
09922 /* "PyClical.pyx":1004
09923 * return exp(m * log(self))
09924 *
09925 * def outer_pow(self, m): # ««««««««
09926 * """
09927 * Outer product power.
09928 */
09929
09930 /* Python wrapper */
09931 static PyObject *__pyx_pw_8PyClical_8clifford_59outer_pow(PyObject *__pyx_v_self, PyObject
*__pyx_v_m); /*proto*/
09932 static char __pyx_doc_8PyClical_8clifford_58outer_pow[] = "\n Outer product power.\n\n
>> x=clifford(\"2+{1}\")\n 1\n >> x=clifford(\"2+{1}\");
print(x.outer_pow(1))\n 2+{1}\n >> x=clifford(\"2+{1}\"); print(x.outer_pow(2))\n

```

```

4+4{1}\n >>> print(clifford("\n1+{1}+{1,2}\n").outer_pow(3))\n 1+3{1}+3{1,2}\n\n ";
09933 static PyObject *__pyx_pw_8PyClical_8clifford_59outer_pow(PyObject *__pyx_v_self, PyObject
*__pyx_v_m) {
09934 PyObject *__pyx_r = 0;
09935 __Pyx_RefNannyDeclarations
09936 __Pyx_RefNannySetupContext("outer_pow (wrapper)", 0);
09937 __pyx_r = __pyx_pf_8PyClical_8clifford_58outer_pow(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_m));
09938
09939 /* function exit code */
09940 __Pyx_RefNannyFinishContext();
09941 return __pyx_r;
09942 }
09943
09944 static PyObject *__pyx_pf_8PyClical_8clifford_58outer_pow(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self, PyObject *__pyx_v_m) {
09945 PyObject *__pyx_r = NULL;
09946 __Pyx_RefNannyDeclarations
09947 PyObject *__pyx_t_1 = NULL;
09948 int __pyx_t_2;
09949 PyObject *__pyx_t_3 = NULL;
09950 int __pyx_lineno = 0;
09951 const char *__pyx_filename = NULL;
09952 int __pyx_clineno = 0;
09953 __Pyx_RefNannySetupContext("outer_pow", 0);
09954
09955 /* "PyClical.pyx":1018
09956 *
09957 *
09958 * return clifford().wrap(self.instance.outer_pow(m))
09959 * # <<<<<<<<<
09960 *
09961 * def __call__(self, grade):
09962 */
09962 __Pyx_XDECREF(__pyx_r);
09963 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1018, __pyx_L1_error)
09964 __Pyx_GOTREF(__pyx_t_1);
09965 __pyx_t_2 = __Pyx_PyInt_As_int(__pyx_v_m); if (unlikely((__pyx_t_2 == (int)-1) &&
PyErr_Occurred())) __PYX_ERR(0, 1018, __pyx_L1_error)
09966 __pyx_t_3 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), __pyx_v_self->instance->outer_pow(__pyx_t_2)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0,
1018, __pyx_L1_error)
09967 __Pyx_GOTREF(__pyx_t_3);
09968 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
09969 __pyx_r = __pyx_t_3;
09970 __pyx_t_3 = 0;
09971 goto __pyx_L0;
09972
09973 /* "PyClical.pyx":1004
09974 *
09975 *
09976 * def outer_pow(self, m):
09977 * # <<<<<<<<<
09978 * """
09979 * Outer product power.
09980 */
09981 /* function exit code */
09982 __pyx_L1_error:;
09983 __Pyx_XDECREF(__pyx_t_1);
09984 __Pyx_XDECREF(__pyx_t_3);
09985 __Pyx_AddTraceback("PyClical.clifford.outer_pow", __pyx_clineno, __pyx_lineno,
__pyx_filename);
09986 __pyx_r = NULL;
09987 __pyx_L0:;
09988 __Pyx_XGIVEREF(__pyx_r);
09989 __Pyx_RefNannyFinishContext();
09990 return __pyx_r;
09991 }
09992
09993 /* "PyClical.pyx":1020
09994 *
09995 *
09996 * return clifford().wrap(self.instance.outer_pow(m))
09997 * # <<<<<<<<<
09998 *
09999 * def __call__(self, grade):
10000 * """
10001 * Pure grade-vector part.
10002 */
10003 /* Python wrapper */
10004 static PyObject *__pyx_pw_8PyClical_8clifford_61__call__(PyObject *__pyx_v_self, PyObject
*__pyx_args, PyObject *__pyx_kwds); /*proto*/
10005 static char __pyx_doc_8PyClical_8clifford_60__call__[] = "\n Pure grade-vector
part.\n\n >>> print(clifford("\n{1}\n") (1))\n {1}\n >>> print(clifford("\n{1}\n") (0))\n
0\n >>> print(clifford("\n1+{1}+{1,2}\n") (0))\n 1\n >>>
print(clifford("\n1+{1}+{1,2}\n") (1))\n {1}\n >>> print(clifford("\n1+{1}+{1,2}\n") (2))\n
{1,2}\n >>> print(clifford("\n1+{1}+{1,2}\n") (3))\n 0\n ";
10004 #if CYTHON_COMPILING_IN_CPYTHON
10005 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_60__call__;

```

```

10006 #endif
10007 static PyObject *__pyx_pw_8PyClical_8clifford_61__call__(PyObject *__pyx_v_self, PyObject
10008 *__pyx_args, PyObject *__pyx_kwds) {
10009 PyObject *__pyx_v_grade = 0;
10010 int __pyx_lineno = 0;
10011 const char *__pyx_filename = NULL;
10012 int __pyx_clineno = 0;
10013 PyObject *__pyx_r = 0;
10014 __Pyx_RefNannyDeclarations
10015 __Pyx_RefNannySetupContext("__call__ (wrapper)", 0);
10016 {
10017 static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_grade, 0};
10018 PyObject* values[1] = {0};
10019 if (unlikely(__pyx_kwds)) {
10020 Py_ssize_t kw_args;
10021 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
10022 switch (pos_args) {
10023 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
10024 CYTHON_FALLTHROUGH;
10025 case 0: break;
10026 default: goto __pyx_L5_argtuple_error;
10027 }
10028 kw_args = PyDict_Size(__pyx_kwds);
10029 switch (pos_args) {
10030 case 0:
10031 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_grade)) != 0))
10032 kw_args--;
10033 else goto __pyx_L5_argtuple_error;
10034 }
10035 if (unlikely(kw_args > 0)) {
10036 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values,
10037 pos_args, "__call__") < 0)) __PYX_ERR(0, 1020, __pyx_L3_error)
10038 }
10039 } else if (PyTuple_GET_SIZE(__pyx_args) != 1) {
10040 goto __pyx_L5_argtuple_error;
10041 } else {
10042 values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
10043 }
10044 __pyx_v_grade = values[0];
10045 }
10046 goto __pyx_L4_argument_unpacking_done;
10047 __pyx_L5_argtuple_error:;
10048 __Pyx_RaiseArgtupleInvalid("__call__", 1, 1, 1, PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0,
10049 1020, __pyx_L3_error)
10050 __pyx_L3_error:;
10051 __Pyx_AddTraceback("PyClical.clifford.__call__", __pyx_clineno, __pyx_lineno,
10052 __pyx_filename);
10053 __Pyx_RefNannyFinishContext();
10054 return NULL;
10055 __pyx_L4_argument_unpacking_done:;
10056 __pyx_r = __pyx_pf_8PyClical_8clifford_60__call__(((struct __pyx_obj_8PyClical_clifford
10057 *)__pyx_v_self), __pyx_v_grade);
10058 /* function exit code */
10059 __Pyx_RefNannyFinishContext();
10060 return __pyx_r;
10061 }
10062 static PyObject *__pyx_pf_8PyClical_8clifford_60__call__(struct __pyx_obj_8PyClical_clifford
10063 *__pyx_v_self, PyObject *__pyx_v_grade) {
10064 PyObject *__pyx_r = NULL;
10065 __Pyx_RefNannyDeclarations
10066 PyObject *__pyx_t_1 = NULL;
10067 int __pyx_t_2;
10068 PyObject *__pyx_t_3 = NULL;
10069 int __pyx_lineno = 0;
10070 const char *__pyx_filename = NULL;
10071 int __pyx_clineno = 0;
10072 __Pyx_RefNannySetupContext("__call__", 0);
10073 /* "PyClical.pyx":1037
10074 *
10075 * 0
10076 * """
10077 * return clifford().wrap(self.instance.call(grade))
10078 * # ««««««««
10079 *
10080 * def scalar(self):
10081 */
10082 __Pyx_XDECREF(__pyx_r);
10083 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
10084 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1037, __pyx_L1_error)
10085 __Pyx_GOTREF(__pyx_t_1);
10086 __pyx_t_2 = __Pyx_PyInt_As_int(__pyx_v_grade); if (unlikely((__pyx_t_2 == (int)-1) &&
10087 PyErr_Occurred())) __PYX_ERR(0, 1037, __pyx_L1_error)
10088 __pyx_t_3 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
10089 *)__pyx_t_1), __pyx_v_self->instance->operator()(__pyx_t_2)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0,
10090 1037, __pyx_L1_error)
10091 __Pyx_GOTREF(__pyx_t_3);

```

```

10082 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
10083 __pyx_r = __pyx_t_3;
10084 __pyx_t_3 = 0;
10085 goto __pyx_L0;
10086
10087 /* "PyClical.pyx":1020
10088 * return clifford().wrap(self.instance.outer_pow(m))
10089 *
10090 * def __call__(self, grade): # ««««««««
10091 * """
10092 * Pure grade-vector part.
10093 */
10094
10095 /* function exit code */
10096 __pyx_L1_error;;
10097 __Pyx_XDECREF(__pyx_t_1);
10098 __Pyx_XDECREF(__pyx_t_3);
10099 __Pyx_AddTraceback("PyClical.clifford.__call__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
10100 __pyx_r = NULL;
10101 __pyx_L0;;
10102 __Pyx_XGIVEREF(__pyx_r);
10103 __Pyx_RefNannyFinishContext();
10104 return __pyx_r;
10105 }
10106
10107 /* "PyClical.pyx":1039
10108 * return clifford().wrap(self.instance.call(grade))
10109 *
10110 * def scalar(self): # ««««««««
10111 * """
10112 * Scalar part.
10113 */
10114
10115 /* Python wrapper */
10116 static PyObject *__pyx_pw_8PyClical_8clifford_63scalar(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
10117 static char __pyx_doc_8PyClical_8clifford_62scalar[] = "\n Scalar part.\n\n >> clifford(\n\"1+{1}+{1,2}\").scalar()\n 1.0\n >> clifford(\n\"{1,2}\").scalar()\n 0.0\n ";
10118 static PyObject *__pyx_pw_8PyClical_8clifford_63scalar(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
10119 PyObject *__pyx_r = 0;
10120 __Pyx_RefNannyDeclarations
10121 __Pyx_RefNannySetupContext("scalar (wrapper)", 0);
10122 __pyx_r = __pyx_pf_8PyClical_8clifford_62scalar(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
10123
10124 /* function exit code */
10125 __Pyx_RefNannyFinishContext();
10126 return __pyx_r;
10127 }
10128
10129 static PyObject *__pyx_pf_8PyClical_8clifford_62scalar(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
10130 PyObject *__pyx_r = NULL;
10131 __Pyx_RefNannyDeclarations
10132 PyObject *__pyx_t_1 = NULL;
10133 int __pyx_lineno = 0;
10134 const char *__pyx_filename = NULL;
10135 int __pyx_clineno = 0;
10136 __Pyx_RefNannySetupContext("scalar", 0);
10137
10138 /* "PyClical.pyx":1048
10139 * 0.0
10140 * """
10141 * return self.instance.scalar() # ««««««««
10142 *
10143 * def pure(self):
10144 */
10145 __Pyx_XDECREF(__pyx_r);
10146 __pyx_t_1 = PyFloat_FromDouble(__pyx_v_self->instance->scalar()); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1048, __pyx_L1_error)
10147 __Pyx_GOTREF(__pyx_t_1);
10148 __pyx_r = __pyx_t_1;
10149 __pyx_t_1 = 0;
10150 goto __pyx_L0;
10151
10152 /* "PyClical.pyx":1039
10153 * return clifford().wrap(self.instance.call(grade))
10154 *
10155 * def scalar(self): # ««««««««
10156 * """
10157 * Scalar part.
10158 */
10159
10160 /* function exit code */

```



```

10161 __pyx_L1_error;;
10162 __Pyx_XDECREF(__pyx_t_1);
10163 __Pyx_AddTraceback("PyClical.clifford.scalar", __pyx_clineno, __pyx_lineno, __pyx_filename);
10164 __pyx_r = NULL;
10165 __pyx_L0;
10166 __Pyx_XGIVEREF(__pyx_r);
10167 __Pyx_RefNannyFinishContext();
10168 return __pyx_r;
10169 }
10170
10171 /* "PyClical.pyx":1050
10172 * return self.instance.scalar()
10173 *
10174 * def pure(self):
10175 * """
10176 * Pure part.
10177 */
10178
10179 /* Python wrapper */
10180 static PyObject *__pyx_pw_8PyClical_8clifford_65pure(PyObject *__pyx_v_self, CYTHON_UNUSED
10181 PyObject *unused); /*proto*/
10182 static char __pyx_doc_8PyClical_8clifford_64pure[] = "\n Pure part.\n\n >>
10183 print(clifford('{1}+{1}+{1,2}\n').pure())\n {1}+{1,2}\n >>
10184 print(clifford('{1,2}\n').pure())\n {1,2}\n ";
10185 static PyObject *__pyx_pw_8PyClical_8clifford_65pure(PyObject *__pyx_v_self, CYTHON_UNUSED
10186 PyObject *unused) {
10187 PyObject *__pyx_r = 0;
10188 __Pyx_RefNannyDeclarations
10189 __Pyx_RefNannySetupContext("pure (wrapper)", 0);
10190 __pyx_r = __pyx_pf_8PyClical_8clifford_64pure(((struct __pyx_obj_8PyClical_clifford
10191 *)__pyx_v_self));
10192
10193 /* function exit code */
10194 __Pyx_RefNannyFinishContext();
10195 return __pyx_r;
10196 }
10197
10198 static PyObject *__pyx_pf_8PyClical_8clifford_64pure(struct __pyx_obj_8PyClical_clifford
10199 *__pyx_v_self) {
10200 PyObject *__pyx_r = NULL;
10201 __Pyx_RefNannyDeclarations
10202 PyObject *__pyx_t_1 = NULL;
10203 PyObject *__pyx_t_2 = NULL;
10204 int __pyx_lineno = 0;
10205 const char *__pyx_filename = NULL;
10206 int __pyx_clineno = 0;
10207 __Pyx_RefNannySetupContext("pure", 0);
10208
10209 /* "PyClical.pyx":1059
10210 * {1,2}
10211 * """
10212 * return clifford().wrap(self.instance.pure())
10213 * # ««««««««
10214 *
10215 * def even(self):
10216 */
10217 __Pyx_XDECREF(__pyx_r);
10218 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
10219 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1059, __pyx_L1_error)
10220 __Pyx_GOTREF(__pyx_t_1);
10221 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
10222 *)__pyx_t_1), __pyx_v_self->instance->pure()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1059,
10223 __pyx_L1_error)
10224 __Pyx_GOTREF(__pyx_t_2);
10225 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
10226 __pyx_r = __pyx_t_2;
10227 __pyx_t_2 = 0;
10228 goto __pyx_L0;
10229
10230 /* "PyClical.pyx":1050
10231 * return self.instance.scalar()
10232 *
10233 * def pure(self):
10234 * """
10235 * Pure part.
10236 */
10237
10238 /* function exit code */
10239 __pyx_L1_error;;
10240 __Pyx_XDECREF(__pyx_t_1);
10241 __Pyx_XDECREF(__pyx_t_2);
10242 __Pyx_AddTraceback("PyClical.clifford.pure", __pyx_clineno, __pyx_lineno, __pyx_filename);
10243 __pyx_r = NULL;
10244 __pyx_L0;
10245 __Pyx_XGIVEREF(__pyx_r);
10246 __Pyx_RefNannyFinishContext();
10247 return __pyx_r;
10248 }

```

```

10239
10240 /* "PyClicl.pyx":1061
10241 * return clifford().wrap(self.instance.pure())
10242 *
10243 * def even(self): # ««««««««
10244 * """
10245 * Even part of multivector, sum of even grade terms.
10246 */
10247
10248 /* Python wrapper */
10249 static PyObject *__pyx_pw_8PyClicl_8clifford_67even(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
10250 static char __pyx_doc_8PyClicl_8clifford_66even[] = "\n Even part of multivector, sum
of even grade terms.\n\n >> print(clifford(\"1+{1}+{1,2}\").even())\n 1+{1,2}\n
";
10251 static PyObject *__pyx_pw_8PyClicl_8clifford_67even(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
10252 PyObject *__pyx_r = 0;
10253 __Pyx_RefNannyDeclarations
10254 __Pyx_RefNannySetupContext("even (wrapper)", 0);
10255 __pyx_r = __pyx_pf_8PyClicl_8clifford_66even(((struct __pyx_obj_8PyClicl_clifford
*)__pyx_v_self));
10256
10257 /* function exit code */
10258 __Pyx_RefNannyFinishContext();
10259 return __pyx_r;
10260 }
10261
10262 static PyObject *__pyx_pf_8PyClicl_8clifford_66even(struct __pyx_obj_8PyClicl_clifford
*__pyx_v_self) {
10263 PyObject *__pyx_r = NULL;
10264 __Pyx_RefNannyDeclarations
10265 PyObject *__pyx_t_1 = NULL;
10266 PyObject *__pyx_t_2 = NULL;
10267 int __pyx_lineno = 0;
10268 const char *__pyx_filename = NULL;
10269 int __pyx_clineno = 0;
10270 __Pyx_RefNannySetupContext("even", 0);
10271
10272 /* "PyClicl.pyx":1068
10273 * 1+{1,2}
10274 * """
10275 * return clifford().wrap(self.instance.even()) # ««««««««
10276 *
10277 * def odd(self):
10278 */
10279 __Pyx_XDECREF(__pyx_r);
10280 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClicl_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1068, __pyx_L1_error)
10281 __Pyx_GOTREF(__pyx_t_1);
10282 __pyx_t_2 = __pyx_f_8PyClicl_8clifford_wrap(((struct __pyx_obj_8PyClicl_clifford
*)__pyx_t_1), __pyx_v_self->instance->even()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1068,
__pyx_L1_error)
10283 __Pyx_GOTREF(__pyx_t_2);
10284 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
10285 __pyx_r = __pyx_t_2;
10286 __pyx_t_2 = 0;
10287 goto __pyx_L0;
10288
10289 /* "PyClicl.pyx":1061
10290 * return clifford().wrap(self.instance.pure())
10291 *
10292 * def even(self): # ««««««««
10293 * """
10294 * Even part of multivector, sum of even grade terms.
10295 */
10296
10297 /* function exit code */
10298 __pyx_L1_error;
10299 __Pyx_XDECREF(__pyx_t_1);
10300 __Pyx_XDECREF(__pyx_t_2);
10301 __Pyx_AddTraceback("PyClicl.clifford.even", __pyx_clineno, __pyx_lineno, __pyx_filename);
10302 __pyx_r = NULL;
10303 __pyx_L0;
10304 __Pyx_XGIVEREF(__pyx_r);
10305 __Pyx_RefNannyFinishContext();
10306 return __pyx_r;
10307 }
10308
10309 /* "PyClicl.pyx":1070
10310 * return clifford().wrap(self.instance.even())
10311 *
10312 * def odd(self): # ««««««««
10313 * """
10314 * Odd part of multivector, sum of odd grade terms.
10315 */
10316

```

```

10317 /* Python wrapper */
10318 static PyObject *__pyx_pw_8PyClical_8clifford_69odd(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
10319 static char __pyx_doc_8PyClical_8clifford_68odd[] = "\n Odd part of multivector, sum of
odd grade terms.\n\n >> print(clifford(\"1+{1}+{1,2}\").odd())\n {1}\n ";
10320 static PyObject *__pyx_pw_8PyClical_8clifford_69odd(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
10321 PyObject *__pyx_r = 0;
10322 __Pyx_RefNannyDeclarations
10323 __Pyx_RefNannySetupContext("odd (wrapper)", 0);
10324 __pyx_r = __pyx_pf_8PyClical_8clifford_68odd(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
10325
10326 /* function exit code */
10327 __Pyx_RefNannyFinishContext();
10328 return __pyx_r;
10329 }
10330
10331 static PyObject *__pyx_pf_8PyClical_8clifford_68odd(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
10332 PyObject *__pyx_r = NULL;
10333 __Pyx_RefNannyDeclarations
10334 PyObject *__pyx_t_1 = NULL;
10335 PyObject *__pyx_t_2 = NULL;
10336 int __pyx_lineno = 0;
10337 const char *__pyx_filename = NULL;
10338 int __pyx_clineno = 0;
10339 __Pyx_RefNannySetupContext("odd", 0);
10340
10341 /* "PyClical.pyx":1077
10342 *
10343 *
10344 * return clifford().wrap(self.instance.odd())
10345 *
10346 *
10347 */
10348 __Pyx_XDECREF(__pyx_r);
10349 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1077, __pyx_L1_error)
10350 __Pyx_GOTREF(__pyx_t_1);
10351 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), __pyx_v_self->instance->odd()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1077,
__pyx_L1_error)
10352 __Pyx_GOTREF(__pyx_t_2);
10353 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
10354 __pyx_r = __pyx_t_2;
10355 __pyx_t_2 = 0;
10356 goto __pyx_L0;
10357
10358 /* "PyClical.pyx":1070
10359 * return clifford().wrap(self.instance.even())
10360 *
10361 *
10362 *
10363 * Odd part of multivector, sum of odd grade terms.
10364 */
10365
10366 /* function exit code */
10367 __pyx_L1_error:;
10368 __Pyx_XDECREF(__pyx_t_1);
10369 __Pyx_XDECREF(__pyx_t_2);
10370 __Pyx_AddTraceback("PyClical.clifford.odd", __pyx_clineno, __pyx_lineno, __pyx_filename);
10371 __pyx_r = NULL;
10372 __pyx_L0:;
10373 __Pyx_XGIVEREF(__pyx_r);
10374 __Pyx_RefNannyFinishContext();
10375 return __pyx_r;
10376 }
10377
10378 /* "PyClical.pyx":1079
10379 * return clifford().wrap(self.instance.odd())
10380 *
10381 *
10382 *
10383 * Vector part of multivector, as a Python list, with respect to frm.
10384 */
10385
10386 /* Python wrapper */
10387 static PyObject *__pyx_pw_8PyClical_8clifford_71vector_part(PyObject *__pyx_v_self, PyObject
*__pyx_args, PyObject *__pyx_kwds); /*proto*/
10388 static char __pyx_doc_8PyClical_8clifford_70vector_part[] = "\n Vector part of
multivector, as a Python list, with respect to frm.\n\n >>
print(clifford(\"1+2{1}+3{2}+4{1,2}\").vector_part())\n [2.0, 3.0]\n >>
print(clifford(\"1+2{1}+3{2}+4{1,2}\").vector_part(index_set({-1,1,2})))\n [0.0, 2.0, 3.0]\n
 ";
10389 static PyObject *__pyx_pw_8PyClical_8clifford_71vector_part(PyObject *__pyx_v_self, PyObject
*__pyx_args, PyObject *__pyx_kwds) {

```

```

10390 PyObject * __pyx_v_frm = 0;
10391 int __pyx_lineno = 0;
10392 const char * __pyx_filename = NULL;
10393 int __pyx_clineno = 0;
10394 PyObject * __pyx_r = 0;
10395 __Pyx_RefNannyDeclarations
10396 __Pyx_RefNannySetupContext("vector_part (wrapper)", 0);
10397 {
10398 static PyObject * __pyx_pyargnames[] = {&__pyx_n_s_frm, 0};
10399 PyObject* values[1] = {0};
10400 values[0] = ((PyObject *)Py_None);
10401 if (unlikely(__pyx_kwds)) {
10402 Py_ssize_t kw_args;
10403 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
10404 switch (pos_args) {
10405 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
10406 CYTHON_FALLTHROUGH;
10407 case 0: break;
10408 default: goto __pyx_L5_argtuple_error;
10409 }
10410 kw_args = PyDict_Size(__pyx_kwds);
10411 switch (pos_args) {
10412 case 0:
10413 if (kw_args > 0) {
10414 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_frm);
10415 if (value) { values[0] = value; kw_args--; }
10416 }
10417 }
10418 if (unlikely(kw_args > 0)) {
10419 if (unlikely(!__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0, values,
10420 pos_args, "vector_part") < 0)) __PYX_ERR(0, 1079, __pyx_L3_error)
10421 }
10422 else {
10423 switch (PyTuple_GET_SIZE(__pyx_args)) {
10424 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
10425 CYTHON_FALLTHROUGH;
10426 case 0: break;
10427 default: goto __pyx_L5_argtuple_error;
10428 }
10429 }
10430 __pyx_v_frm = values[0];
10431 }
10432 goto __pyx_L4_argument_unpacking_done;
10433 __pyx_L5_argtuple_error:;
10434 __Pyx_RaiseArgtupleInvalid("vector_part", 0, 0, 1, PyTuple_GET_SIZE(__pyx_args));
10435 __PYX_ERR(0, 1079, __pyx_L3_error)
10436 __pyx_L3_error:;
10437 __Pyx_AddTraceback("PyClical.clifford.vector_part", __pyx_clineno, __pyx_lineno,
10438 __pyx_filename);
10439 __Pyx_RefNannyFinishContext();
10440 return NULL;
10441 __pyx_L4_argument_unpacking_done:;
10442 __pyx_r = __pyx_pf_8PyClical_8clifford_70vector_part(((struct __pyx_obj_8PyClical_clifford
10443 *)__pyx_v_self), __pyx_v_frm);
10444
10445 /* function exit code */
10446 __Pyx_RefNannyFinishContext();
10447 return __pyx_r;
10448 }
10449
10450 static PyObject * __pyx_pf_8PyClical_8clifford_70vector_part(struct
10451 __pyx_obj_8PyClical_clifford * __pyx_v_self, PyObject * __pyx_v_frm) {
10452 PyObject * __pyx_v_error_msg_prefix = NULL;
10453 std::vector<scalar_t> __pyx_v_vec;
10454 int __pyx_v_n;
10455 int __pyx_v_i;
10456 PyObject * __pyx_v_lst = NULL;
10457 PyObject * __pyx_v_err = NULL;
10458 PyObject * __pyx_r = NULL;
10459 __Pyx_RefNannyDeclarations
10460 PyObject * __pyx_t_1 = NULL;
10461 PyObject * __pyx_t_2 = NULL;
10462 PyObject * __pyx_t_3 = NULL;
10463 int __pyx_t_4;
10464 int __pyx_t_5;
10465 std::vector<scalar_t> __pyx_t_6;
10466 PyObject * __pyx_t_7 = NULL;
10467 int __pyx_t_8;
10468 int __pyx_t_9;
10469 int __pyx_t_10;
10470 PyObject * __pyx_t_11 = NULL;
10471 PyObject * __pyx_t_12 = NULL;
10472 PyObject * __pyx_t_13 = NULL;
10473 PyObject * __pyx_t_14 = NULL;
10474 PyObject * __pyx_t_15 = NULL;
10475 char const * __pyx_t_16;
10476 PyObject * __pyx_t_17 = NULL;

```

```

10472 PyObject *__pyx_t_18 = NULL;
10473 PyObject *__pyx_t_19 = NULL;
10474 PyObject *__pyx_t_20 = NULL;
10475 PyObject *__pyx_t_21 = NULL;
10476 PyObject *__pyx_t_22 = NULL;
10477 int __pyx_lineno = 0;
10478 const char *__pyx_filename = NULL;
10479 int __pyx_clineno = 0;
10480 __Pyx_RefNannySetupContext("vector_part", 0);
10481
10482 /* "PyClical.pyx":1088
10483 * [0.0, 2.0, 3.0]
10484 * """
10485 * error_msg_prefix = "Cannot take vector part of " # ««««««««
10486 * cdef vector[scalar_t] vec
10487 * cdef int n
10488 */
10489 __Pyx_INCREF(__pyx_kp_u_Cannot_take_vector_part_of);
10490 __pyx_v_error_msg_prefix = __pyx_kp_u_Cannot_take_vector_part_of;
10491
10492 /* "PyClical.pyx":1092
10493 * cdef int n
10494 * cdef int i
10495 * try: # ««««««««
10496 * if frm is None:
10497 * vec = self.instance.vector_part()
10498 */
10499 {
10500 __Pyx_PyThreadState_declare
10501 __Pyx_PyThreadState_assign
10502 __Pyx_ExceptionSave(&__pyx_t_1, &__pyx_t_2, &__pyx_t_3);
10503 __Pyx_XGOTREF(__pyx_t_1);
10504 __Pyx_XGOTREF(__pyx_t_2);
10505 __Pyx_XGOTREF(__pyx_t_3);
10506 /*try:*/ {
10507
10508 /* "PyClical.pyx":1093
10509 * cdef int i
10510 * try:
10511 * if frm is None: # ««««««««
10512 * vec = self.instance.vector_part()
10513 * else:
10514 */
10515 __pyx_t_4 = (__pyx_v_frm == Py_None);
10516 __pyx_t_5 = (__pyx_t_4 != 0);
10517 if (__pyx_t_5) {
10518
10519 /* "PyClical.pyx":1094
10520 * try:
10521 * if frm is None:
10522 * vec = self.instance.vector_part() # ««««««««
10523 * else:
10524 * vec = self.instance.vector_part((<index_set>frm).unwrap())
10525 */
10526 __pyx_t_6 = __pyx_v_self->instance->vector_part();
10527 __pyx_v_vec = __pyx_t_6;
10528
10529 /* "PyClical.pyx":1093
10530 * cdef int i
10531 * try:
10532 * if frm is None: # ««««««««
10533 * vec = self.instance.vector_part()
10534 * else:
10535 */
10536 goto __pyx_L9;
10537 }
10538
10539 /* "PyClical.pyx":1096
10540 * vec = self.instance.vector_part()
10541 * else:
10542 * vec = self.instance.vector_part((<index_set>frm).unwrap()) # ««««««««
10543 * n = vec.size()
10544 * lst = [0.0]*n
10545 */
10546 /*else*/ {
10547 try {
10548 __pyx_t_6 =
10549 __pyx_v_self->instance->vector_part(__pyx_f_8PyClical_9index_set_unwrap(((struct
10550 __pyx_obj_8PyClical_index_set *)__pyx_v_frm)));
10551 } catch (...) {
10552 __Pyx_CppExn2PyErr();
10553 __PYX_ERR(0, 1096, __pyx_L3_error)
10554 }
10555 __pyx_v_vec = __pyx_t_6;
10556 __pyx_L9:;

```

```

10557 /* "PyClical.pyx":1097
10558 * else:
10559 * vec = self.instance.vector_part((<index_set>frm).unwrap())
10560 * n = vec.size() # ««««««««
10561 * lst = [0.0]*n
10562 * for i in xrange(n):
10563 */
10564 __pyx_v_n = __pyx_v_vec.size();
10565
10566 /* "PyClical.pyx":1098
10567 * vec = self.instance.vector_part((<index_set>frm).unwrap())
10568 * n = vec.size()
10569 * lst = [0.0]*n # ««««««««
10570 * for i in xrange(n):
10571 * lst[i] = vec[i]
10572 */
10573 __pyx_t_7 = PyList_New(1 * ((__pyx_v_n<0) ? 0:__pyx_v_n)); if (unlikely(!__pyx_t_7))
__PYX_ERR(0, 1098, __pyx_L3_error)
10574 __Pyx_GOTREF(__pyx_t_7);
10575 { Py_ssize_t __pyx_temp;
10576 for (__pyx_temp=0; __pyx_temp < __pyx_v_n; __pyx_temp++) {
10577 __Pyx_INCREF(__pyx_float_0_0);
10578 __Pyx_GIVEREF(__pyx_float_0_0);
10579 PyList_SET_ITEM(__pyx_t_7, __pyx_temp, __pyx_float_0_0);
10580 }
10581 }
10582 __pyx_v_lst = ((PyObject *)__pyx_t_7);
10583 __pyx_t_7 = 0;
10584
10585 /* "PyClical.pyx":1099
10586 * n = vec.size()
10587 * lst = [0.0]*n
10588 * for i in xrange(n): # ««««««««
10589 * lst[i] = vec[i]
10590 * return lst
10591 */
10592 __pyx_t_8 = __pyx_v_n;
10593 __pyx_t_9 = __pyx_t_8;
10594 for (__pyx_t_10 = 0; __pyx_t_10 < __pyx_t_9; __pyx_t_10+=1) {
10595 __pyx_v_i = __pyx_t_10;
10596
10597 /* "PyClical.pyx":1100
10598 * lst = [0.0]*n
10599 * for i in xrange(n):
10600 * lst[i] = vec[i] # ««««««««
10601 * return lst
10602 * except RuntimeError as err:
10603 */
10604 __pyx_t_7 = PyFloat_FromDouble((__pyx_v_vec[__pyx_v_i])); if (unlikely(!__pyx_t_7))
__PYX_ERR(0, 1100, __pyx_L3_error)
10605 __Pyx_GOTREF(__pyx_t_7);
10606 if (unlikely(__Pyx_SetItemInt(__pyx_v_lst, __pyx_v_i, __pyx_t_7, int, 1,
__Pyx_PyInt_From_int, 1, 1, 1) < 0)) __PYX_ERR(0, 1100, __pyx_L3_error)
10607 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
10608 }
10609
10610 /* "PyClical.pyx":1101
10611 * for i in xrange(n):
10612 * lst[i] = vec[i]
10613 * return lst # ««««««««
10614 * except RuntimeError as err:
10615 * raise ValueError(error_msg_prefix + str(self) + " using invalid "
10616 */
10617 __Pyx_XDECREF(__pyx_r);
10618 __Pyx_INCREF(__pyx_v_lst);
10619 __pyx_r = __pyx_v_lst;
10620 goto __pyx_L7_try_return;
10621
10622 /* "PyClical.pyx":1092
10623 * cdef int n
10624 * cdef int i
10625 * try: # ««««««««
10626 * if frm is None:
10627 * vec = self.instance.vector_part()
10628 */
10629 }
10630 __pyx_L3_error:;
10631 __Pyx_XDECREF(__pyx_t_7); __pyx_t_7 = 0;
10632
10633 /* "PyClical.pyx":1102
10634 * lst[i] = vec[i]
10635 * return lst
10636 * except RuntimeError as err: # ««««««««
10637 * raise ValueError(error_msg_prefix + str(self) + " using invalid "
10638 * + repr(frm) + " as frame:\n\t"
10639 */
10640 __pyx_t_8 = __Pyx_PyErr_ExceptionMatches(__pyx_builtin_RuntimeError);

```

```

10641 if (__pyx_t_8) {
10642 __Pyx_AddTraceback("PyClical.clifford.vector_part", __pyx_clineno, __pyx_lineno,
__pyx_filename);
10643 if (__Pyx_GetException(&__pyx_t_7, &__pyx_t_11, &__pyx_t_12) < 0) __PYX_ERR(0, 1102,
__pyx_L5_except_error)
10644 __Pyx_GOTREF(__pyx_t_7);
10645 __Pyx_GOTREF(__pyx_t_11);
10646 __Pyx_GOTREF(__pyx_t_12);
10647 __Pyx_INCREF(__pyx_t_11);
10648 __pyx_v_err = __pyx_t_11;
10649 /*try:*/ {
10650
10651 /* "PyClical.pyx":1103
10652 * return lst
10653 * except RuntimeError as err:
10654 * raise ValueError(error_msg_prefix + str(self) + " using invalid "
10655 * + repr(frm) + " as frame:\n\t"
10656 * + str(err))
10657 */
10658 __pyx_t_13 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), ((PyObject
*)__pyx_v_self)); if (unlikely(!__pyx_t_13)) __PYX_ERR(0, 1103, __pyx_L17_error)
10659 __Pyx_GOTREF(__pyx_t_13);
10660 __pyx_t_14 = __Pyx_PyUnicode_Concat(__pyx_v_error_msg_prefix, __pyx_t_13); if
(unlikely(!__pyx_t_14)) __PYX_ERR(0, 1103, __pyx_L17_error)
10661 __Pyx_GOTREF(__pyx_t_14);
10662 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
10663 __pyx_t_13 = __Pyx_PyUnicode_Concat(__pyx_t_14, __pyx_kp_u_using_invalid); if
(unlikely(!__pyx_t_13)) __PYX_ERR(0, 1103, __pyx_L17_error)
10664 __Pyx_GOTREF(__pyx_t_13);
10665 __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
10666
10667 /* "PyClical.pyx":1104
10668 * except RuntimeError as err:
10669 * raise ValueError(error_msg_prefix + str(self) + " using invalid "
10670 * + repr(frm) + " as frame:\n\t"
10671 * + str(err))
10672 */
10673 __pyx_t_14 = PyObject_Repr(__pyx_v_frm); if (unlikely(!__pyx_t_14)) __PYX_ERR(0, 1104,
__pyx_L17_error)
10674 __Pyx_GOTREF(__pyx_t_14);
10675 __pyx_t_15 = PyNumber_Add(__pyx_t_13, __pyx_t_14); if (unlikely(!__pyx_t_15))
__PYX_ERR(0, 1104, __pyx_L17_error)
10676 __Pyx_GOTREF(__pyx_t_15);
10677 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
10678 __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
10679 __pyx_t_14 = PyNumber_Add(__pyx_t_15, __pyx_kp_u_as_frame); if (unlikely(!__pyx_t_14))
__PYX_ERR(0, 1104, __pyx_L17_error)
10680 __Pyx_GOTREF(__pyx_t_14);
10681 __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
10682
10683 /* "PyClical.pyx":1105
10684 * raise ValueError(error_msg_prefix + str(self) + " using invalid "
10685 * + repr(frm) + " as frame:\n\t"
10686 * + str(err))
10687 * # ««««««««
10688 *
10689 * def involute(self):
10690 */
10691 __pyx_t_15 = __Pyx_PyObject_CallOneArg(((PyObject *)(&PyUnicode_Type)), __pyx_v_err);
if (unlikely(!__pyx_t_15)) __PYX_ERR(0, 1105, __pyx_L17_error)
10692 __Pyx_GOTREF(__pyx_t_15);
10693 __pyx_t_13 = PyNumber_Add(__pyx_t_14, __pyx_t_15); if (unlikely(!__pyx_t_13))
__PYX_ERR(0, 1105, __pyx_L17_error)
10694 __Pyx_GOTREF(__pyx_t_13);
10695 __Pyx_DECREF(__pyx_t_14); __pyx_t_14 = 0;
10696 __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
10697
10698 /* "PyClical.pyx":1103
10699 * return lst
10700 * except RuntimeError as err:
10701 * raise ValueError(error_msg_prefix + str(self) + " using invalid "
10702 * + repr(frm) + " as frame:\n\t"
10703 * + str(err))
10704 */
10705 __pyx_t_15 = __Pyx_PyObject_CallOneArg(__pyx_builtin_ValueError, __pyx_t_13); if
(unlikely(!__pyx_t_15)) __PYX_ERR(0, 1103, __pyx_L17_error)
10706 __Pyx_GOTREF(__pyx_t_15);
10707 __Pyx_DECREF(__pyx_t_13); __pyx_t_13 = 0;
10708 __Pyx_Raise(__pyx_t_15, 0, 0, 0);
10709 __Pyx_DECREF(__pyx_t_15); __pyx_t_15 = 0;
10710 __PYX_ERR(0, 1103, __pyx_L17_error)
10711 }
10712
10713 /* "PyClical.pyx":1102
10714 * lst[i] = vec[i]
10715 * return lst
10716 * except RuntimeError as err:
10717 * # ««««««««

```

```

10717 * raise ValueError(error_msg_prefix + str(self) + " using invalid "
10718 * + repr(frm) + " as frame:\n\t"
10719 */
10720 /*finally:*/ {
10721 __pyx_L17_error;;
10722 /*exception exit:*/{
10723 __Pyx_PyThreadState_declare
10724 __Pyx_PyThreadState_assign
10725 __pyx_t_17 = 0; __pyx_t_18 = 0; __pyx_t_19 = 0; __pyx_t_20 = 0; __pyx_t_21 = 0;
__pyx_t_22 = 0;
10726 __Pyx_XDECREFF(__pyx_t_13); __pyx_t_13 = 0;
10727 __Pyx_XDECREFF(__pyx_t_14); __pyx_t_14 = 0;
10728 __Pyx_XDECREFF(__pyx_t_15); __pyx_t_15 = 0;
10729 if (PY_MAJOR_VERSION >= 3) __Pyx_ExceptionSwap(&__pyx_t_20, &__pyx_t_21,
&__pyx_t_22);
10730 if ((PY_MAJOR_VERSION < 3) || unlikely(__Pyx_GetException(&__pyx_t_17, &__pyx_t_18,
&__pyx_t_19) < 0)) __Pyx_ErrFetch(&__pyx_t_17, &__pyx_t_18, &__pyx_t_19);
10731 __Pyx_XGOTREF(__pyx_t_17);
10732 __Pyx_XGOTREF(__pyx_t_18);
10733 __Pyx_XGOTREF(__pyx_t_19);
10734 __Pyx_XGOTREF(__pyx_t_20);
10735 __Pyx_XGOTREF(__pyx_t_21);
10736 __Pyx_XGOTREF(__pyx_t_22);
10737 __pyx_t_8 = __pyx_lineno; __pyx_t_9 = __pyx_clineno; __pyx_t_16 = __pyx_filename;
10738 {
10739 __Pyx_DECREF(__pyx_v_err);
10740 __pyx_v_err = NULL;
10741 }
10742 if (PY_MAJOR_VERSION >= 3) {
10743 __Pyx_XGIVEREF(__pyx_t_20);
10744 __Pyx_XGIVEREF(__pyx_t_21);
10745 __Pyx_XGIVEREF(__pyx_t_22);
10746 __Pyx_ExceptionReset(__pyx_t_20, __pyx_t_21, __pyx_t_22);
10747 }
10748 __Pyx_XGIVEREF(__pyx_t_17);
10749 __Pyx_XGIVEREF(__pyx_t_18);
10750 __Pyx_XGIVEREF(__pyx_t_19);
10751 __Pyx_ErrRestore(__pyx_t_17, __pyx_t_18, __pyx_t_19);
10752 __pyx_t_17 = 0; __pyx_t_18 = 0; __pyx_t_19 = 0; __pyx_t_20 = 0; __pyx_t_21 = 0;
__pyx_t_22 = 0;
10753 __pyx_lineno = __pyx_t_8; __pyx_clineno = __pyx_t_9; __pyx_filename = __pyx_t_16;
10754 goto __pyx_L5_except_error;
10755 }
10756 }
10757 }
10758 goto __pyx_L5_except_error;
10759 __pyx_L5_except_error;;
10760
10761 /* "PyClical.pyx":1092
10762 * cdef int n
10763 * cdef int i
10764 * try:
10765 * if frm is None:
10766 * vec = self.instance.vector_part()
10767 */
10768 __Pyx_XGIVEREF(__pyx_t_1);
10769 __Pyx_XGIVEREF(__pyx_t_2);
10770 __Pyx_XGIVEREF(__pyx_t_3);
10771 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
10772 goto __pyx_L1_error;
10773 __pyx_L7_try_return;;
10774 __Pyx_XGIVEREF(__pyx_t_1);
10775 __Pyx_XGIVEREF(__pyx_t_2);
10776 __Pyx_XGIVEREF(__pyx_t_3);
10777 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
10778 goto __pyx_L0;
10779 }
10780
10781 /* "PyClical.pyx":1079
10782 * return clifford().wrap(self.instance.odd())
10783 *
10784 * def vector_part(self, frm = None):
10785 * """
10786 * Vector part of multivector, as a Python list, with respect to frm.
10787 */
10788
10789 /* function exit code */
10790 __pyx_L1_error;;
10791 __Pyx_XDECREFF(__pyx_t_7);
10792 __Pyx_XDECREFF(__pyx_t_11);
10793 __Pyx_XDECREFF(__pyx_t_12);
10794 __Pyx_XDECREFF(__pyx_t_13);
10795 __Pyx_XDECREFF(__pyx_t_14);
10796 __Pyx_XDECREFF(__pyx_t_15);
10797 __Pyx_AddTraceback("PyClical.clifford.vector_part", __pyx_clineno, __pyx_lineno,
__pyx_filename);
10798 __pyx_r = NULL;

```



```

10799 __pyx_L0;;
10800 __Pyx_XDECREF(__pyx_v_error_msg_prefix);
10801 __Pyx_XDECREF(__pyx_v_lst);
10802 __Pyx_XDECREF(__pyx_v_err);
10803 __Pyx_XGIVEREF(__pyx_r);
10804 __Pyx_RefNannyFinishContext();
10805 return __pyx_r;
10806 }
10807
10808 /* "PyClical.pyx":1107
10809 * + str(err))
10810 *
10811 * def involute(self): # ««««««««
10812 * """
10813 * Main involution, each {i} is replaced by -{i} in each term,
10814 */
10815
10816 /* Python wrapper */
10817 static PyObject *__pyx_pw_8PyClical_8clifford_73involute(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
10818 static char __pyx_doc_8PyClical_8clifford_72involute[] = "\n Main involution, each {i}
is replaced by -{i} in each term,\n eg. clifford(\"{1}\") -> -clifford(\"{1}\").\n\n >>
print(clifford(\"{1}\").involute())\n -{1}\n >> print((clifford(\"{2}\") *
clifford(\"{1}\")).involute())\n -{1,2}\n >> print((clifford(\"{1}\") *
clifford(\"{2}\")).involute())\n {1,2}\n >>
print(clifford(\"1+{1}+{1,2}\").involute())\n 1-{1}+{1,2}\n ";
10819 static PyObject *__pyx_pw_8PyClical_8clifford_73involute(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
10820 PyObject *__pyx_r = 0;
10821 __Pyx_RefNannyDeclarations
10822 __Pyx_RefNannySetupContext("involute (wrapper)", 0);
10823 __pyx_r = __pyx_pf_8PyClical_8clifford_72involute(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
10824
10825 /* function exit code */
10826 __Pyx_RefNannyFinishContext();
10827 return __pyx_r;
10828 }
10829
10830 static PyObject *__pyx_pf_8PyClical_8clifford_72involute(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
10831 PyObject *__pyx_r = NULL;
10832 __Pyx_RefNannyDeclarations
10833 PyObject *__pyx_t_1 = NULL;
10834 PyObject *__pyx_t_2 = NULL;
10835 int __pyx_lineno = 0;
10836 const char *__pyx_filename = NULL;
10837 int __pyx_clineno = 0;
10838 __Pyx_RefNannySetupContext("involute", 0);
10839
10840 /* "PyClical.pyx":1121
10841 * 1-{1}+{1,2}
10842 * """
10843 * return clifford().wrap(self.instance.involute()) # ««««««««
10844 *
10845 * def reverse(self):
10846 */
10847 __Pyx_XDECREF(__pyx_r);
10848 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1121, __pyx_L1_error)
10849 __Pyx_GOTREF(__pyx_t_1);
10850 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), __pyx_v_self->instance->involute()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1121,
__pyx_L1_error)
10851 __Pyx_GOTREF(__pyx_t_2);
10852 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
10853 __pyx_r = __pyx_t_2;
10854 __pyx_t_2 = 0;
10855 goto __pyx_L0;
10856
10857 /* "PyClical.pyx":1107
10858 * + str(err))
10859 *
10860 * def involute(self): # ««««««««
10861 * """
10862 * Main involution, each {i} is replaced by -{i} in each term,
10863 */
10864
10865 /* function exit code */
10866 __pyx_L1_error:;
10867 __Pyx_XDECREF(__pyx_t_1);
10868 __Pyx_XDECREF(__pyx_t_2);
10869 __Pyx_AddTraceback("PyClical.clifford.involute", __pyx_clineno, __pyx_lineno,
__pyx_filename);
10870 __pyx_r = NULL;
10871 __pyx_L0;;
10872 __Pyx_XGIVEREF(__pyx_r);

```

```

10873 __Pyx_RefNannyFinishContext();
10874 return __pyx_r;
10875 }
10876
10877 /* "PyClical.pyx":1123
10878 * return clifford().wrap(self.instance.involute())
10879 *
10880 * def reverse(self):
10881 * """
10882 * Reversion, eg. clifford("{1}")*clifford("{2}") -> clifford("{2}")*clifford("{1}").
10883 */
10884
10885 /* Python wrapper */
10886 static PyObject *__pyx_pw_8PyClical_8clifford_75reverse(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
10887 static char __pyx_doc_8PyClical_8clifford_74reverse[] = "\n Reversion, eg.
clifford(\"{1}\")*clifford(\"{2}\") -> clifford(\"{2}\")*clifford(\"{1}\").\n\n >>
print(clifford(\"{1}\").reverse())\n {1}\n >> print((clifford(\"{2}\") *
clifford(\"{1}\")).reverse())\n {1,2}\n >> print((clifford(\"{1}\") *
clifford(\"{2}\")).reverse())\n -{1,2}\n >> print(clifford(\"1+{1}+{1,2}\").reverse())\n
1+{1}-{1,2}\n ";
10888 static PyObject *__pyx_pw_8PyClical_8clifford_75reverse(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
10889 PyObject *__pyx_r = 0;
10890 __Pyx_RefNannyDeclarations
10891 __Pyx_RefNannySetupContext("reverse (wrapper)", 0);
10892 __pyx_r = __pyx_pf_8PyClical_8clifford_74reverse(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
10893
10894 /* function exit code */
10895 __Pyx_RefNannyFinishContext();
10896 return __pyx_r;
10897 }
10898
10899 static PyObject *__pyx_pf_8PyClical_8clifford_74reverse(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
10900 PyObject *__pyx_r = NULL;
10901 __Pyx_RefNannyDeclarations
10902 PyObject *__pyx_t_1 = NULL;
10903 PyObject *__pyx_t_2 = NULL;
10904 int __pyx_lineno = 0;
10905 const char *__pyx_filename = NULL;
10906 int __pyx_clineno = 0;
10907 __Pyx_RefNannySetupContext("reverse", 0);
10908
10909 /* "PyClical.pyx":1136
10910 * 1+{1}-{1,2}
10911 * """
10912 * return clifford().wrap(self.instance.reverse())
10913 *
10914 * def conj(self):
10915 */
10916 __Pyx_XDECREF(__pyx_r);
10917 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1136, __pyx_L1_error)
10918 __Pyx_GOTREF(__pyx_t_1);
10919 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), __pyx_v_self->instance->reverse()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1136,
__pyx_L1_error)
10920 __Pyx_GOTREF(__pyx_t_2);
10921 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
10922 __pyx_r = __pyx_t_2;
10923 __pyx_t_2 = 0;
10924 goto __pyx_L0;
10925
10926 /* "PyClical.pyx":1123
10927 * return clifford().wrap(self.instance.involute())
10928 *
10929 * def reverse(self):
10930 * """
10931 * Reversion, eg. clifford("{1}")*clifford("{2}") -> clifford("{2}")*clifford("{1}").
10932 */
10933
10934 /* function exit code */
10935 __pyx_L1_error:;
10936 __Pyx_XDECREF(__pyx_t_1);
10937 __Pyx_XDECREF(__pyx_t_2);
10938 __Pyx_AddTraceback("PyClical.clifford.reverse", __pyx_clineno, __pyx_lineno,
__pyx_filename);
10939 __pyx_r = NULL;
10940 __pyx_L0:;
10941 __Pyx_XGIVEREF(__pyx_r);
10942 __Pyx_RefNannyFinishContext();
10943 return __pyx_r;
10944 }
10945
10946 /* "PyClical.pyx":1138

```

```

10947 * return clifford().wrap(self.instance.reverse())
10948 *
10949 * def conj(self): # ««««««««
10950 * """
10951 * Conjugation, reverse o involute == involute o reverse.
10952 */
10953
10954 /* Python wrapper */
10955 static PyObject *__pyx_pw_8PyClical_8clifford_77conj(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
10956 static char __pyx_doc_8PyClical_8clifford_76conj[] = "\n Conjugation, reverse o
involute == involute o reverse.\n\n >> print((clifford(\"{1}\")).conj())\n -{1}\n
>> print((clifford(\"{2}\") * clifford(\"{1}\")).conj())\n {1,2}\n >>
print((clifford(\"{1}\") * clifford(\"{2}\")).conj())\n -{1,2}\n >>
print(clifford(\"1+{1}+{1,2}\").conj())\n 1-{1}-{1,2}\n ";
10957 static PyObject *__pyx_pw_8PyClical_8clifford_77conj(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
10958 PyObject *__pyx_r = 0;
10959 __Pyx_RefNannyDeclarations
10960 __Pyx_RefNannySetupContext("conj (wrapper)", 0);
10961 __pyx_r = __pyx_pf_8PyClical_8clifford_76conj(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
10962
10963 /* function exit code */
10964 __Pyx_RefNannyFinishContext();
10965 return __pyx_r;
10966 }
10967
10968 static PyObject *__pyx_pf_8PyClical_8clifford_76conj(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
10969 PyObject *__pyx_r = NULL;
10970 __Pyx_RefNannyDeclarations
10971 PyObject *__pyx_t_1 = NULL;
10972 PyObject *__pyx_t_2 = NULL;
10973 int __pyx_lineno = 0;
10974 const char *__pyx_filename = NULL;
10975 int __pyx_clineno = 0;
10976 __Pyx_RefNannySetupContext("conj", 0);
10977
10978 /* "PyClical.pyx":1151
10979 * 1-{1}-{1,2}
10980 * """
10981 * return clifford().wrap(self.instance.conj()) # ««««««««
10982 *
10983 * def quad(self):
10984 */
10985 __Pyx_XDECREF(__pyx_r);
10986 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1151, __pyx_L1_error)
10987 __Pyx_GOTREF(__pyx_t_1);
10988 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), __pyx_v_self->instance->conj()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1151,
__pyx_L1_error)
10989 __Pyx_GOTREF(__pyx_t_2);
10990 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
10991 __pyx_r = __pyx_t_2;
10992 __pyx_t_2 = 0;
10993 goto __pyx_L0;
10994
10995 /* "PyClical.pyx":1138
10996 * return clifford().wrap(self.instance.reverse())
10997 *
10998 * def conj(self): # ««««««««
10999 * """
11000 * Conjugation, reverse o involute == involute o reverse.
11001 */
11002
11003 /* function exit code */
11004 __pyx_L1_error:;
11005 __Pyx_XDECREF(__pyx_t_1);
11006 __Pyx_XDECREF(__pyx_t_2);
11007 __Pyx_AddTraceback("PyClical.clifford.conj", __pyx_clineno, __pyx_lineno, __pyx_filename);
11008 __pyx_r = NULL;
11009 __pyx_L0:;
11010 __Pyx_XGIVEREF(__pyx_r);
11011 __Pyx_RefNannyFinishContext();
11012 return __pyx_r;
11013 }
11014
11015 /* "PyClical.pyx":1153
11016 * return clifford().wrap(self.instance.conj())
11017 *
11018 * def quad(self): # ««««««««
11019 * """
11020 * Quadratic form == (rev(x)*x)(0).
11021 */
11022

```

```

11023 /* Python wrapper */
11024 static PyObject *__pyx_pw_8PyClical_8clifford_79quad(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
11025 static char __pyx_doc_8PyClical_8clifford_78quad[] = "\n Quadratic form ==
(rev(x)*x)(0).\n\n >> print(clifford(\"1+{1}+{1,2}\").quad())\n 3.0\n >>
print(clifford(\"1+{-1}+{1,2}+{1,2,3}\").quad())\n 2.0\n ";
11026 static PyObject *__pyx_pw_8PyClical_8clifford_79quad(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
11027 PyObject *__pyx_r = 0;
11028 __Pyx_RefNannyDeclarations
11029 __Pyx_RefNannySetupContext("quad (wrapper)", 0);
11030 __pyx_r = __pyx_pf_8PyClical_8clifford_78quad(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
11031
11032 /* function exit code */
11033 __Pyx_RefNannyFinishContext();
11034 return __pyx_r;
11035 }
11036
11037 static PyObject *__pyx_pf_8PyClical_8clifford_78quad(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
11038 PyObject *__pyx_r = NULL;
11039 __Pyx_RefNannyDeclarations
11040 PyObject *__pyx_t_1 = NULL;
11041 int __pyx_lineno = 0;
11042 const char *__pyx_filename = NULL;
11043 int __pyx_clineno = 0;
11044 __Pyx_RefNannySetupContext("quad", 0);
11045
11046 /* "PyClical.pyx":1162
11047 * 2.0
11048 * """
11049 * return self.instance.quad()
11050 * # <<<<<<<<<
11051 *
11052 */
11053 __Pyx_XDECREF(__pyx_r);
11054 __pyx_t_1 = PyFloat_FromDouble(__pyx_v_self->instance->quad()); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1162, __pyx_L1_error)
11055 __Pyx_GOTREF(__pyx_t_1);
11056 __pyx_r = __pyx_t_1;
11057 __pyx_t_1 = 0;
11058 goto __pyx_L0;
11059
11060 /* "PyClical.pyx":1153
11061 * return clifford().wrap(self.instance.conj())
11062 *
11063 *
11064 * def quad(self):
11065 * # <<<<<<<<<
11066 * Quadratic form == (rev(x)*x)(0).
11067 *
11068 */
11069 /* function exit code */
11070 __pyx_L1_error;
11071 __Pyx_XDECREF(__pyx_t_1);
11072 __Pyx_AddTraceback("PyClical.clifford.quad", __pyx_clineno, __pyx_lineno, __pyx_filename);
11073 __pyx_r = NULL;
11074 __pyx_L0;
11075 __Pyx_XGIVEREF(__pyx_r);
11076 __Pyx_RefNannyFinishContext();
11077 return __pyx_r;
11078 }
11079
11080 /* "PyClical.pyx":1164
11081 * return self.instance.quad()
11082 *
11083 *
11084 * def norm(self):
11085 * # <<<<<<<<<
11086 * """
11087 * Norm == sum of squares of coordinates.
11088 *
11089 */
11090 static PyObject *__pyx_pw_8PyClical_8clifford_81norm(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
11091 static char __pyx_doc_8PyClical_8clifford_80norm[] = "\n Norm == sum of squares of
coordinates.\n\n >> clifford(\"1+{1}+{1,2}\").norm()\n 3.0\n >>
clifford(\"1+{-1}+{1,2}+{1,2,3}\").norm()\n 4.0\n ";
11092 static PyObject *__pyx_pw_8PyClical_8clifford_81norm(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
11093 PyObject *__pyx_r = 0;
11094 __Pyx_RefNannyDeclarations
11095 __Pyx_RefNannySetupContext("norm (wrapper)", 0);
11096 __pyx_r = __pyx_pf_8PyClical_8clifford_80norm(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
11097
11098 /* function exit code */
11099 __Pyx_RefNannyFinishContext();

```

Generated by Doxygen

```

11178 *
11179 * def max_abs(self):
11180 */
11181 __Pyx_XDECREF(__pyx_r);
11182 __pyx_t_1 = PyFloat_FromDouble(abs(__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1182, __pyx_L1_error)
11183 __Pyx_GOTREF(__pyx_t_1);
11184 __pyx_r = __pyx_t_1;
11185 __pyx_t_1 = 0;
11186 goto __pyx_L0;
11187
11188 /* "PyClical.pyx":1175
11189 * return self.instance.norm()
11190 *
11191 * def abs(self):
11192 * """
11193 * Absolute value: square root of norm.
11194 */
11195
11196 /* function exit code */
11197 __pyx_L1_error:;
11198 __Pyx_XDECREF(__pyx_t_1);
11199 __Pyx_AddTraceback("PyClical.clifford.abs", __pyx_clineno, __pyx_lineno, __pyx_filename);
11200 __pyx_r = NULL;
11201 __pyx_L0:;
11202 __Pyx_XGIVEREF(__pyx_r);
11203 __Pyx_RefNannyFinishContext();
11204 return __pyx_r;
11205 }
11206
11207 /* "PyClical.pyx":1184
11208 * return glucat.abs(self.unwrap())
11209 *
11210 * def max_abs(self):
11211 * """
11212 * Maximum of absolute values of components of multivector: multivector infinity norm.
11213 */
11214
11215 /* Python wrapper */
11216 static PyObject *__pyx_pw_8PyClical_8clifford_85max_abs(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
11217 static char __pyx_doc_8PyClical_8clifford_84max_abs[] = "\n Maximum of absolute values
of components of multivector: multivector infinity norm.\n\n >>
clifford(\\"1+{-1}+{1,2}+{1,2,3}\\").max_abs()\n 1.0\n >>
clifford(\\"3+2{1}+{1,2}\\").max_abs()\n 3.0\n ";
11218 static PyObject *__pyx_pw_8PyClical_8clifford_85max_abs(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
11219 PyObject *__pyx_r = 0;
11220 __Pyx_RefNannyDeclarations
11221 __Pyx_RefNannySetupContext("max_abs (wrapper)", 0);
11222 __pyx_r = __pyx_pf_8PyClical_8clifford_84max_abs(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
11223
11224 /* function exit code */
11225 __Pyx_RefNannyFinishContext();
11226 return __pyx_r;
11227 }
11228
11229 static PyObject *__pyx_pf_8PyClical_8clifford_84max_abs(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
11230 PyObject *__pyx_r = NULL;
11231 __Pyx_RefNannyDeclarations
11232 PyObject *__pyx_t_1 = NULL;
11233 int __pyx_lineno = 0;
11234 const char *__pyx_filename = NULL;
11235 int __pyx_clineno = 0;
11236 __Pyx_RefNannySetupContext("max_abs", 0);
11237
11238 /* "PyClical.pyx":1193
11239 * 3.0
11240 * """
11241 * return self.instance.max_abs()
11242 *
11243 * def truncated(self, limit):
11244 */
11245 __Pyx_XDECREF(__pyx_r);
11246 __pyx_t_1 = PyFloat_FromDouble(__pyx_v_self->instance->max_abs()); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1193, __pyx_L1_error)
11247 __Pyx_GOTREF(__pyx_t_1);
11248 __pyx_r = __pyx_t_1;
11249 __pyx_t_1 = 0;
11250 goto __pyx_L0;
11251
11252 /* "PyClical.pyx":1184
11253 * return glucat.abs(self.unwrap())
11254 *
11255 * def max_abs(self):
11256 */

```

```

11256 * """
11257 * Maximum of absolute values of components of multivector: multivector infinity norm.
11258 */
11259
11260 /* function exit code */
11261 __pyx_L1_error++;
11262 __Pyx_XDECREF(__pyx_t_1);
11263 __Pyx_AddTraceback("PyClicl.clifford.max_abs", __pyx_clineno, __pyx_lineno,
__pyx_filename);
11264 __pyx_r = NULL;
11265 __pyx_L0;
11266 __Pyx_XGIVEREF(__pyx_r);
11267 __Pyx_RefNannyFinishContext();
11268 return __pyx_r;
11269 }
11270
11271 /* "PyClicl.pyx":1195
11272 * return self.instance.max_abs()
11273 *
11274 * def truncated(self, limit):
11275 * """
11276 * Remove all terms of self with relative size smaller than limit.
11277 */
11278
11279 /* Python wrapper */
11280 static PyObject *__pyx_pw_8PyClicl_8clifford_87truncated(PyObject *__pyx_v_self, PyObject
*__pyx_v_limit); /*proto*/
11281 static char __pyx_doc_8PyClicl_8clifford_86truncated[] = "\n Remove all terms of self
with relative size smaller than limit.\n\n >>
clifford(\"1e8+{1}+1e-8{1,2}\").truncated(1.0e-6)\n clifford(\"100000000\")\n >>
clifford(\"1e4+{1}+1e-4{1,2}\").truncated(1.0e-6)\n clifford(\"10000+{1}\")\n ";
11282 static PyObject *__pyx_pw_8PyClicl_8clifford_87truncated(PyObject *__pyx_v_self, PyObject
*__pyx_v_limit) {
11283 PyObject *__pyx_r = 0;
11284 __Pyx_RefNannyDeclarations
11285 __Pyx_RefNannySetupContext("truncated (wrapper)", 0);
11286 __pyx_r = __pyx_pf_8PyClicl_8clifford_86truncated(((struct __pyx_obj_8PyClicl_clifford
*)__pyx_v_self), ((PyObject *)__pyx_v_limit));
11287
11288 /* function exit code */
11289 __Pyx_RefNannyFinishContext();
11290 return __pyx_r;
11291 }
11292
11293 static PyObject *__pyx_pf_8PyClicl_8clifford_86truncated(struct __pyx_obj_8PyClicl_clifford
*__pyx_v_self, PyObject *__pyx_v_limit) {
11294 PyObject *__pyx_r = NULL;
11295 __Pyx_RefNannyDeclarations
11296 PyObject *__pyx_t_1 = NULL;
11297 scalar_t __pyx_t_2;
11298 PyObject *__pyx_t_3 = NULL;
11299 int __pyx_lineno = 0;
11300 const char *__pyx_filename = NULL;
11301 int __pyx_clineno = 0;
11302 __Pyx_RefNannySetupContext("truncated", 0);
11303
11304 /* "PyClicl.pyx":1204
11305 * clifford("10000+{1}")
11306 * """
11307 * return clifford().wrap(self.instance.truncated(limit))
11308 *
11309 * def isinf(self):
11310 */
11311 __Pyx_XDECREF(__pyx_r);
11312 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClicl_clifford)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1204, __pyx_L1_error)
11313 __Pyx_GOTREF(__pyx_t_1);
11314 __pyx_t_2 = __pyx_PyFloat_AsDouble(__pyx_v_limit); if (unlikely((__pyx_t_2 ==
((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 1204, __pyx_L1_error)
11315 __pyx_t_3 = __pyx_f_8PyClicl_8clifford_wrap(((struct __pyx_obj_8PyClicl_clifford
*)__pyx_t_1), __pyx_v_self->instance->truncated(__pyx_t_2)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0,
1204, __pyx_L1_error)
11316 __Pyx_GOTREF(__pyx_t_3);
11317 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
11318 __pyx_r = __pyx_t_3;
11319 __pyx_t_3 = 0;
11320 goto __pyx_L0;
11321
11322 /* "PyClicl.pyx":1195
11323 * return self.instance.max_abs()
11324 *
11325 * def truncated(self, limit):
11326 * """
11327 * Remove all terms of self with relative size smaller than limit.
11328 */
11329
11330 /* function exit code */

```

```

11331 __pyx_L1_error;;
11332 __Pyx_XDECREF(__pyx_t_1);
11333 __Pyx_XDECREF(__pyx_t_3);
11334 __Pyx_AddTraceback("PyClical.clifford.truncated", __pyx_clineno, __pyx_lineno,
__pyx_filename);
11335 __pyx_r = NULL;
11336 __pyx_L0;;
11337 __Pyx_XGIVEREF(__pyx_r);
11338 __Pyx_RefNannyFinishContext();
11339 return __pyx_r;
11340 }
11341
11342 /* "PyClical.pyx":1206
11343 * return clifford().wrap(self.instance.truncated(limit))
11344 *
11345 * def isinf(self):
11346 * """
11347 * Check if a multivector contains any infinite values.
11348 */
11349
11350 /* Python wrapper */
11351 static PyObject *__pyx_pw_8PyClical_8clifford_89isinf(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
11352 static char __pyx_doc_8PyClical_8clifford_88isinf[] = "\n Check if a multivector
contains any infinite values.\n\n >> clifford().isinf()\n False\n ";
11353 static PyObject *__pyx_pw_8PyClical_8clifford_89isinf(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
11354 PyObject *__pyx_r = 0;
11355 __Pyx_RefNannyDeclarations
11356 __Pyx_RefNannySetupContext("isinf (wrapper)", 0);
11357 __pyx_r = __pyx_pf_8PyClical_8clifford_88isinf(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
11358
11359 /* function exit code */
11360 __Pyx_RefNannyFinishContext();
11361 return __pyx_r;
11362 }
11363
11364 static PyObject *__pyx_pf_8PyClical_8clifford_88isinf(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
11365 PyObject *__pyx_r = NULL;
11366 __Pyx_RefNannyDeclarations
11367 PyObject *__pyx_t_1 = NULL;
11368 int __pyx_lineno = 0;
11369 const char *__pyx_filename = NULL;
11370 int __pyx_clineno = 0;
11371 __Pyx_RefNannySetupContext("isinf", 0);
11372
11373 /* "PyClical.pyx":1213
11374 * False
11375 * """
11376 * return self.instance.isnan()
11377 * # ««««««««
11378 * def isnan(self):
11379 */
11380 __Pyx_XDECREF(__pyx_r);
11381 __pyx_t_1 = __Pyx_PyBool_FromLong(__pyx_v_self->instance->isnan()); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1213, __pyx_L1_error)
11382 __Pyx_GOTREF(__pyx_t_1);
11383 __pyx_r = __pyx_t_1;
11384 __pyx_t_1 = 0;
11385 goto __pyx_L0;
11386
11387 /* "PyClical.pyx":1206
11388 * return clifford().wrap(self.instance.truncated(limit))
11389 *
11390 * def isinf(self):
11391 * """
11392 * Check if a multivector contains any infinite values.
11393 */
11394
11395 /* function exit code */
11396 __pyx_L1_error;;
11397 __Pyx_XDECREF(__pyx_t_1);
11398 __Pyx_AddTraceback("PyClical.clifford.isinf", __pyx_clineno, __pyx_lineno, __pyx_filename);
11399 __pyx_r = NULL;
11400 __pyx_L0;;
11401 __Pyx_XGIVEREF(__pyx_r);
11402 __Pyx_RefNannyFinishContext();
11403 return __pyx_r;
11404 }
11405
11406 /* "PyClical.pyx":1215
11407 * return self.instance.isnan()
11408 *
11409 * def isnan(self):
11410 * """

```



```

11411 * Check if a multivector contains any IEEE NaN values.
11412 */
11413
11414 /* Python wrapper */
11415 static PyObject *__pyx_pw_8PyClical_8clifford_91isnan(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
11416 static char __pyx_doc_8PyClical_8clifford_90isnan[] = "\n Check if a multivector
contains any IEEE NaN values.\n\n >> clifford().isnan()\n False\n ";
11417 static PyObject *__pyx_pf_8PyClical_8clifford_91isnan(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
11418 PyObject *__pyx_r = 0;
11419 __Pyx_RefNannyDeclarations
11420 __Pyx_RefNannySetupContext("isnan (wrapper)", 0);
11421 __pyx_r = __pyx_pf_8PyClical_8clifford_90isnan(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
11422
11423 /* function exit code */
11424 __Pyx_RefNannyFinishContext();
11425 return __pyx_r;
11426 }
11427
11428 static PyObject *__pyx_pf_8PyClical_8clifford_90isnan(struct __pyx_obj_8PyClical_clifford
*__pyx_v_self) {
11429 PyObject *__pyx_r = NULL;
11430 __Pyx_RefNannyDeclarations
11431 PyObject *__pyx_t_1 = NULL;
11432 int __pyx_lineno = 0;
11433 const char *__pyx_filename = NULL;
11434 int __pyx_clineno = 0;
11435 __Pyx_RefNannySetupContext("isnan", 0);
11436
11437 /* "PyClical.pyx":1222
11438 * False
11439 * """
11440 * return self.instance.isnan() # ««««««««
11441 *
11442 * def frame(self):
11443 */
11444 __Pyx_XDECREF(__pyx_r);
11445 __pyx_t_1 = __Pyx_PyBool_FromLong(__pyx_v_self->instance->isnan()); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1222, __pyx_L1_error)
11446 __Pyx_GOTREF(__pyx_t_1);
11447 __pyx_r = __pyx_t_1;
11448 __pyx_t_1 = 0;
11449 goto __pyx_L0;
11450
11451 /* "PyClical.pyx":1215
11452 * return self.instance.isnan()
11453 *
11454 * def isnan(self): # ««««««««
11455 * """
11456 * Check if a multivector contains any IEEE NaN values.
11457 */
11458
11459 /* function exit code */
11460 __pyx_L1_error:;
11461 __Pyx_XDECREF(__pyx_t_1);
11462 __Pyx_AddTraceback("PyClical.clifford.isnan", __pyx_clineno, __pyx_lineno, __pyx_filename);
11463 __pyx_r = NULL;
11464 __pyx_L0:;
11465 __Pyx_XGIVEREF(__pyx_r);
11466 __Pyx_RefNannyFinishContext();
11467 return __pyx_r;
11468 }
11469
11470 /* "PyClical.pyx":1224
11471 * return self.instance.isnan()
11472 *
11473 * def frame(self): # ««««««««
11474 * """
11475 * Subalgebra generated by all generators of terms of given multivector.
11476 */
11477
11478 /* Python wrapper */
11479 static PyObject *__pyx_pw_8PyClical_8clifford_93frame(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
11480 static char __pyx_doc_8PyClical_8clifford_92frame[] = "\n Subalgebra generated by all
generators of terms of given multivector.\n\n >>
print(clifford(\"1+3{-1}+2{1,2}+4{-2,7}\").frame())\n {-2,-1,1,2,7}\n >>
s=clifford(\"1+3{-1}+2{1,2}+4{-2,7}\").frame(); type(s)\n <class 'PyClical.index_set'>\n
 ";
11481 static PyObject *__pyx_pf_8PyClical_8clifford_93frame(PyObject *__pyx_v_self, CYTHON_UNUSED
PyObject *unused) {
11482 PyObject *__pyx_r = 0;
11483 __Pyx_RefNannyDeclarations
11484 __Pyx_RefNannySetupContext("frame (wrapper)", 0);
11485 __pyx_r = __pyx_pf_8PyClical_8clifford_92frame(((struct __pyx_obj_8PyClical_clifford

```

```

 *)__pyx_v_self));
11486
11487 /* function exit code */
11488 __Pyx_RefNannyFinishContext();
11489 return __pyx_r;
11490 }
11491
11492 static PyObject *__pyx_pf_8PyClical_8clifford_92frame(struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self) {
11493 PyObject *__pyx_r = NULL;
11494 __Pyx_RefNannyDeclarations
11495 PyObject *__pyx_t_1 = NULL;
11496 PyObject *__pyx_t_2 = NULL;
11497 int __pyx_lineno = 0;
11498 const char *__pyx_filename = NULL;
11499 int __pyx_clineno = 0;
11500 __Pyx_RefNannySetupContext("frame", 0);
11501
11502 /* "PyClical.pyx":1233
11503 * <class 'PyClical.index_set'>
11504 * """
11505 * return index_set().wrap(self.instance.frame()) # ««««««««
11506 *
11507 * def __repr__(self):
11508 */
11509 __Pyx_XDECREF(__pyx_r);
11510 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_index_set)); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1233, __pyx_L1_error)
11511 __Pyx_GOTREF(__pyx_t_1);
11512 __pyx_t_2 = __pyx_f_8PyClical_9index_set_wrap(((struct __pyx_obj_8PyClical_index_set
*)__pyx_t_1), __pyx_v_self->instance->frame()); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1233,
__pyx_L1_error)
11513 __Pyx_GOTREF(__pyx_t_2);
11514 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
11515 __pyx_r = __pyx_t_2;
11516 __pyx_t_2 = 0;
11517 goto __pyx_L0;
11518
11519 /* "PyClical.pyx":1224
11520 * return self.instance.isnan()
11521 *
11522 * def frame(self): # ««««««««
11523 * """
11524 * Subalgebra generated by all generators of terms of given multivector.
11525 */
11526
11527 /* function exit code */
11528 __pyx_L1_error:;
11529 __Pyx_XDECREF(__pyx_t_1);
11530 __Pyx_XDECREF(__pyx_t_2);
11531 __Pyx_AddTraceback("PyClical.clifford.frame", __pyx_clineno, __pyx_lineno, __pyx_filename);
11532 __pyx_r = NULL;
11533 __pyx_L0:;
11534 __Pyx_XGIVEREF(__pyx_r);
11535 __Pyx_RefNannyFinishContext();
11536 return __pyx_r;
11537 }
11538
11539 /* "PyClical.pyx":1235
11540 * return index_set().wrap(self.instance.frame())
11541 *
11542 * def __repr__(self): # ««««««««
11543 * """
11544 * The official string representation of self.
11545 */
11546
11547 /* Python wrapper */
11548 static PyObject *__pyx_pw_8PyClical_8clifford_95__repr__(PyObject *__pyx_v_self); /*proto*/
11549 static char __pyx_doc_8PyClical_8clifford_94__repr__[] = "\n The
\342\200\234official\342\200\235 string representation of self.\n\n >>
clifford(\\"1+3{-1}+2{1,2}+4{-2,7}\\").__repr__()\n 'clifford(\\"1+3{-1}+2{1,2}+4{-2,7}\\")'\n
";
11550 #if CYTHON_COMPILING_IN_CPYTHON
11551 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_94__repr__;
11552 #endif
11553 static PyObject *__pyx_pw_8PyClical_8clifford_95__repr__(PyObject *__pyx_v_self) {
11554 PyObject *__pyx_r = 0;
11555 __Pyx_RefNannyDeclarations
11556 __Pyx_RefNannySetupContext("__repr__ (wrapper)", 0);
11557 __pyx_r = __pyx_pf_8PyClical_8clifford_94__repr__(((struct __pyx_obj_8PyClical_clifford
*)__pyx_v_self));
11558
11559 /* function exit code */
11560 __Pyx_RefNannyFinishContext();
11561 return __pyx_r;
11562 }
11563

```

```

11564 static PyObject *__pyx_pf_8PyClical_8clifford_94__repr__(struct __pyx_obj_8PyClical_clifford
11565 *__pyx_v_self) {
11566 PyObject *__pyx_r = NULL;
11567 __Pyx_RefNannyDeclarations
11568 PyObject *__pyx_t_1 = NULL;
11569 int __pyx_lineno = 0;
11570 const char *__pyx_filename = NULL;
11571 int __pyx_clineno = 0;
11572 __Pyx_RefNannySetupContext("__repr__", 0);
11573
11574 /* "PyClical.pyx":1242
11575 *
11576 * return clifford_to_repr(self.unwrap()).decode() # ««««««««
11577 *
11578 * def __str__(self):
11579 */
11580 __Pyx_XDECREF(__pyx_r);
11581 __pyx_t_1 =
11582 __Pyx_decode_cpp_string(clifford_to_repr(__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self)), 0,
11583 PY_SSIZE_T_MAX, NULL, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1242, __pyx_L1_error)
11584 __Pyx_GOTREF(__pyx_t_1);
11585 __pyx_r = __pyx_t_1;
11586 __pyx_t_1 = 0;
11587 goto __pyx_L0;
11588
11589 /* "PyClical.pyx":1235
11590 * return index_set().wrap(self.instance.frame())
11591 *
11592 * def __repr__(self): # ««««««««
11593 * """
11594 * The official string representation of self.
11595 */
11596 /* function exit code */
11597 __pyx_L1_error++;
11598 __Pyx_XDECREF(__pyx_t_1);
11599 __Pyx_AddTraceback("PyClical.clifford.__repr__", __pyx_clineno, __pyx_lineno,
11600 __pyx_filename);
11601 __pyx_r = NULL;
11602 __pyx_L0:;
11603 __Pyx_XGIVEREF(__pyx_r);
11604 __Pyx_RefNannyFinishContext();
11605 return __pyx_r;
11606 }
11607
11608 /* "PyClical.pyx":1244
11609 * return clifford_to_repr(self.unwrap()).decode()
11610 *
11611 * def __str__(self): # ««««««««
11612 * """
11613 * The informal string representation of self.
11614 */
11615
11616 /* Python wrapper */
11617 static PyObject *__pyx_pw_8PyClical_8clifford_97__str__(PyObject *__pyx_v_self); /*proto*/
11618 static char __pyx_doc_8PyClical_8clifford_96__str__[] = "\n The
11619 \342\200\234informal\342\200\235 string representation of self.\n\n >>
11620 clifford(\342\200\234\342\200\235\342\200\235 string representation of self.\n\n '1+3{-1}+2{1,2}+4{-2,7}'\n ";
11621 #if CYTHON_COMPILING_IN_CPYTHON
11622 struct wrapperbase __pyx_wrapperbase_8PyClical_8clifford_96__str__;
11623 #endif
11624 static PyObject *__pyx_pw_8PyClical_8clifford_97__str__(PyObject *__pyx_v_self) {
11625 PyObject *__pyx_r = 0;
11626 __Pyx_RefNannyDeclarations
11627 __Pyx_RefNannySetupContext("__str__ (wrapper)", 0);
11628 __pyx_r = __pyx_pf_8PyClical_8clifford_96__str__(((struct __pyx_obj_8PyClical_clifford
11629 *)__pyx_v_self));
11630
11631 /* function exit code */
11632 __Pyx_RefNannyFinishContext();
11633 return __pyx_r;
11634 }
11635
11636 static PyObject *__pyx_pf_8PyClical_8clifford_96__str__(struct __pyx_obj_8PyClical_clifford
11637 *__pyx_v_self) {
11638 PyObject *__pyx_r = NULL;
11639 __Pyx_RefNannyDeclarations
11640 PyObject *__pyx_t_1 = NULL;
11641 int __pyx_lineno = 0;
11642 const char *__pyx_filename = NULL;
11643 int __pyx_clineno = 0;
11644 __Pyx_RefNannySetupContext("__str__", 0);
11645
11646 /* "PyClical.pyx":1251
11647 * '1+3{-1}+2{1,2}+4{-2,7}'
11648 *
11649 * """

```

```

11643 * return clifford_to_str(self.unwrap()).decode() # ««««««««
11644 *
11645 * def clifford_hidden_doctests():
11646 */
11647 __Pyx_XDECREF(__pyx_r);
11648 __pyx_t_1 =
__Pyx_decode_cpp_string(clifford_to_str(__pyx_f_8PyClical_8clifford_unwrap(__pyx_v_self)), 0,
PY_SSIZE_T_MAX, NULL, NULL, NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1251, __pyx_L1_error)
11649 __Pyx_GOTREF(__pyx_t_1);
11650 __pyx_r = __pyx_t_1;
11651 __pyx_t_1 = 0;
11652 goto __pyx_L0;
11653
11654 /* "PyClical.pyx":1244
11655 * return clifford_to_repr(self.unwrap()).decode()
11656 *
11657 * def __str__(self): # ««««««««
11658 * """
11659 * The informal string representation of self.
11660 */
11661
11662 /* function exit code */
11663 __pyx_L1_error:;
11664 __Pyx_XDECREF(__pyx_t_1);
11665 __Pyx_AddTraceback("PyClical.clifford.__str__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
11666 __pyx_r = NULL;
11667 __pyx_L0:;
11668 __Pyx_XGIVEREF(__pyx_r);
11669 __Pyx_RefNannyFinishContext();
11670 return __pyx_r;
11671 }
11672
11673 /* "(tree fragment)":1
11674 * def __reduce_cython__(self): # ««««««««
11675 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
11676 * def __setstate_cython__(self, __pyx_state):
11677 */
11678
11679 /* Python wrapper */
11680 static PyObject *__pyx_pw_8PyClical_8clifford_99__reduce_cython__(PyObject *__pyx_v_self,
CYTHON_UNUSED PyObject *unused); /*proto*/
11681 static PyObject *__pyx_pw_8PyClical_8clifford_99__reduce_cython__(PyObject *__pyx_v_self,
CYTHON_UNUSED PyObject *unused) {
11682 PyObject *__pyx_r = 0;
11683 __Pyx_RefNannyDeclarations
11684 __Pyx_RefNannySetupContext("__reduce_cython__ (wrapper)", 0);
11685 __pyx_r = __pyx_pf_8PyClical_8clifford_98__reduce_cython__(((struct
__pyx_obj_8PyClical_clifford *)__pyx_v_self));
11686
11687 /* function exit code */
11688 __Pyx_RefNannyFinishContext();
11689 return __pyx_r;
11690 }
11691
11692 static PyObject *__pyx_pf_8PyClical_8clifford_98__reduce_cython__(CYTHON_UNUSED struct
__pyx_obj_8PyClical_clifford *)__pyx_v_self) {
11693 PyObject *__pyx_r = NULL;
11694 __Pyx_RefNannyDeclarations
11695 PyObject *__pyx_t_1 = NULL;
11696 int __pyx_lineno = 0;
11697 const char *__pyx_filename = NULL;
11698 int __pyx_clineno = 0;
11699 __Pyx_RefNannySetupContext("__reduce_cython__", 0);
11700
11701 /* "(tree fragment)":2
11702 * def __reduce_cython__(self):
11703 * raise TypeError("no default __reduce__ due to non-trivial __cinit__") # ««««««««
11704 * def __setstate_cython__(self, __pyx_state):
11705 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
11706 */
11707 __pyx_t_1 = __Pyx_PyObject_Call(__pyx_builtin_TypeError, __pyx_tuple__11, NULL); if
(unlikely(!__pyx_t_1)) __PYX_ERR(1, 2, __pyx_L1_error)
11708 __Pyx_GOTREF(__pyx_t_1);
11709 __Pyx_Raise(__pyx_t_1, 0, 0, 0);
11710 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
11711 __PYX_ERR(1, 2, __pyx_L1_error)
11712
11713 /* "(tree fragment)":1
11714 * def __reduce_cython__(self): # ««««««««
11715 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
11716 * def __setstate_cython__(self, __pyx_state):
11717 */
11718
11719 /* function exit code */
11720 __pyx_L1_error:;
11721 __Pyx_XDECREF(__pyx_t_1);

```

```

11722 __Pyx_AddTraceback("PyClical.clifford.__reduce_cython__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
11723 __pyx_r = NULL;
11724 __Pyx_XGIVEREF(__pyx_r);
11725 __Pyx_RefNannyFinishContext();
11726 return __pyx_r;
11727 }
11728
11729 /* "(tree fragment)":3
11730 * def __reduce_cython__(self):
11731 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
11732 * def __setstate_cython__(self, __pyx_state):
11733 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
11734 */
11735
11736 /* Python wrapper */
11737 static PyObject *__pyx_pw_8PyClical_8clifford_101__setstate_cython__(PyObject *__pyx_v_self,
PyObject *__pyx_v__pyx_state); /*proto*/
11738 static PyObject *__pyx_pw_8PyClical_8clifford_101__setstate_cython__(PyObject *__pyx_v_self,
PyObject *__pyx_v__pyx_state) {
11739 PyObject *__pyx_r = 0;
11740 __Pyx_RefNannyDeclarations
11741 __Pyx_RefNannySetupContext("__setstate_cython__ (wrapper)", 0);
11742 __pyx_r = __pyx_pf_8PyClical_8clifford_100__setstate_cython__(((struct
__pyx_obj_8PyClical_clifford *)__pyx_v_self), ((PyObject *)__pyx_v__pyx_state));
11743
11744 /* function exit code */
11745 __Pyx_RefNannyFinishContext();
11746 return __pyx_r;
11747 }
11748
11749 static PyObject *__pyx_pf_8PyClical_8clifford_100__setstate_cython__(CYTHON_UNUSED struct
__pyx_obj_8PyClical_clifford *__pyx_v_self, CYTHON_UNUSED PyObject *__pyx_v__pyx_state) {
11750 PyObject *__pyx_r = NULL;
11751 __Pyx_RefNannyDeclarations
11752 PyObject *__pyx_t_1 = NULL;
11753 int __pyx_lineno = 0;
11754 const char *__pyx_filename = NULL;
11755 int __pyx_clineno = 0;
11756 __Pyx_RefNannySetupContext("__setstate_cython__", 0);
11757
11758 /* "(tree fragment)":4
11759 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
11760 * def __setstate_cython__(self, __pyx_state):
11761 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
11762 */
11763 __pyx_t_1 = __Pyx_PyObject_Call(__pyx_builtin_TypeError, __pyx_tuple__12, NULL); if
(unlikely(!__pyx_t_1)) __PYX_ERR(1, 4, __pyx_L1_error)
11764 __Pyx_GOTREF(__pyx_t_1);
11765 __Pyx_Raise(__pyx_t_1, 0, 0, 0);
11766 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
11767 __PYX_ERR(1, 4, __pyx_L1_error)
11768
11769 /* "(tree fragment)":3
11770 * def __reduce_cython__(self):
11771 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
11772 * def __setstate_cython__(self, __pyx_state):
11773 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
11774 */
11775
11776 /* function exit code */
11777 __pyx_L1_error:;
11778 __Pyx_XDECREF(__pyx_t_1);
11779 __Pyx_AddTraceback("PyClical.clifford.__setstate_cython__", __pyx_clineno, __pyx_lineno,
__pyx_filename);
11780 __pyx_r = NULL;
11781 __Pyx_XGIVEREF(__pyx_r);
11782 __Pyx_RefNannyFinishContext();
11783 return __pyx_r;
11784 }
11785
11786 /* "PyClical.pyx":1253
11787 * return clifford_to_str(self.unwrap()).decode()
11788 *
11789 * def clifford_hidden_doctests():
11790 * """
11791 * Tests for functions that Doctest cannot see.
11792 */
11793
11794 /* Python wrapper */
11795 static PyObject *__pyx_pw_8PyClical_9clifford_hidden_doctests(PyObject *__pyx_self,
CYTHON_UNUSED PyObject *unused); /*proto*/
11796 static char __pyx_doc_8PyClical_8clifford_hidden_doctests[] = "\n Tests for functions that
Doctest cannot see.\n\n For clifford.__cinit__: Construct an object of type clifford.\n\n >>
print(clifford(2))\n 2\n >> print(clifford(2.0))\n 2\n >> print(clifford(1.0e-1))\n
0.1\n >> print(clifford(\"2\")\n 2\n >> print(clifford(\"{1,2,3}\")\n 2{1,2,3}\n
print(clifford(clifford(\"2{1,2,3}\")\n 2{1,2,3}\n >> print(clifford(\"-{1}\")\n -{1}\n

```

```

>> print(clifford(2,index_set({1,2})))\n 2{1,2}\n >> print(clifford([2,3],index_set({1,2})))\n
2{1}+3{2}\n >> print(clifford([1,2]))\n Traceback (most recent call last):\n ...
TypeError: Cannot initialize clifford object from <class 'list'>.\n >> print(clifford(None))\n
Traceback (most recent call last):\n ...
TypeError: Cannot initialize clifford object from
<class 'NoneType'>.\n >> print(clifford(None,[1,2]))\n Traceback (most recent call last):\n
...
TypeError: Cannot initialize clifford object from (<class 'NoneType'>, <class 'list'>).\n
>> print(clifford([1,2],[1,2]))\n Traceback (most recent call last):\n ...
TypeError:
Cannot initialize clifford object from (<class 'list'>, <class 'list'>).\n >>
print(clifford("\n"))\n Traceback (most recent call last):\n ...
ValueError: Cannot
initialize clifford object from invalid string ".\n >> print(clifford("\{")\n Traceback (most
recent call last):\n ...
ValueError: Cannot initialize clifford object from invalid string
'\{'.\n >> print(clifford("\{1")\n Traceback (most recent call last):\n ...
ValueError: Cannot initialize clifford object from invalid string '\{1'.\n >>
print(clifford("\{+")\n Traceback (most recent call last):\n ...
ValueError: Cannot
initialize clifford object from invalid string '+'.\n >> print(clifford("\{-")\n Traceback
(most recent call last):\n ...
ValueError: Cannot initialize clifford object fro"m invalid
string '-'.\n >> print(clifford("\{1}+"))\n Traceback (most recent call last):\n ...
ValueError: Cannot initialize clifford object from invalid string '{1}+'.\n For
clifford.__richcmp__: Compare objects of type clifford.\n >> clifford("\{1}") ==
clifford("\{1{1}")\n True\n >> clifford("\{1}") != clifford("\{1.0{1}")\n False\n >>
clifford("\{1}") != clifford("\{1.0}")\n True\n >> clifford("\{1,2}") == None\n False\n
>> clifford("\{1,2}") != None\n True\n >> None == clifford("\{1,2}")\n False\n >> None
!= clifford("\{1,2}")\n True\n ";
11797 static PyMethodDef __pyx_mdef_8PyClical_9clifford_hidden_doctests =
{"clifford_hidden_doctests", (PyCFunction)__pyx_pw_8PyClical_9clifford_hidden_doctests, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_hidden_doctests};
11798 static PyObject *__pyx_pw_8PyClical_9clifford_hidden_doctests(PyObject *__pyx_self,
CYTHON_UNUSED PyObject *unused) {
11799 PyObject *__pyx_r = 0;
11800 __Pyx_RefNannyDeclarations
11801 __Pyx_RefNannySetupContext("clifford_hidden_doctests (wrapper)", 0);
11802 __pyx_r = __pyx_pf_8PyClical_8clifford_hidden_doctests(__pyx_self);
11803
11804 /* function exit code */
11805 __Pyx_RefNannyFinishContext();
11806 return __pyx_r;
11807 }
11808
11809 static PyObject *__pyx_pf_8PyClical_8clifford_hidden_doctests(CYTHON_UNUSED PyObject
*__pyx_self) {
11810 PyObject *__pyx_r = NULL;
11811 __Pyx_RefNannyDeclarations
11812 __Pyx_RefNannySetupContext("clifford_hidden_doctests", 0);
11813
11814 /* "PyClical.pyx":1335
11815 * True
11816 * """
11817 * return # ««««««««
11818 *
11819 * cpdef inline error_squared_tol(obj):
11820 */
11821 __Pyx_XDECREF(__pyx_r);
11822 __pyx_r = Py_None; __Pyx_INCREF(Py_None);
11823 goto __pyx_L0;
11824
11825 /* "PyClical.pyx":1253
11826 * return clifford_to_str(self.unwrap()).decode()
11827 *
11828 * def clifford_hidden_doctests(): # ««««««««
11829 * """
11830 * Tests for functions that Doctest cannot see.
11831 */
11832
11833 /* function exit code */
11834 __pyx_L0:;
11835 __Pyx_XGIVEREF(__pyx_r);
11836 __Pyx_RefNannyFinishContext();
11837 return __pyx_r;
11838 }
11839
11840 /* "PyClical.pyx":1337
11841 * return
11842 *
11843 * cpdef inline error_squared_tol(obj): # ««««««««
11844 * """
11845 * Quadratic norm error tolerance relative to a specific multivector.
11846 */
11847
11848 static PyObject *__pyx_pw_8PyClical_11error_squared_tol(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
11849 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_error_squared_tol(PyObject
*__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
11850 PyObject *__pyx_r = NULL;
11851 __Pyx_RefNannyDeclarations
11852 PyObject *__pyx_t_1 = NULL;
11853 int __pyx_lineno = 0;
11854 const char *__pyx_filename = NULL;

```

```

11855 int __pyx_clineno = 0;
11856 __Pyx_RefNannySetupContext("error_squared_tol", 0);
11857
11858 /* "PyClical.pyx":1344
11859 * 0.0
11860 * """
11861 * return glucat.error_squared_tol(toClifford(obj)) # ««««««««
11862 *
11863 * cpdef inline error_squared(lhs, rhs, threshold):
11864 */
11865 __Pyx_XDECREF(__pyx_r);
11866 __pyx_t_1 =
PyFloat_FromDouble(error_squared_tol(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1344, __pyx_L1_error)
11867 __Pyx_GOTREF(__pyx_t_1);
11868 __pyx_r = __pyx_t_1;
11869 __pyx_t_1 = 0;
11870 goto __pyx_L0;
11871
11872 /* "PyClical.pyx":1337
11873 * return
11874 *
11875 * cpdef inline error_squared_tol(obj): # ««««««««
11876 * """
11877 * Quadratic norm error tolerance relative to a specific multivector.
11878 */
11879
11880 /* function exit code */
11881 __pyx_L1_error;
11882 __Pyx_XDECREF(__pyx_t_1);
11883 __Pyx_AddTraceback("PyClical.error_squared_tol", __pyx_clineno, __pyx_lineno,
__pyx_filename);
11884 __pyx_r = 0;
11885 __pyx_L0;
11886 __Pyx_XGIVEREF(__pyx_r);
11887 __Pyx_RefNannyFinishContext();
11888 return __pyx_r;
11889 }
11890
11891 /* Python wrapper */
11892 static PyObject * __pyx_pw_8PyClical_11error_squared_tol(PyObject * __pyx_self, PyObject
* __pyx_v_obj); /*proto*/
11893 static char __pyx_doc_8PyClical_10error_squared_tol[] = "\n Quadratic norm error
tolerance relative to a specific multivector.\n\n >> print(error_squared_tol(clifford(\"{1}\") *
3.0 - error_squared_tol(clifford(\"{1}-2{2}+3{3}\")\n 0.0\n ";
11894 static PyObject * __pyx_pf_8PyClical_11error_squared_tol(PyObject * __pyx_self, PyObject
* __pyx_v_obj) {
11895 PyObject * __pyx_r = 0;
11896 __Pyx_RefNannyDeclarations
11897 __Pyx_RefNannySetupContext("error_squared_tol (wrapper)", 0);
11898 __pyx_r = __pyx_pf_8PyClical_10error_squared_tol(__pyx_self, ((PyObject
*) __pyx_v_obj));
11899
11900 /* function exit code */
11901 __Pyx_RefNannyFinishContext();
11902 return __pyx_r;
11903 }
11904
11905 static PyObject * __pyx_pf_8PyClical_10error_squared_tol(CYTHON_UNUSED PyObject
* __pyx_self, PyObject * __pyx_v_obj) {
11906 PyObject * __pyx_r = NULL;
11907 __Pyx_RefNannyDeclarations
11908 PyObject * __pyx_t_1 = NULL;
11909 int __pyx_lineno = 0;
11910 const char * __pyx_filename = NULL;
11911 int __pyx_clineno = 0;
11912 __Pyx_RefNannySetupContext("error_squared_tol", 0);
11913 __Pyx_XDECREF(__pyx_r);
11914 __pyx_t_1 = __pyx_f_8PyClical_error_squared_tol(__pyx_v_obj, 0); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1337, __pyx_L1_error)
11915 __Pyx_GOTREF(__pyx_t_1);
11916 __pyx_r = __pyx_t_1;
11917 __pyx_t_1 = 0;
11918 goto __pyx_L0;
11919
11920 /* function exit code */
11921 __pyx_L1_error;
11922 __Pyx_XDECREF(__pyx_t_1);
11923 __Pyx_AddTraceback("PyClical.error_squared_tol", __pyx_clineno, __pyx_lineno,
__pyx_filename);
11924 __pyx_r = NULL;
11925 __pyx_L0;
11926 __Pyx_XGIVEREF(__pyx_r);
11927 __Pyx_RefNannyFinishContext();
11928 return __pyx_r;
11929 }
11930

```

```

11931 /* "PyClicl.pyx":1346
11932 * return glucat.error_squared_tol(toClifford(obj))
11933 *
11934 * cpdef inline error_squared(lhs, rhs, threshold): # ««««««««
11935 * """
11936 * Relative or absolute error using the quadratic norm.
11937 */
11938
11939 static PyObject *__pyx_pw_8PyClicl_13error_squared(PyObject *__pyx_self, PyObject
11940 *__pyx_args, PyObject *__pyx_kwds); /*proto*/
11941 static CYTHON_INLINE PyObject *__pyx_f_8PyClicl_error_squared(PyObject *__pyx_v_lhs,
11942 PyObject *__pyx_v_rhs, PyObject *__pyx_v_threshold, CYTHON_UNUSED int __pyx_skip_dispatch) {
11943 PyObject *__pyx_r = NULL;
11944 __Pyx_RefNannyDeclarations
11945 scalar_t __pyx_t_1;
11946 PyObject *__pyx_t_2 = NULL;
11947 int __pyx_lineno = 0;
11948 const char *__pyx_filename = NULL;
11949 int __pyx_clineno = 0;
11950 __Pyx_RefNannySetupContext("error_squared", 0);
11951
11952 /* "PyClicl.pyx":1357
11953 * 25.0
11954 * """
11955 * return glucat.error_squared(toClifford(lhs), toClifford(rhs), <scalar_t>threshold)
11956 * # ««««««««
11957 *
11958 * cpdef inline approx_equal(lhs, rhs, threshold=None, tol=None):
11959 */
11960 __Pyx_XDECREF(__pyx_r);
11961 __pyx_t_1 = __pyx_PyFloat_AsDouble(__pyx_v_threshold); if (unlikely((__pyx_t_1 ==
11962 ((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 1357, __pyx_L1_error)
11963 __pyx_t_2 =
11964 PyFloat_FromDouble(error_squared(__pyx_f_8PyClicl_toClifford(__pyx_v_lhs),
11965 __pyx_f_8PyClicl_toClifford(__pyx_v_rhs), ((scalar_t)__pyx_t_1)); if (unlikely(!__pyx_t_2))
11966 __PYX_ERR(0, 1357, __pyx_L1_error)
11967 __Pyx_GOTREF(__pyx_t_2);
11968 __pyx_r = __pyx_t_2;
11969 __pyx_t_2 = 0;
11970 goto __pyx_L0;
11971
11972 /* "PyClicl.pyx":1346
11973 * return glucat.error_squared_tol(toClifford(obj))
11974 *
11975 * cpdef inline error_squared(lhs, rhs, threshold): # ««««««««
11976 * """
11977 * Relative or absolute error using the quadratic norm.
11978 */
11979
11980 /* function exit code */
11981 __pyx_L1_error;
11982 __Pyx_XDECREF(__pyx_t_2);
11983 __Pyx_AddTraceback("PyClicl.error_squared", __pyx_clineno, __pyx_lineno,
11984 __pyx_filename);
11985 __pyx_r = 0;
11986 __pyx_L0;
11987 __Pyx_XGIVEREF(__pyx_r);
11988 __Pyx_RefNannyFinishContext();
11989 return __pyx_r;
11990 }
11991
11992 /* Python wrapper */
11993 static PyObject *__pyx_pw_8PyClicl_13error_squared(PyObject *__pyx_self, PyObject
11994 *__pyx_args, PyObject *__pyx_kwds); /*proto*/
11995 static char __pyx_doc_8PyClicl_12error_squared[] = "\n Relative or absolute error
11996 using the quadratic norm.\n\n >> err2=scalar_epsilon*scalar_epsilon\n\n >>
11997 print(error_squared(clifford(\"{1}\"), clifford(\"{1}\"), err2))\n 0.0\n >>
11998 print(error_squared(clifford(\"{1}{2}-3{2}+4{3}\"), clifford(\"{1}\"), err2))\n 25.0\n ";
11999 static PyObject *__pyx_pw_8PyClicl_13error_squared(PyObject *__pyx_self, PyObject
12000 *__pyx_args, PyObject *__pyx_kwds) {
12001 PyObject *__pyx_v_lhs = 0;
12002 PyObject *__pyx_v_rhs = 0;
12003 PyObject *__pyx_v_threshold = 0;
12004 int __pyx_lineno = 0;
12005 const char *__pyx_filename = NULL;
12006 int __pyx_clineno = 0;
12007 PyObject *__pyx_r = 0;
12008 __Pyx_RefNannyDeclarations
12009 __Pyx_RefNannySetupContext("error_squared (wrapper)", 0);
12010 {
12011 static PyObject *__pyx_pyargnames[] =
12012 {&__pyx_n_s_lhs,&__pyx_n_s_rhs,&__pyx_n_s_threshold,0};
12013 PyObject* values[3] = {0,0,0};
12014 if (unlikely(__pyx_kwds)) {
12015 Py_ssize_t kw_args;
12016 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
12017 switch (pos_args) {

```



```

12004 case 3: values[2] = PyTuple_GET_ITEM(__pyx_args, 2);
12005 CYTHON_FALLTHROUGH;
12006 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
12007 CYTHON_FALLTHROUGH;
12008 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
12009 CYTHON_FALLTHROUGH;
12010 case 0: break;
12011 default: goto __pyx_L5_argtuple_error;
12012 }
12013 kw_args = PyDict_Size(__pyx_kwds);
12014 switch (pos_args) {
12015 case 0:
12016 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_lhs)) !=
12017 0)) kw_args--;
12018 else goto __pyx_L5_argtuple_error;
12019 CYTHON_FALLTHROUGH;
12020 case 1:
12021 if (likely((values[1] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_rhs)) !=
12022 0)) kw_args--;
12023 else {
12024 __Pyx_RaiseArgtupleInvalid("error_squared", 1, 3, 3, 1); __PYX_ERR(0, 1346,
12025 __pyx_L3_error)
12026 }
12027 CYTHON_FALLTHROUGH;
12028 case 2:
12029 if (likely((values[2] = __Pyx_PyDict_GetItemStr(__pyx_kwds,
12030 __pyx_n_s_threshold)) != 0)) kw_args--;
12031 else {
12032 __Pyx_RaiseArgtupleInvalid("error_squared", 1, 3, 3, 2); __PYX_ERR(0, 1346,
12033 __pyx_L3_error)
12034 }
12035 }
12036 if (unlikely(kw_args > 0)) {
12037 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
12038 values, pos_args, "error_squared") < 0)) __PYX_ERR(0, 1346, __pyx_L3_error)
12039 }
12040 } else if (PyTuple_GET_SIZE(__pyx_args) != 3) {
12041 goto __pyx_L5_argtuple_error;
12042 } else {
12043 values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
12044 values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
12045 values[2] = PyTuple_GET_ITEM(__pyx_args, 2);
12046 }
12047 __pyx_v_lhs = values[0];
12048 __pyx_v_rhs = values[1];
12049 __pyx_v_threshold = values[2];
12050 }
12051 goto __pyx_L4_argument_unpacking_done;
12052 __pyx_L5_argtuple_error:;
12053 __Pyx_RaiseArgtupleInvalid("error_squared", 1, 3, 3, PyTuple_GET_SIZE(__pyx_args));
12054 __PYX_ERR(0, 1346, __pyx_L3_error)
12055 __pyx_L3_error:;
12056 __Pyx_AddTraceback("PyClical.error_squared", __pyx_clineno, __pyx_lineno,
12057 __pyx_filename);
12058 __Pyx_RefNannyFinishContext();
12059 return NULL;
12060 __pyx_L4_argument_unpacking_done:;
12061 __pyx_r = __pyx_pf_8PyClical_12error_squared(__pyx_self, __pyx_v_lhs, __pyx_v_rhs,
12062 __pyx_v_threshold);
12063
12064 /* function exit code */
12065 __Pyx_RefNannyFinishContext();
12066 return __pyx_r;
12067 }
12068
12069 static PyObject * __pyx_pf_8PyClical_12error_squared(CYTHON_UNUSED PyObject
12070 * __pyx_self, PyObject * __pyx_v_lhs, PyObject * __pyx_v_rhs, PyObject * __pyx_v_threshold) {
12071 PyObject * __pyx_r = NULL;
12072 __Pyx_RefNannyDeclarations
12073 PyObject * __pyx_t_1 = NULL;
12074 int __pyx_lineno = 0;
12075 const char * __pyx_filename = NULL;
12076 int __pyx_clineno = 0;
12077 __Pyx_RefNannySetupContext("error_squared", 0);
12078 __Pyx_XDECREF(__pyx_r);
12079 __pyx_t_1 = __pyx_f_8PyClical_error_squared(__pyx_v_lhs, __pyx_v_rhs,
12080 __pyx_v_threshold, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1346, __pyx_L1_error)
12081 __Pyx_GOTREF(__pyx_t_1);
12082 __pyx_r = __pyx_t_1;
12083 __pyx_t_1 = 0;
12084 goto __pyx_L0;
12085
12086 /* function exit code */
12087 __pyx_L1_error:;
12088 __Pyx_XDECREF(__pyx_t_1);
12089 __Pyx_AddTraceback("PyClical.error_squared", __pyx_clineno, __pyx_lineno,
12090 __pyx_filename);

```

```

12079 __pyx_r = NULL;
12080 __pyx_L0:;
12081 __Pyx_XGIVEREF(__pyx_r);
12082 __Pyx_RefNannyFinishContext();
12083 return __pyx_r;
12084 }
12085
12086 /* "PyClicl.pyx":1359
12087 * return glucat.error_squared(toClifford(lhs), toClifford(rhs), <scalar_t>threshold)
12088 *
12089 * cpdef inline approx_equal(lhs, rhs, threshold=None, tol=None): # ««««««««
12090 * """
12091 * Test for approximate equality of multivectors.
12092 */
12093
12094 static PyObject *__pyx_pw_8PyClicl_15approx_equal(PyObject *__pyx_self, PyObject
12095 *__pyx_args, PyObject *__pyx_kws); /*proto*/
12096 static CYTHON_INLINE PyObject *__pyx_f_8PyClicl_approx_equal(PyObject *__pyx_v_lhs,
12097 PyObject *__pyx_v_rhs, CYTHON_UNUSED int __pyx_skip_dispatch, struct
12098 __pyx_opt_args_8PyClicl_approx_equal *__pyx_optional_args) {
12099 PyObject *__pyx_v_threshold = ((PyObject *)Py_None);
12100 PyObject *__pyx_v_tol = ((PyObject *)Py_None);
12101 PyObject *__pyx_r = NULL;
12102 __Pyx_RefNannyDeclarations
12103 PyObject *__pyx_t_1 = NULL;
12104 int __pyx_t_2;
12105 PyObject *__pyx_t_3 = NULL;
12106 scalar_t __pyx_t_4;
12107 scalar_t __pyx_t_5;
12108 int __pyx_lineno = 0;
12109 const char *__pyx_filename = NULL;
12110 int __pyx_clineno = 0;
12111 __Pyx_RefNannySetupContext("approx_equal", 0);
12112 if (__pyx_optional_args) {
12113 if (__pyx_optional_args->__pyx_n > 0) {
12114 __pyx_v_threshold = __pyx_optional_args->threshold;
12115 if (__pyx_optional_args->__pyx_n > 1) {
12116 __pyx_v_tol = __pyx_optional_args->tol;
12117 }
12118 }
12119 __Pyx_INCREF(__pyx_v_threshold);
12120 __Pyx_INCREF(__pyx_v_tol);
12121
12122 /* "PyClicl.pyx":1374
12123 * True
12124 * """
12125 * threshold = error_squared_tol(rhs) if threshold is None else threshold # ««««««««
12126 * tol = error_squared_tol(rhs) if tol is None else tol
12127 * return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold,
12128 <scalar_t>tol)
12129 */
12130 __pyx_t_2 = (__pyx_v_threshold == Py_None);
12131 if ((__pyx_t_2 != 0)) {
12132 __pyx_t_3 = __pyx_f_8PyClicl_error_squared_tol(__pyx_v_rhs, 0); if
12133 (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1374, __pyx_L1_error)
12134 __Pyx_GOTREF(__pyx_t_3);
12135 __pyx_t_1 = __pyx_t_3;
12136 __pyx_t_3 = 0;
12137 } else {
12138 __Pyx_INCREF(__pyx_v_threshold);
12139 __pyx_t_1 = __pyx_v_threshold;
12140 }
12141 __Pyx_DECREF_SET(__pyx_v_threshold, __pyx_t_1);
12142 __pyx_t_1 = 0;
12143
12144 /* "PyClicl.pyx":1375
12145 * """
12146 * threshold = error_squared_tol(rhs) if threshold is None else threshold
12147 * tol = error_squared_tol(rhs) if tol is None else tol # ««««««««
12148 * return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold,
12149 <scalar_t>tol)
12150 */
12151 __pyx_t_2 = (__pyx_v_tol == Py_None);
12152 if ((__pyx_t_2 != 0)) {
12153 __pyx_t_3 = __pyx_f_8PyClicl_error_squared_tol(__pyx_v_rhs, 0); if
12154 (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1375, __pyx_L1_error)
12155 __Pyx_GOTREF(__pyx_t_3);
12156 __pyx_t_1 = __pyx_t_3;
12157 __pyx_t_3 = 0;
12158 } else {
12159 __Pyx_INCREF(__pyx_v_tol);
12160 __pyx_t_1 = __pyx_v_tol;
12161 }
12162 __Pyx_DECREF_SET(__pyx_v_tol, __pyx_t_1);
12163 __pyx_t_1 = 0;

```

```

12159
12160 /* "PyClical.pyx":1376
12161 * threshold = error_squared_tol(rhs) if threshold is None else threshold
12162 * tol = error_squared_tol(rhs) if tol is None else tol
12163 * return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold,
12164 * # ««««««««
12165 * cpdef inline inv(obj):
12166 */
12167 __Pyx_XDECREF(__pyx_r);
12168 __pyx_t_4 = __pyx_PyFloat_AsDouble(__pyx_v_threshold); if (unlikely((__pyx_t_4 ==
12169 ((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 1376, __pyx_L1_error)
12169 __pyx_t_5 = __pyx_PyFloat_AsDouble(__pyx_v_tol); if (unlikely((__pyx_t_5 ==
12170 ((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 1376, __pyx_L1_error)
12170 __pyx_t_1 =
__Pyx_PyBool_FromLong(approx_equal(__pyx_f_8PyClical_toClifford(__pyx_v_lhs),
__pyx_f_8PyClical_toClifford(__pyx_v_rhs), ((scalar_t)__pyx_t_4), ((scalar_t)__pyx_t_5))); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1376, __pyx_L1_error)
12171 __Pyx_GOTREF(__pyx_t_1);
12172 __pyx_r = __pyx_t_1;
12173 __pyx_t_1 = 0;
12174 goto __pyx_L0;
12175
12176 /* "PyClical.pyx":1359
12177 * return glucat.error_squared(toClifford(lhs), toClifford(rhs), <scalar_t>threshold)
12178 *
12179 * cpdef inline approx_equal(lhs, rhs, threshold=None, tol=None):
12180 * """
12181 * Test for approximate equality of multivectors.
12182 */
12183
12184 /* function exit code */
12185 __pyx_L1_error:;
12186 __Pyx_XDECREF(__pyx_t_1);
12187 __Pyx_XDECREF(__pyx_t_3);
12188 __Pyx_AddTraceback("PyClical.approx_equal", __pyx_clineno, __pyx_lineno,
__pyx_filename);
12189 __pyx_r = 0;
12190 __pyx_L0:;
12191 __Pyx_XDECREF(__pyx_v_threshold);
12192 __Pyx_XDECREF(__pyx_v_tol);
12193 __Pyx_XGIVEREF(__pyx_r);
12194 __Pyx_RefNannyFinishContext();
12195 return __pyx_r;
12196 }
12197
12198 /* Python wrapper */
12199 static PyObject * __pyx_pw_8PyClical_15approx_equal(PyObject * __pyx_self, PyObject
* __pyx_args, PyObject * __pyx_kwds); /*proto*/
12200 static char __pyx_doc_8PyClical_14approx_equal[] = "\n Test for approximate
equality of multivectors.\n\n >> err2=scalar_epsilon*scalar_epsilon\n\n >>
print(approx_equal(clifford(\"1{1}\", clifford(\"1{1}\")))\n True\n >>
print(approx_equal(clifford(\"1{1}-3{2}+4{3}\", clifford(\"1{1}\")))\n False\n >>
print(approx_equal(clifford(\"1{1}-3{2}+4{3}+0.001\", clifford(\"1{1}-3{2}+4{3}\"), err2, err2))\n
False\n >> print(approx_equal(clifford(\"1{1}-3{2}+4{3}+1.0e-30\", clifford(\"1{1}-3{2}+4{3}\"),
err2, err2))\n True\n ";
12201 static PyObject * __pyx_pw_8PyClical_15approx_equal(PyObject * __pyx_self, PyObject
* __pyx_args, PyObject * __pyx_kwds) {
12202 PyObject * __pyx_v_lhs = 0;
12203 PyObject * __pyx_v_rhs = 0;
12204 PyObject * __pyx_v_threshold = 0;
12205 PyObject * __pyx_v_tol = 0;
12206 int __pyx_lineno = 0;
12207 const char * __pyx_filename = NULL;
12208 int __pyx_clineno = 0;
12209 PyObject * __pyx_r = 0;
12210 __Pyx_RefNannyDeclarations
12211 __Pyx_RefNannySetupContext("approx_equal (wrapper)", 0);
12212 {
12213 static PyObject * __pyx_pyargnames[] =
12214 {&__pyx_n_s_lhs,&__pyx_n_s_rhs,&__pyx_n_s_threshold,&__pyx_n_s_tol,0};
12214 PyObject* values[4] = {0,0,0,0};
12215 values[2] = ((PyObject *)Py_None);
12216 values[3] = ((PyObject *)Py_None);
12217 if (unlikely(__pyx_kwds)) {
12218 Py_ssize_t kw_args;
12219 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
12220 switch (pos_args) {
12221 case 4: values[3] = PyTuple_GET_ITEM(__pyx_args, 3);
12222 CYTHON_FALLTHROUGH;
12223 case 3: values[2] = PyTuple_GET_ITEM(__pyx_args, 2);
12224 CYTHON_FALLTHROUGH;
12225 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
12226 CYTHON_FALLTHROUGH;
12227 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
12228 CYTHON_FALLTHROUGH;
12229 case 0: break;

```

```

12230 default: goto __pyx_L5_argtuple_error;
12231 }
12232 kw_args = PyDict_Size(__pyx_kwds);
12233 switch (pos_args) {
12234 case 0:
12235 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_lhs)) !=
0)) kw_args--;
12236 else goto __pyx_L5_argtuple_error;
12237 CYTHON_FALLTHROUGH;
12238 case 1:
12239 if (likely((values[1] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_rhs)) !=
0)) kw_args--;
12240 else {
12241 __Pyx_RaiseArgtupleInvalid("approx_equal", 0, 2, 4, 1); __PYX_ERR(0, 1359,
__pyx_L3_error)
12242 }
12243 CYTHON_FALLTHROUGH;
12244 case 2:
12245 if (kw_args > 0) {
12246 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_threshold);
12247 if (value) { values[2] = value; kw_args--; }
12248 }
12249 CYTHON_FALLTHROUGH;
12250 case 3:
12251 if (kw_args > 0) {
12252 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_tol);
12253 if (value) { values[3] = value; kw_args--; }
12254 }
12255 }
12256 if (unlikely(kw_args > 0)) {
12257 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "approx_equal") < 0)) __PYX_ERR(0, 1359, __pyx_L3_error)
12258 }
12259 } else {
12260 switch (PyTuple_GET_SIZE(__pyx_args)) {
12261 case 4: values[3] = PyTuple_GET_ITEM(__pyx_args, 3);
12262 CYTHON_FALLTHROUGH;
12263 case 3: values[2] = PyTuple_GET_ITEM(__pyx_args, 2);
12264 CYTHON_FALLTHROUGH;
12265 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
12266 values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
12267 break;
12268 default: goto __pyx_L5_argtuple_error;
12269 }
12270 }
12271 __pyx_v_lhs = values[0];
12272 __pyx_v_rhs = values[1];
12273 __pyx_v_threshold = values[2];
12274 __pyx_v_tol = values[3];
12275 }
12276 goto __pyx_L4_argument_unpacking_done;
12277 __pyx_L5_argtuple_error:;
12278 __Pyx_RaiseArgtupleInvalid("approx_equal", 0, 2, 4, PyTuple_GET_SIZE(__pyx_args));
__PYX_ERR(0, 1359, __pyx_L3_error)
12279 __pyx_L3_error:;
12280 __Pyx_AddTraceback("PyClical.approx_equal", __pyx_clineno, __pyx_lineno,
__pyx_filename);
12281 __Pyx_RefNannyFinishContext();
12282 return NULL;
12283 __pyx_L4_argument_unpacking_done:;
12284 __pyx_r = __pyx_pf_8PyClical_14approx_equal(__pyx_self, __pyx_v_lhs, __pyx_v_rhs,
__pyx_v_threshold, __pyx_v_tol);
12285
12286 /* function exit code */
12287 __Pyx_RefNannyFinishContext();
12288 return __pyx_r;
12289 }
12290
12291 static PyObject * __pyx_pf_8PyClical_14approx_equal(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_lhs, PyObject *__pyx_v_rhs, PyObject *__pyx_v_threshold, PyObject *__pyx_v_tol) {
12292 PyObject *__pyx_r = NULL;
12293 __Pyx_RefNannyDeclarations
12294 PyObject *__pyx_t_1 = NULL;
12295 struct __pyx_opt_args_8PyClical_approx_equal __pyx_t_2;
12296 int __pyx_lineno = 0;
12297 const char *__pyx_filename = NULL;
12298 int __pyx_clineno = 0;
12299 __Pyx_RefNannySetupContext("approx_equal", 0);
12300 __Pyx_XDECREF(__pyx_r);
12301 __pyx_t_2.__pyx_n = 2;
12302 __pyx_t_2.threshold = __pyx_v_threshold;
12303 __pyx_t_2.tol = __pyx_v_tol;
12304 __pyx_t_1 = __pyx_f_8PyClical_approx_equal(__pyx_v_lhs, __pyx_v_rhs, 0, &__pyx_t_2);
 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1359, __pyx_L1_error)
12305 __Pyx_GOTREF(__pyx_t_1);
12306 __pyx_r = __pyx_t_1;
12307 __pyx_t_1 = 0;

```

```

12308 goto __pyx_L0;
12309
12310 /* function exit code */
12311 __pyx_L1_error++;
12312 __Pyx_XDECREF(__pyx_t_1);
12313 __Pyx_AddTraceback("PyClical.approx_equal", __pyx_clineno, __pyx_lineno,
__pyx_filename);
12314 __pyx_r = NULL;
12315 __pyx_L0;
12316 __Pyx_XGIVEREF(__pyx_r);
12317 __Pyx_RefNannyFinishContext();
12318 return __pyx_r;
12319 }
12320
12321 /* "PyClical.pyx":1378
12322 * return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold,
<scalar_t>tol)
12323 *
12324 * cpdef inline inv(obj): # ««««««««
12325 * """
12326 * Geometric multiplicative inverse.
12327 */
12328
12329 static PyObject * __pyx_pw_8PyClical_17inv(PyObject * __pyx_self, PyObject
* __pyx_v_obj); /*proto*/
12330 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_inv(PyObject * __pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
12331 PyObject * __pyx_r = NULL;
12332 __Pyx_RefNannyDeclarations
12333 PyObject * __pyx_t_1 = NULL;
12334 PyObject * __pyx_t_2 = NULL;
12335 PyObject * __pyx_t_3 = NULL;
12336 int __pyx_lineno = 0;
12337 const char * __pyx_filename = NULL;
12338 int __pyx_clineno = 0;
12339 __Pyx_RefNannySetupContext("inv", 0);
12340
12341 /* "PyClical.pyx":1391
12342 * nan
12343 * """
12344 * return clifford(obj).inv() # ««««««««
12345 *
12346 * cpdef inline scalar(obj):
12347 */
12348 __Pyx_XDECREF(__pyx_r);
12349 __pyx_t_2 = __Pyx_PyObject_CallOneArg((PyObject *) __pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1391, __pyx_L1_error)
12350 __Pyx_GOTREF(__pyx_t_2);
12351 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_inv); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1391, __pyx_L1_error)
12352 __Pyx_GOTREF(__pyx_t_3);
12353 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
12354 __pyx_t_2 = NULL;
12355 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
12356 __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
12357 if (likely(__pyx_t_2)) {
12358 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
12359 __Pyx_INCREF(__pyx_t_2);
12360 __Pyx_INCREF(function);
12361 __Pyx_DECREF_SET(__pyx_t_3, function);
12362 }
12363 }
12364 __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
12365 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
12366 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1391, __pyx_L1_error)
12367 __Pyx_GOTREF(__pyx_t_1);
12368 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
12369 __pyx_r = __pyx_t_1;
12370 __pyx_t_1 = 0;
12371 goto __pyx_L0;
12372
12373 /* "PyClical.pyx":1378
12374 * return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold,
<scalar_t>tol)
12375 *
12376 * cpdef inline inv(obj): # ««««««««
12377 * """
12378 * Geometric multiplicative inverse.
12379 */
12380
12381 /* function exit code */
12382 __pyx_L1_error++;
12383 __Pyx_XDECREF(__pyx_t_1);
12384 __Pyx_XDECREF(__pyx_t_2);
12385 __Pyx_XDECREF(__pyx_t_3);
12386 __Pyx_AddTraceback("PyClical.inv", __pyx_clineno, __pyx_lineno, __pyx_filename);

```

```

12387 __pyx_r = 0;
12388 __pyx_L0:;
12389 __Pyx_XGIVEREF(__pyx_r);
12390 __Pyx_RefNannyFinishContext();
12391 return __pyx_r;
12392 }
12393
12394 /* Python wrapper */
12395 static PyObject *__pyx_pw_8PyClical_17inv(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
12396 static char __pyx_doc_8PyClical_16inv[] = "\n Geometric multiplicative inverse.\n\n
>> print(inv(clifford(\"{1}\")))\n {1}\n >> print(inv(clifford(\"{-1}\")))\n {-1}\n >>
print(inv(clifford(\"{-2,-1}\")))\n {-2,-1}\n >> print(inv(clifford(\"{-1}+{1}\")))\n nan\n
";
12397 static PyObject *__pyx_pw_8PyClical_17inv(PyObject *__pyx_self, PyObject *__pyx_v_obj)
{
12398 PyObject *__pyx_r = 0;
12399 __Pyx_RefNannyDeclarations
12400 __Pyx_RefNannySetupContext("inv (wrapper)", 0);
12401 __pyx_r = __pyx_pf_8PyClical_16inv(__pyx_self, ((PyObject *)__pyx_v_obj));
12402
12403 /* function exit code */
12404 __Pyx_RefNannyFinishContext();
12405 return __pyx_r;
12406 }
12407
12408 static PyObject *__pyx_pf_8PyClical_16inv(CYTHON_UNUSED PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
12409 PyObject *__pyx_r = NULL;
12410 __Pyx_RefNannyDeclarations
12411 PyObject *__pyx_t_1 = NULL;
12412 int __pyx_lineno = 0;
12413 const char *__pyx_filename = NULL;
12414 int __pyx_clineno = 0;
12415 __Pyx_RefNannySetupContext("inv", 0);
12416 __Pyx_XDECREF(__pyx_r);
12417 __pyx_t_1 = __pyx_f_8PyClical_inv(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1378, __pyx_L1_error)
12418 __Pyx_GOTREF(__pyx_t_1);
12419 __pyx_r = __pyx_t_1;
12420 __pyx_t_1 = 0;
12421 goto __pyx_L0;
12422
12423 /* function exit code */
12424 __pyx_L1_error:;
12425 __Pyx_XDECREF(__pyx_t_1);
12426 __Pyx_AddTraceback("PyClical.inv", __pyx_clineno, __pyx_lineno, __pyx_filename);
12427 __pyx_r = NULL;
12428 __pyx_L0:;
12429 __Pyx_XGIVEREF(__pyx_r);
12430 __Pyx_RefNannyFinishContext();
12431 return __pyx_r;
12432 }
12433
12434 /* "PyClical.pyx":1393
12435 * return clifford(obj).inv()
12436 *
12437 * cpdef inline scalar(obj): # ««««««
12438 * """
12439 * Scalar part.
12440 */
12441
12442 static PyObject *__pyx_pw_8PyClical_19scalar(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
12443 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_scalar(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
12444 PyObject *__pyx_r = NULL;
12445 __Pyx_RefNannyDeclarations
12446 PyObject *__pyx_t_1 = NULL;
12447 PyObject *__pyx_t_2 = NULL;
12448 PyObject *__pyx_t_3 = NULL;
12449 int __pyx_lineno = 0;
12450 const char *__pyx_filename = NULL;
12451 int __pyx_clineno = 0;
12452 __Pyx_RefNannySetupContext("scalar", 0);
12453
12454 /* "PyClical.pyx":1402
12455 * 0.0
12456 * """
12457 * return clifford(obj).scalar() # ««««««
12458 *
12459 * cpdef inline real(obj):
12460 */
12461 __Pyx_XDECREF(__pyx_r);
12462 __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1402, __pyx_L1_error)
12463 __Pyx_GOTREF(__pyx_t_2);

```

```

12464 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_scalar); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1402, __pyx_L1_error)
12465 __Pyx_GOTREF(__pyx_t_3);
12466 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
12467 __pyx_t_2 = NULL;
12468 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
12469 __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
12470 if (likely(__pyx_t_2)) {
12471 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
12472 __Pyx_INCREF(__pyx_t_2);
12473 __Pyx_INCREF(function);
12474 __Pyx_DECREF_SET(__pyx_t_3, function);
12475 }
12476 }
12477 __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
12478 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
12479 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1402, __pyx_L1_error)
12480 __Pyx_GOTREF(__pyx_t_1);
12481 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
12482 __pyx_r = __pyx_t_1;
12483 __pyx_t_1 = 0;
12484 goto __pyx_L0;
12485
12486 /* "PyClical.pyx":1393
12487 * return clifford(obj).inv()
12488 *
12489 * cpdef inline scalar(obj): # ««««««««
12490 * """
12491 * Scalar part.
12492 */
12493
12494 /* function exit code */
12495 __pyx_L1_error++;
12496 __Pyx_XDECREF(__pyx_t_1);
12497 __Pyx_XDECREF(__pyx_t_2);
12498 __Pyx_XDECREF(__pyx_t_3);
12499 __Pyx_AddTraceback("PyClical.scalar", __pyx_clineno, __pyx_lineno, __pyx_filename);
12500 __pyx_r = 0;
12501 __pyx_L0++;
12502 __Pyx_XGIVEREF(__pyx_r);
12503 __Pyx_RefNannyFinishContext();
12504 return __pyx_r;
12505 }
12506
12507 /* Python wrapper */
12508 static PyObject * __pyx_pw_8PyClical_19scalar(PyObject * __pyx_self, PyObject
*__pyx_v_obj); /*proto*/
12509 static char __pyx_doc_8PyClical_18scalar[] = "\n Scalar part.\n\n >>
scalar(clifford(\"1+{1}+{1,2}\"))\n 1.0\n >> scalar(clifford(\"{1,2}\"))\n 0.0\n ";
12510 static PyObject * __pyx_pw_8PyClical_19scalar(PyObject * __pyx_self, PyObject
*__pyx_v_obj) {
12511 PyObject * __pyx_r = 0;
12512 __Pyx_RefNannyDeclarations
12513 __Pyx_RefNannySetupContext("scalar (wrapper)", 0);
12514 __pyx_r = __pyx_pf_8PyClical_18scalar(__pyx_self, ((PyObject *) __pyx_v_obj));
12515
12516 /* function exit code */
12517 __Pyx_RefNannyFinishContext();
12518 return __pyx_r;
12519 }
12520
12521 static PyObject * __pyx_pf_8PyClical_18scalar(CYTHON_UNUSED PyObject * __pyx_self,
PyObject * __pyx_v_obj) {
12522 PyObject * __pyx_r = NULL;
12523 __Pyx_RefNannyDeclarations
12524 PyObject * __pyx_t_1 = NULL;
12525 int __pyx_lineno = 0;
12526 const char * __pyx_filename = NULL;
12527 int __pyx_clineno = 0;
12528 __Pyx_RefNannySetupContext("scalar", 0);
12529 __Pyx_XDECREF(__pyx_r);
12530 __pyx_t_1 = __pyx_f_8PyClical_scalar(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1393, __pyx_L1_error)
12531 __Pyx_GOTREF(__pyx_t_1);
12532 __pyx_r = __pyx_t_1;
12533 __pyx_t_1 = 0;
12534 goto __pyx_L0;
12535
12536 /* function exit code */
12537 __pyx_L1_error++;
12538 __Pyx_XDECREF(__pyx_t_1);
12539 __Pyx_AddTraceback("PyClical.scalar", __pyx_clineno, __pyx_lineno, __pyx_filename);
12540 __pyx_r = NULL;
12541 __pyx_L0++;
12542 __Pyx_XGIVEREF(__pyx_r);
12543 __Pyx_RefNannyFinishContext();

```

```

12544 return __pyx_r;
12545 }
12546
12547 /* "PyClical.pyx":1404
12548 * return clifford(obj).scalar()
12549 *
12550 * cpdef inline real(obj):
12551 * """
12552 * Real part: synonym for scalar part.
12553 */
12554
12555 static PyObject * __pyx_pw_8PyClical_21real(PyObject * __pyx_self, PyObject
*__pyx_v_obj); /*proto*/
12556 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_real(PyObject * __pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
12557 PyObject * __pyx_r = NULL;
12558 __Pyx_RefNannyDeclarations
12559 PyObject * __pyx_t_1 = NULL;
12560 PyObject * __pyx_t_2 = NULL;
12561 PyObject * __pyx_t_3 = NULL;
12562 int __pyx_lineno = 0;
12563 const char * __pyx_filename = NULL;
12564 int __pyx_clineno = 0;
12565 __Pyx_RefNannySetupContext("real", 0);
12566
12567 /* "PyClical.pyx":1413
12568 * 0.0
12569 * """
12570 * return clifford(obj).scalar()
12571 *
12572 * cpdef inline imag(obj):
12573 */
12574 __Pyx_XDECREF(__pyx_r);
12575 __pyx_t_2 = __Pyx_PyObject_CallOneArg((PyObject *) __pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1413, __pyx_L1_error)
12576 __Pyx_GOTREF(__pyx_t_2);
12577 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_scalar); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1413, __pyx_L1_error)
12578 __Pyx_GOTREF(__pyx_t_3);
12579 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
12580 __pyx_t_2 = NULL;
12581 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
12582 __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
12583 if (likely(__pyx_t_2)) {
12584 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
12585 __Pyx_INCREF(__pyx_t_2);
12586 __Pyx_INCREF(function);
12587 __Pyx_DECREF_SET(__pyx_t_3, function);
12588 }
12589 }
12590 __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
12591 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
12592 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1413, __pyx_L1_error)
12593 __Pyx_GOTREF(__pyx_t_1);
12594 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
12595 __pyx_r = __pyx_t_1;
12596 __pyx_t_1 = 0;
12597 goto __pyx_L0;
12598
12599 /* "PyClical.pyx":1404
12600 * return clifford(obj).scalar()
12601 *
12602 * cpdef inline real(obj):
12603 * """
12604 * Real part: synonym for scalar part.
12605 */
12606
12607 /* function exit code */
12608 __pyx_L1_error:;
12609 __Pyx_XDECREF(__pyx_t_1);
12610 __Pyx_XDECREF(__pyx_t_2);
12611 __Pyx_XDECREF(__pyx_t_3);
12612 __Pyx_AddTraceback("PyClical.real", __pyx_clineno, __pyx_lineno, __pyx_filename);
12613 __pyx_r = 0;
12614 __pyx_L0:;
12615 __Pyx_XGIVEREF(__pyx_r);
12616 __Pyx_RefNannyFinishContext();
12617 return __pyx_r;
12618 }
12619
12620 /* Python wrapper */
12621 static PyObject * __pyx_pw_8PyClical_21real(PyObject * __pyx_self, PyObject
*__pyx_v_obj); /*proto*/
12622 static char __pyx_doc_8PyClical_20real[] = "\n Real part: synonym for scalar
part.\n\n >> real(clifford(\"1+{1}+{1,2}\"))\n 1.0\n >> real(clifford(\"{1,2}\"))\n 0.0\n";

```



```

12623 static PyObject *__pyx_pw_8PyClical_21real(PyObject *__pyx_self, PyObject
12624 *__pyx_v_obj) {
12625 PyObject *__pyx_r = 0;
12626 __Pyx_RefNannyDeclarations
12627 __Pyx_RefNannySetupContext("real (wrapper)", 0);
12628 __pyx_r = __pyx_pf_8PyClical_20real(__pyx_self, (PyObject *)__pyx_v_obj));
12629
12630 /* function exit code */
12631 __Pyx_RefNannyFinishContext();
12632 return __pyx_r;
12633 }
12634
12635 static PyObject *__pyx_pf_8PyClical_20real(CYTHON_UNUSED PyObject *__pyx_self,
12636 PyObject *__pyx_v_obj) {
12637 PyObject *__pyx_r = NULL;
12638 __Pyx_RefNannyDeclarations
12639 PyObject *__pyx_t_1 = NULL;
12640 int __pyx_lineno = 0;
12641 const char *__pyx_filename = NULL;
12642 int __pyx_clineno = 0;
12643 __Pyx_RefNannySetupContext("real", 0);
12644 __Pyx_XDECREF(__pyx_r);
12645 __pyx_t_1 = __pyx_f_8PyClical_real(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
12646 __PYX_ERR(0, 1404, __pyx_L1_error)
12647 __Pyx_GOTREF(__pyx_t_1);
12648 __pyx_r = __pyx_t_1;
12649 __pyx_t_1 = 0;
12650 goto __pyx_L0;
12651
12652 /* function exit code */
12653 __pyx_L1_error:;
12654 __Pyx_XDECREF(__pyx_t_1);
12655 __Pyx_AddTraceback("PyClical.real", __pyx_clineno, __pyx_lineno, __pyx_filename);
12656 __pyx_r = NULL;
12657 __pyx_L0:;
12658 __Pyx_XGIVEREF(__pyx_r);
12659 __Pyx_RefNannyFinishContext();
12660 return __pyx_r;
12661 }
12662
12663 /* "PyClical.pyx":1415
12664 * return clifford(obj).scalar()
12665 *
12666 * cpdef inline imag(obj):
12667 * """
12668 * Imaginary part: deprecated (always 0).
12669 */
12670
12671 static PyObject *__pyx_pw_8PyClical_23imag(PyObject *__pyx_self, PyObject
12672 *__pyx_v_obj); /*proto*/
12673 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_imag(CYTHON_UNUSED PyObject
12674 *__pyx_v_obj, CYTHON_UNUSED int __pyx_skip_dispatch) {
12675 PyObject *__pyx_r = NULL;
12676 __Pyx_RefNannyDeclarations
12677 __Pyx_RefNannySetupContext("imag", 0);
12678
12679 /* "PyClical.pyx":1424
12680 * 0.0
12681 * """
12682 * return 0.0
12683 * # ««««««««
12684 *
12685 * cpdef inline pure(obj):
12686 */
12687 __Pyx_XDECREF(__pyx_r);
12688 __Pyx_INCREF(__pyx_float_0_0);
12689 __pyx_r = __pyx_float_0_0;
12690 goto __pyx_L0;
12691
12692 /* "PyClical.pyx":1415
12693 * return clifford(obj).scalar()
12694 *
12695 * cpdef inline imag(obj):
12696 * """
12697 * Imaginary part: deprecated (always 0).
12698 */
12699
12700 /* function exit code */
12701 __pyx_L0:;
12702 __Pyx_XGIVEREF(__pyx_r);
12703 __Pyx_RefNannyFinishContext();
12704 return __pyx_r;
12705 }
12706
12707 /* Python wrapper */
12708 static PyObject *__pyx_pw_8PyClical_23imag(PyObject *__pyx_self, PyObject
12709 *__pyx_v_obj); /*proto*/
12710 static char __pyx_doc_8PyClical_22imag[] = "\n Imaginary part: deprecated (always

```

```

0).\n\n >> imag(clifford("\1+\1+\1,2\"))\n 0.0\n >> imag(clifford("\{1,2\}"))\n 0.0\n";
12704 static PyObject *__pyx_pw_8PyClical_23imag(PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
12705 PyObject *__pyx_r = 0;
12706 __Pyx_RefNannyDeclarations
12707 __Pyx_RefNannySetupContext("imag (wrapper)", 0);
12708 __pyx_r = __pyx_pf_8PyClical_22imag(__pyx_self, ((PyObject *)__pyx_v_obj));
12709
12710 /* function exit code */
12711 __Pyx_RefNannyFinishContext();
12712 return __pyx_r;
12713 }
12714
12715 static PyObject *__pyx_pf_8PyClical_22imag(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj) {
12716 PyObject *__pyx_r = NULL;
12717 __Pyx_RefNannyDeclarations
12718 PyObject *__pyx_t_1 = NULL;
12719 int __pyx_lineno = 0;
12720 const char *__pyx_filename = NULL;
12721 int __pyx_clineno = 0;
12722 __Pyx_RefNannySetupContext("imag", 0);
12723 __Pyx_XDECREF(__pyx_r);
12724 __pyx_t_1 = __pyx_f_8PyClical_imag(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1415, __pyx_L1_error)
12725 __Pyx_GOTREF(__pyx_t_1);
12726 __pyx_r = __pyx_t_1;
12727 __pyx_t_1 = 0;
12728 goto __pyx_L0;
12729
12730 /* function exit code */
12731 __pyx_L1_error:;
12732 __Pyx_XDECREF(__pyx_t_1);
12733 __Pyx_AddTraceback("PyClical.imag", __pyx_clineno, __pyx_lineno, __pyx_filename);
12734 __pyx_r = NULL;
12735 __pyx_L0:;
12736 __Pyx_XGIVEREF(__pyx_r);
12737 __Pyx_RefNannyFinishContext();
12738 return __pyx_r;
12739 }
12740
12741 /* "PyClical.pyx":1426
12742 * return 0.0
12743 *
12744 * cpdef inline pure(obj): # ««««««««
12745 * """
12746 * Pure part
12747 */
12748
12749 static PyObject *__pyx_pw_8PyClical_25pure(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
12750 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_pure(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
12751 PyObject *__pyx_r = NULL;
12752 __Pyx_RefNannyDeclarations
12753 PyObject *__pyx_t_1 = NULL;
12754 PyObject *__pyx_t_2 = NULL;
12755 PyObject *__pyx_t_3 = NULL;
12756 int __pyx_lineno = 0;
12757 const char *__pyx_filename = NULL;
12758 int __pyx_clineno = 0;
12759 __Pyx_RefNannySetupContext("pure", 0);
12760
12761 /* "PyClical.pyx":1435
12762 * {1,2}
12763 * """
12764 * return clifford(obj).pure() # ««««««««
12765 *
12766 * cpdef inline even(obj):
12767 */
12768 __Pyx_XDECREF(__pyx_r);
12769 __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1435, __pyx_L1_error)
12770 __Pyx_GOTREF(__pyx_t_2);
12771 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_pure); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1435, __pyx_L1_error)
12772 __Pyx_GOTREF(__pyx_t_3);
12773 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
12774 __pyx_t_2 = NULL;
12775 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
12776 __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
12777 if (likely(__pyx_t_2)) {
12778 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
12779 __Pyx_INCREF(__pyx_t_2);
12780 __Pyx_INCREF(function);
12781 __Pyx_DECREF_SET(__pyx_t_3, function);

```

```

12782 }
12783 }
12784 __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
12785 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
12786 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1435, __pyx_L1_error)
12787 __Pyx_GOTREF(__pyx_t_1);
12788 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
12789 __pyx_r = __pyx_t_1;
12790 __pyx_t_1 = 0;
12791 goto __pyx_L0;
12792
12793 /* "PyClical.pyx":1426
12794 * return 0.0
12795 *
12796 * cpdef inline pure(obj): # ««««««««
12797 * """
12798 * Pure part
12799 */
12800
12801 /* function exit code */
12802 __pyx_L1_error;;
12803 __Pyx_XDECREF(__pyx_t_1);
12804 __Pyx_XDECREF(__pyx_t_2);
12805 __Pyx_XDECREF(__pyx_t_3);
12806 __Pyx_AddTraceback("PyClical.pure", __pyx_clineno, __pyx_lineno, __pyx_filename);
12807 __pyx_r = 0;
12808 __pyx_L0;;
12809 __Pyx_XGIVEREF(__pyx_r);
12810 __Pyx_RefNannyFinishContext();
12811 return __pyx_r;
12812 }
12813
12814 /* Python wrapper */
12815 static PyObject * __pyx_pw_8PyClical_25pure(PyObject * __pyx_self, PyObject
*__pyx_v_obj); /*proto*/
12816 static char __pyx_doc_8PyClical_24pure[] = "\n Pure part\n\n >>
print(pure(clifford(\"1+{1}+{1,2}\")))\n {1}+{1,2}\n >> print(pure(clifford(\"{1,2}\")))\n
{1,2}\n ";
12817 static PyObject * __pyx_pw_8PyClical_25pure(PyObject * __pyx_self, PyObject
*__pyx_v_obj) {
12818 PyObject * __pyx_r = 0;
12819 __Pyx_RefNannyDeclarations
12820 __Pyx_RefNannySetupContext("pure (wrapper)", 0);
12821 __pyx_r = __pyx_pf_8PyClical_24pure(__pyx_self, ((PyObject *) __pyx_v_obj));
12822
12823 /* function exit code */
12824 __Pyx_RefNannyFinishContext();
12825 return __pyx_r;
12826 }
12827
12828 static PyObject * __pyx_pf_8PyClical_24pure(CYTHON_UNUSED PyObject * __pyx_self,
PyObject * __pyx_v_obj) {
12829 PyObject * __pyx_r = NULL;
12830 __Pyx_RefNannyDeclarations
12831 PyObject * __pyx_t_1 = NULL;
12832 int __pyx_lineno = 0;
12833 const char * __pyx_filename = NULL;
12834 int __pyx_clineno = 0;
12835 __Pyx_RefNannySetupContext("pure", 0);
12836 __Pyx_XDECREF(__pyx_r);
12837 __pyx_t_1 = __pyx_f_8PyClical_pure(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1426, __pyx_L1_error)
12838 __Pyx_GOTREF(__pyx_t_1);
12839 __pyx_r = __pyx_t_1;
12840 __pyx_t_1 = 0;
12841 goto __pyx_L0;
12842
12843 /* function exit code */
12844 __pyx_L1_error;;
12845 __Pyx_XDECREF(__pyx_t_1);
12846 __Pyx_AddTraceback("PyClical.pure", __pyx_clineno, __pyx_lineno, __pyx_filename);
12847 __pyx_r = NULL;
12848 __pyx_L0;;
12849 __Pyx_XGIVEREF(__pyx_r);
12850 __Pyx_RefNannyFinishContext();
12851 return __pyx_r;
12852 }
12853
12854 /* "PyClical.pyx":1437
12855 * return clifford(obj).pure()
12856 *
12857 * cpdef inline even(obj): # ««««««««
12858 * """
12859 * Even part of multivector, sum of even grade terms.
12860 */
12861

```

```

12862 static PyObject *__pyx_pw_8PyClical_27even(PyObject *__pyx_self, PyObject
12863 *__pyx_v_obj); /*proto*/
12863 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_even(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
12864 PyObject *__pyx_r = NULL;
12865 __Pyx_RefNannyDeclarations
12866 PyObject *__pyx_t_1 = NULL;
12867 PyObject *__pyx_t_2 = NULL;
12868 PyObject *__pyx_t_3 = NULL;
12869 int __pyx_lineno = 0;
12870 const char *__pyx_filename = NULL;
12871 int __pyx_clineno = 0;
12872 __Pyx_RefNannySetupContext("even", 0);
12873
12874 /* "PyClical.pyx":1444
12875 * 1+{1,2}
12876 * """
12877 * return clifford(obj).even() # ««««««««
12878 *
12879 * cpdef inline odd(obj):
12880 */
12881 __Pyx_XDECREF(__pyx_r);
12882 __pyx_t_2 = __Pyx_PyObject_CallOneArg((PyObject *)__pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1444, __pyx_L1_error)
12883 __Pyx_GOTREF(__pyx_t_2);
12884 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_even); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1444, __pyx_L1_error)
12885 __Pyx_GOTREF(__pyx_t_3);
12886 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
12887 __pyx_t_2 = NULL;
12888 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
12889 __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
12890 if (likely(__pyx_t_2)) {
12891 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
12892 __Pyx_INCREF(__pyx_t_2);
12893 __Pyx_INCREF(function);
12894 __Pyx_DECREF_SET(__pyx_t_3, function);
12895 }
12896 }
12897 __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
12898 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
12899 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1444, __pyx_L1_error)
12900 __Pyx_GOTREF(__pyx_t_1);
12901 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
12902 __pyx_r = __pyx_t_1;
12903 __pyx_t_1 = 0;
12904 goto __pyx_L0;
12905
12906 /* "PyClical.pyx":1437
12907 * return clifford(obj).pure()
12908 *
12909 * cpdef inline even(obj): # ««««««««
12910 * """
12911 * Even part of multivector, sum of even grade terms.
12912 */
12913
12914 /* function exit code */
12915 __pyx_L1_error:;
12916 __Pyx_XDECREF(__pyx_t_1);
12917 __Pyx_XDECREF(__pyx_t_2);
12918 __Pyx_XDECREF(__pyx_t_3);
12919 __Pyx_AddTraceback("PyClical.even", __pyx_clineno, __pyx_lineno, __pyx_filename);
12920 __pyx_r = 0;
12921 __pyx_L0:;
12922 __Pyx_XGIVEREF(__pyx_r);
12923 __Pyx_RefNannyFinishContext();
12924 return __pyx_r;
12925 }
12926
12927 /* Python wrapper */
12928 static PyObject *__pyx_pw_8PyClical_27even(PyObject *__pyx_self, PyObject
12929 *__pyx_v_obj); /*proto*/
12929 static char __pyx_doc_8PyClical_26even[] = "\n Even part of multivector, sum of
even grade terms.\n\n >> print(even(clifford(\"1+{1}+{1,2}\")))\n 1+{1,2}\n ";
12930 static PyObject *__pyx_pw_8PyClical_27even(PyObject *__pyx_self, PyObject
__pyx_v_obj) {
12931 PyObject *__pyx_r = 0;
12932 __Pyx_RefNannyDeclarations
12933 __Pyx_RefNannySetupContext("even (wrapper)", 0);
12934 __pyx_r = __pyx_f_8PyClical_26even(__pyx_self, ((PyObject *)__pyx_v_obj));
12935
12936 /* function exit code */
12937 __Pyx_RefNannyFinishContext();
12938 return __pyx_r;
12939 }
12940

```

```

12941 static PyObject *__pyx_pf_8PyClical_26even(CYTHON_UNUSED PyObject *__pyx_self,
12942 PyObject *__pyx_v_obj) {
12943 PyObject *__pyx_r = NULL;
12944 __Pyx_RefNannyDeclarations
12945 PyObject *__pyx_t_1 = NULL;
12946 int __pyx_lineno = 0;
12947 const char *__pyx_filename = NULL;
12948 int __pyx_clineno = 0;
12949 __Pyx_RefNannySetupContext("even", 0);
12950 __Pyx_XDECREF(__pyx_r);
12951 __pyx_t_1 = __pyx_f_8PyClical_even(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
12952 __PYX_ERR(0, 1437, __pyx_L1_error)
12953 __Pyx_GOTREF(__pyx_t_1);
12954 __pyx_r = __pyx_t_1;
12955 __pyx_t_1 = 0;
12956 goto __pyx_L0;
12957
12958 /* function exit code */
12959 __Pyx_XDECREF(__pyx_t_1);
12960 __Pyx_AddTraceback("PyClical.even", __pyx_clineno, __pyx_lineno, __pyx_filename);
12961 __pyx_r = NULL;
12962 __pyx_L0:;
12963 __Pyx_XGIVEREF(__pyx_r);
12964 __Pyx_RefNannyFinishContext();
12965 return __pyx_r;
12966 }
12967
12968 /* "PyClical.pyx":1446
12969 * return clifford(obj).even()
12970 *
12971 * cpdef inline odd(obj):
12972 * """
12973 * Odd part of multivector, sum of odd grade terms.
12974 */
12975 static PyObject *__pyx_pw_8PyClical_29odd(PyObject *__pyx_self, PyObject
12976 *__pyx_v_obj); /*proto*/
12977 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_odd(PyObject *__pyx_v_obj,
12978 CYTHON_UNUSED int __pyx_skip_dispatch) {
12979 PyObject *__pyx_r = NULL;
12980 __Pyx_RefNannyDeclarations
12981 PyObject *__pyx_t_1 = NULL;
12982 PyObject *__pyx_t_2 = NULL;
12983 PyObject *__pyx_t_3 = NULL;
12984 int __pyx_lineno = 0;
12985 const char *__pyx_filename = NULL;
12986 int __pyx_clineno = 0;
12987 __Pyx_RefNannySetupContext("odd", 0);
12988
12989 /* "PyClical.pyx":1453
12990 * {1}
12991 * """
12992 * return clifford(obj).odd()
12993 * # ««««««««
12994 * cpdef inline involute(obj):
12995 */
12996 __Pyx_XDECREF(__pyx_r);
12997 __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford),
12998 __pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1453, __pyx_L1_error)
12999 __Pyx_GOTREF(__pyx_t_2);
13000 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_odd); if
13001 (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1453, __pyx_L1_error)
13002 __Pyx_GOTREF(__pyx_t_3);
13003 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
13004 __pyx_t_2 = NULL;
13005 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
13006 __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
13007 if (likely(__pyx_t_2)) {
13008 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
13009 __Pyx_INCREF(__pyx_t_2);
13010 __Pyx_INCREF(function);
13011 __Pyx_DECREF_SET(__pyx_t_3, function);
13012 }
13013 }
13014 __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) :
13015 __Pyx_PyObject_CallNoArg(__pyx_t_3);
13016 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
13017 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1453, __pyx_L1_error)
13018 __Pyx_GOTREF(__pyx_t_1);
13019 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
13020 __pyx_r = __pyx_t_1;
13021 __pyx_t_1 = 0;
13022 goto __pyx_L0;
13023
13024 /* "PyClical.pyx":1446
13025 * return clifford(obj).even()

```

```

13021 *
13022 * cpdef inline odd(obj): # ««««««««
13023 * """
13024 * Odd part of multivector, sum of odd grade terms.
13025 */
13026
13027 /* function exit code */
13028 __pyx_L1_error;
13029 __Pyx_XDECREF(__pyx_t_1);
13030 __Pyx_XDECREF(__pyx_t_2);
13031 __Pyx_XDECREF(__pyx_t_3);
13032 __Pyx_AddTraceback("PyClical.odd", __pyx_clineno, __pyx_lineno, __pyx_filename);
13033 __pyx_r = 0;
13034 __pyx_L0;
13035 __Pyx_XGIVEREF(__pyx_r);
13036 __Pyx_RefNannyFinishContext();
13037 return __pyx_r;
13038 }
13039
13040 /* Python wrapper */
13041 static PyObject *__pyx_pw_8PyClical_29odd(PyObject *__pyx_self, PyObject
__pyx_v_obj); /*proto*/
13042 static char __pyx_doc_8PyClical_28odd[] = "\n Odd part of multivector, sum of odd
grade terms.\n\n >> print(odd(clifford(\"1+{1}+{1,2}\"))\n {1}\n ";
13043 static PyObject *__pyx_pw_8PyClical_29odd(PyObject *__pyx_self, PyObject *__pyx_v_obj)
{
13044 PyObject *__pyx_r = 0;
13045 __Pyx_RefNannyDeclarations
13046 __Pyx_RefNannySetupContext("odd (wrapper)", 0);
13047 __pyx_r = __pyx_pf_8PyClical_28odd(__pyx_self, ((PyObject *)__pyx_v_obj));
13048
13049 /* function exit code */
13050 __Pyx_RefNannyFinishContext();
13051 return __pyx_r;
13052 }
13053
13054 static PyObject *__pyx_pf_8PyClical_28odd(CYTHON_UNUSED PyObject *__pyx_self, PyObject
__pyx_v_obj) {
13055 PyObject *__pyx_r = NULL;
13056 __Pyx_RefNannyDeclarations
13057 PyObject *__pyx_t_1 = NULL;
13058 int __pyx_lineno = 0;
13059 const char *__pyx_filename = NULL;
13060 int __pyx_clineno = 0;
13061 __Pyx_RefNannySetupContext("odd", 0);
13062 __Pyx_XDECREF(__pyx_r);
13063 __pyx_t_1 = __pyx_f_8PyClical_odd(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1446, __pyx_L1_error)
13064 __Pyx_GOTREF(__pyx_t_1);
13065 __pyx_r = __pyx_t_1;
13066 __pyx_t_1 = 0;
13067 goto __pyx_L0;
13068
13069 /* function exit code */
13070 __pyx_L1_error;
13071 __Pyx_XDECREF(__pyx_t_1);
13072 __Pyx_AddTraceback("PyClical.odd", __pyx_clineno, __pyx_lineno, __pyx_filename);
13073 __pyx_r = NULL;
13074 __pyx_L0;
13075 __Pyx_XGIVEREF(__pyx_r);
13076 __Pyx_RefNannyFinishContext();
13077 return __pyx_r;
13078 }
13079
13080 /* "PyClical.pyx":1455
13081 * return clifford(obj).odd()
13082 *
13083 * cpdef inline involute(obj): # ««««««««
13084 * """
13085 * Main involution, each {i} is replaced by -{i} in each term, eg. {1}*{2} -> (-{2})*(-{1})
13086 */
13087
13088 static PyObject *__pyx_pw_8PyClical_31involute(PyObject *__pyx_self, PyObject
__pyx_v_obj); /*proto*/
13089 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_involute(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
13090 PyObject *__pyx_r = NULL;
13091 __Pyx_RefNannyDeclarations
13092 PyObject *__pyx_t_1 = NULL;
13093 PyObject *__pyx_t_2 = NULL;
13094 PyObject *__pyx_t_3 = NULL;
13095 int __pyx_lineno = 0;
13096 const char *__pyx_filename = NULL;
13097 int __pyx_clineno = 0;
13098 __Pyx_RefNannySetupContext("involute", 0);
13099
13100 /* "PyClical.pyx":1468

```

```

13101 * 1-{{1}}+{{1,2}}
13102 * """
13103 * return clifford(obj).involute() # ««««««««
13104 *
13105 * cpdef inline reverse(obj):
13106 */
13107 __Pyx_XDECREF(__pyx_r);
13108 __pyx_t_2 = __Pyx_PyObject_CallOneArg((PyObject *)__pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1468, __pyx_L1_error)
13109 __Pyx_GOTREF(__pyx_t_2);
13110 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_involute); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1468, __pyx_L1_error)
13111 __Pyx_GOTREF(__pyx_t_3);
13112 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
13113 __pyx_t_2 = NULL;
13114 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
13115 __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
13116 if (likely(__pyx_t_2)) {
13117 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
13118 __Pyx_INCREF(__pyx_t_2);
13119 __Pyx_INCREF(function);
13120 __Pyx_DECREF_SET(__pyx_t_3, function);
13121 }
13122 }
13123 __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
13124 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
13125 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1468, __pyx_L1_error)
13126 __Pyx_GOTREF(__pyx_t_1);
13127 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
13128 __pyx_r = __pyx_t_1;
13129 __pyx_t_1 = 0;
13130 goto __pyx_L0;
13131
13132 /* "PyClical.pyx":1455
13133 * return clifford(obj).odd()
13134 *
13135 * cpdef inline involute(obj): # ««««««««
13136 * """
13137 * Main involution, each {i} is replaced by -{i} in each term, eg. {1}*{2} -> (-{2})*(-{1})
13138 */
13139
13140 /* function exit code */
13141 __pyx_L1_error;
13142 __Pyx_XDECREF(__pyx_t_1);
13143 __Pyx_XDECREF(__pyx_t_2);
13144 __Pyx_XDECREF(__pyx_t_3);
13145 __Pyx_AddTraceback("PyClical.involute", __pyx_clineno, __pyx_lineno,
__pyx_filename);
13146 __pyx_r = 0;
13147 __pyx_L0;
13148 __Pyx_XGIVEREF(__pyx_r);
13149 __Pyx_RefNannyFinishContext();
13150 return __pyx_r;
13151 }
13152
13153 /* Python wrapper */
13154 static PyObject * __pyx_pw_8PyClical_31involute(PyObject * __pyx_self, PyObject
__pyx_v_obj); /*proto*/
13155 static char __pyx_doc_8PyClical_30involute[] = "\n Main involution, each {i} is
replaced by -{i} in each term, eg. {1}*{2} -> (-{2})*(-{1})\n\n >>
print(involute(clifford(\"{1}\")))\n -{1}\n >> print(involute(clifford(\"{2}\"))
* clifford(\"{1}\"))\n -{1,2}\n >> print(involute(clifford(\"{1}\") * clifford(\"{2}\"))
\n {1,2}\n >> print(involute(clifford(\"1+{1}+{1,2}\")))\n 1-{{1}}+{{1,2}}\n ";
13156 static PyObject * __pyx_pf_8PyClical_31involute(PyObject * __pyx_self, PyObject
__pyx_v_obj) {
13157 PyObject * __pyx_r = 0;
13158 __Pyx_RefNannyDeclarations
13159 __Pyx_RefNannySetupContext("involute (wrapper)", 0);
13160 __pyx_r = __pyx_pf_8PyClical_30involute(__pyx_self, ((PyObject *)__pyx_v_obj));
13161
13162 /* function exit code */
13163 __Pyx_RefNannyFinishContext();
13164 return __pyx_r;
13165 }
13166
13167 static PyObject * __pyx_pf_8PyClical_30involute(CYTHON_UNUSED PyObject * __pyx_self,
PyObject * __pyx_v_obj) {
13168 PyObject * __pyx_r = NULL;
13169 __Pyx_RefNannyDeclarations
13170 PyObject * __pyx_t_1 = NULL;
13171 int __pyx_lineno = 0;
13172 const char * __pyx_filename = NULL;
13173 int __pyx_clineno = 0;
13174 __Pyx_RefNannySetupContext("involute", 0);
13175 __Pyx_XDECREF(__pyx_r);
13176 __pyx_t_1 = __pyx_f_8PyClical_involute(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))

```

```

__PYX_ERR(0, 1455, __pyx_l1_error)
13177 __Pyx_GOTREF(__pyx_t_1);
13178 __pyx_r = __pyx_t_1;
13179 __pyx_t_1 = 0;
13180 goto __pyx_L0;
13181
13182 /* function exit code */
13183 __pyx_l1_error;
13184 __Pyx_XDECREF(__pyx_t_1);
13185 __Pyx_AddTraceback("PyClical.involute", __pyx_clineno, __pyx_lineno,
__pyx_filename);
13186 __pyx_r = NULL;
13187 __pyx_L0;
13188 __Pyx_XGIVEREF(__pyx_r);
13189 __Pyx_RefNannyFinishContext();
13190 return __pyx_r;
13191 }
13192
13193 /* "PyClical.pyx":1470
13194 * return clifford(obj).involute()
13195 *
13196 * cpdef inline reverse(obj): # ««««««««
13197 * """
13198 * Reversion, eg. {1}*{2} -> {2}*{1}
13199 */
13200
13201 static PyObject * __pyx_pw_8PyClical_33reverse(PyObject * __pyx_self, PyObject
* __pyx_v_obj); /*proto*/
13202 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_reverse(PyObject * __pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
13203 PyObject * __pyx_r = NULL;
13204 __Pyx_RefNannyDeclarations
13205 PyObject * __pyx_t_1 = NULL;
13206 PyObject * __pyx_t_2 = NULL;
13207 PyObject * __pyx_t_3 = NULL;
13208 int __pyx_lineno = 0;
13209 const char * __pyx_filename = NULL;
13210 int __pyx_clineno = 0;
13211 __Pyx_RefNannySetupContext("reverse", 0);
13212
13213 /* "PyClical.pyx":1483
13214 * 1+{1}-{1,2}
13215 * """
13216 * return clifford(obj).reverse() # ««««««««
13217 *
13218 * cpdef inline conj(obj):
13219 */
13220 __Pyx_XDECREF(__pyx_r);
13221 __pyx_t_2 = __Pyx_PyObject_CallOneArg((PyObject *) __pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1483, __pyx_l1_error)
13222 __Pyx_GOTREF(__pyx_t_2);
13223 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_reverse); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1483, __pyx_l1_error)
13224 __Pyx_GOTREF(__pyx_t_3);
13225 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
13226 __pyx_t_2 = NULL;
13227 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
13228 __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
13229 if (likely(__pyx_t_2)) {
13230 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
13231 __Pyx_INCREF(__pyx_t_2);
13232 __Pyx_INCREF(function);
13233 __Pyx_DECREF_SET(__pyx_t_3, function);
13234 }
13235 }
13236 __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
13237 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
13238 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1483, __pyx_l1_error)
13239 __Pyx_GOTREF(__pyx_t_1);
13240 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
13241 __pyx_r = __pyx_t_1;
13242 __pyx_t_1 = 0;
13243 goto __pyx_L0;
13244
13245 /* "PyClical.pyx":1470
13246 * return clifford(obj).involute()
13247 *
13248 * cpdef inline reverse(obj): # ««««««««
13249 * """
13250 * Reversion, eg. {1}*{2} -> {2}*{1}
13251 */
13252
13253 /* function exit code */
13254 __pyx_l1_error;
13255 __Pyx_XDECREF(__pyx_t_1);
13256 __Pyx_XDECREF(__pyx_t_2);

```



```

13257 __Pyx_XDECREF(__pyx_t_3);
13258 __Pyx_AddTraceback("PyClical.reverse", __pyx_clineno, __pyx_lineno, __pyx_filename);
13259 __pyx_r = 0;
13260 __pyx_L0:;
13261 __Pyx_XGIVEREF(__pyx_r);
13262 __Pyx_RefNannyFinishContext();
13263 return __pyx_r;
13264 }
13265
13266 /* Python wrapper */
13267 static PyObject *__pyx_pw_8PyClical_33reverse(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
13268 static char __pyx_doc_8PyClical_32reverse[] = "\n Reversion, eg. {1}*{2} ->
{2}*{1}\n\n >> print(reverse(clifford(\"{1}\")))\n {1}\n >> print(reverse(clifford(\"{2}\")) *
clifford(\"{1}\"))\n {1,2}\n >> print(reverse(clifford(\"{1}\")) * clifford(\"{2}\"))\n
-{1,2}\n >> print(reverse(clifford(\"1+{1}+{1,2}\"))\n 1+{1}-{1,2}\n ";
13269 static PyObject *__pyx_pf_8PyClical_33reverse(PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
13270 PyObject *__pyx_r = 0;
13271 __Pyx_RefNannyDeclarations
13272 __Pyx_RefNannySetupContext("reverse (wrapper)", 0);
13273 __pyx_r = __pyx_pf_8PyClical_32reverse(__pyx_self, ((PyObject *)__pyx_v_obj));
13274
13275 /* function exit code */
13276 __Pyx_RefNannyFinishContext();
13277 return __pyx_r;
13278 }
13279
13280 static PyObject *__pyx_pf_8PyClical_32reverse(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj) {
13281 PyObject *__pyx_r = NULL;
13282 __Pyx_RefNannyDeclarations
13283 PyObject *__pyx_t_1 = NULL;
13284 int __pyx_lineno = 0;
13285 const char *__pyx_filename = NULL;
13286 int __pyx_clineno = 0;
13287 __Pyx_RefNannySetupContext("reverse", 0);
13288 __Pyx_XDECREF(__pyx_r);
13289 __pyx_t_1 = __pyx_f_8PyClical_reverse(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1470, __pyx_L1_error)
13290 __Pyx_GOTREF(__pyx_t_1);
13291 __pyx_r = __pyx_t_1;
13292 __pyx_t_1 = 0;
13293 goto __pyx_L0;
13294
13295 /* function exit code */
13296 __pyx_L1_error:;
13297 __Pyx_XDECREF(__pyx_t_1);
13298 __Pyx_AddTraceback("PyClical.reverse", __pyx_clineno, __pyx_lineno, __pyx_filename);
13299 __pyx_r = NULL;
13300 __pyx_L0:;
13301 __Pyx_XGIVEREF(__pyx_r);
13302 __Pyx_RefNannyFinishContext();
13303 return __pyx_r;
13304 }
13305
13306 /* "PyClical.pyx":1485
13307 * return clifford(obj).reverse()
13308 *
13309 * cpdef inline conj(obj): # ««««««««
13310 * """
13311 * Conjugation, reverse o involute == involute o reverse.
13312 */
13313
13314 static PyObject *__pyx_pw_8PyClical_35conj(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
13315 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_conj(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
13316 PyObject *__pyx_r = NULL;
13317 __Pyx_RefNannyDeclarations
13318 PyObject *__pyx_t_1 = NULL;
13319 PyObject *__pyx_t_2 = NULL;
13320 PyObject *__pyx_t_3 = NULL;
13321 int __pyx_lineno = 0;
13322 const char *__pyx_filename = NULL;
13323 int __pyx_clineno = 0;
13324 __Pyx_RefNannySetupContext("conj", 0);
13325
13326 /* "PyClical.pyx":1498
13327 * 1-{1}-{1,2}
13328 * """
13329 * return clifford(obj).conj() # ««««««««
13330 *
13331 * cpdef inline quad(obj):
13332 */
13333 __Pyx_XDECREF(__pyx_r);
13334 __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford),

```

```

__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1498, __pyx_L1_error)
13335 __Pyx_GOTREF(__pyx_t_2);
13336 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_conj); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1498, __pyx_L1_error)
13337 __Pyx_GOTREF(__pyx_t_3);
13338 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
13339 __pyx_t_2 = NULL;
13340 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
13341 __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
13342 if (likely(__pyx_t_2)) {
13343 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
13344 __Pyx_INCREF(__pyx_t_2);
13345 __Pyx_INCREF(function);
13346 __Pyx_DECREF_SET(__pyx_t_3, function);
13347 }
13348 }
13349 __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
13350 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
13351 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1498, __pyx_L1_error)
13352 __Pyx_GOTREF(__pyx_t_1);
13353 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
13354 __pyx_r = __pyx_t_1;
13355 __pyx_t_1 = 0;
13356 goto __pyx_L0;
13357
13358 /* "PyClical.pyx":1485
13359 * return clifford(obj).reverse()
13360 *
13361 * cpdef inline conj(obj): # ««««««««
13362 * """
13363 * Conjugation, reverse o involute == involute o reverse.
13364 */
13365
13366 /* function exit code */
13367 __pyx_L1_error;
13368 __Pyx_XDECREF(__pyx_t_1);
13369 __Pyx_XDECREF(__pyx_t_2);
13370 __Pyx_XDECREF(__pyx_t_3);
13371 __Pyx_AddTraceback("PyClical.conj", __pyx_clineno, __pyx_lineno, __pyx_filename);
13372 __pyx_r = 0;
13373 __pyx_L0;
13374 __Pyx_XGIVEREF(__pyx_r);
13375 __Pyx_RefNannyFinishContext();
13376 return __pyx_r;
13377 }
13378
13379 /* Python wrapper */
13380 static PyObject * __pyx_pw_8PyClical_35conj(PyObject * __pyx_self, PyObject
*__pyx_v_obj); /*proto*/
13381 static char __pyx_doc_8PyClical_34conj[] = "\n Conjugation, reverse o involute ==
involute o reverse.\n\n >> print(conj(clifford(\"{1}\")))\n -{1}\n >>
print(conj(clifford(\"{2}\")) * clifford(\"{1}\"))\n {1,2}\n >> print(conj(clifford(\"{1}\") *
clifford(\"{2}\")))\n -{1,2}\n >> print(conj(clifford(\"1+{1}+{1,2}\")))\n 1-{1}-{1,2}\n
";
13382 static PyObject * __pyx_pw_8PyClical_35conj(PyObject * __pyx_self, PyObject
*__pyx_v_obj) {
13383 PyObject * __pyx_r = 0;
13384 __Pyx_RefNannyDeclarations
13385 __Pyx_RefNannySetupContext("conj (wrapper)", 0);
13386 __pyx_r = __pyx_pf_8PyClical_34conj(__pyx_self, ((PyObject *) __pyx_v_obj));
13387
13388 /* function exit code */
13389 __Pyx_RefNannyFinishContext();
13390 return __pyx_r;
13391 }
13392
13393 static PyObject * __pyx_pf_8PyClical_34conj(CYTHON_UNUSED PyObject * __pyx_self,
PyObject * __pyx_v_obj) {
13394 PyObject * __pyx_r = NULL;
13395 __Pyx_RefNannyDeclarations
13396 PyObject * __pyx_t_1 = NULL;
13397 int __pyx_lineno = 0;
13398 const char * __pyx_filename = NULL;
13399 int __pyx_clineno = 0;
13400 __Pyx_RefNannySetupContext("conj", 0);
13401 __Pyx_XDECREF(__pyx_r);
13402 __pyx_t_1 = __pyx_pf_8PyClical_conj(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1485, __pyx_L1_error)
13403 __Pyx_GOTREF(__pyx_t_1);
13404 __pyx_r = __pyx_t_1;
13405 __pyx_t_1 = 0;
13406 goto __pyx_L0;
13407
13408 /* function exit code */
13409 __pyx_L1_error;
13410 __Pyx_XDECREF(__pyx_t_1);

```

```

13411 __Pyx_AddTraceback("PyClical.conj", __pyx_clineno, __pyx_lineno, __pyx_filename);
13412 __pyx_r = NULL;
13413 __pyx_L0;
13414 __Pyx_XGIVEREF(__pyx_r);
13415 __Pyx_RefNannyFinishContext();
13416 return __pyx_r;
13417 }
13418
13419 /* "PyClical.pyx":1500
13420 * return clifford(obj).conj()
13421 *
13422 * cpdef inline quad(obj): # ««««««««
13423 * """
13424 * Quadratic form == (rev(x)*x)(0).
13425 */
13426
13427 static PyObject *__pyx_pw_8PyClical_37quad(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
13428 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_quad(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
13429 PyObject *__pyx_r = NULL;
13430 __Pyx_RefNannyDeclarations
13431 PyObject *__pyx_t_1 = NULL;
13432 PyObject *__pyx_t_2 = NULL;
13433 PyObject *__pyx_t_3 = NULL;
13434 int __pyx_lineno = 0;
13435 const char *__pyx_filename = NULL;
13436 int __pyx_clineno = 0;
13437 __Pyx_RefNannySetupContext("quad", 0);
13438
13439 /* "PyClical.pyx":1509
13440 * 2.0
13441 * """
13442 * return clifford(obj).quad() # ««««««««
13443 *
13444 * cpdef inline norm(obj):
13445 */
13446 __Pyx_XDECREF(__pyx_r);
13447 __pyx_t_2 = __Pyx_PyObject_CallOneArg((PyObject *)__pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1509, __pyx_L1_error)
13448 __Pyx_GOTREF(__pyx_t_2);
13449 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_quad); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1509, __pyx_L1_error)
13450 __Pyx_GOTREF(__pyx_t_3);
13451 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
13452 __pyx_t_2 = NULL;
13453 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
13454 __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
13455 if (likely(__pyx_t_2)) {
13456 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
13457 __Pyx_INCREF(__pyx_t_2);
13458 __Pyx_INCREF(function);
13459 __Pyx_DECREF_SET(__pyx_t_3, function);
13460 }
13461 }
13462 __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
13463 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
13464 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1509, __pyx_L1_error)
13465 __Pyx_GOTREF(__pyx_t_1);
13466 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
13467 __pyx_r = __pyx_t_1;
13468 __pyx_t_1 = 0;
13469 goto __pyx_L0;
13470
13471 /* "PyClical.pyx":1500
13472 * return clifford(obj).conj()
13473 *
13474 * cpdef inline quad(obj): # ««««««««
13475 * """
13476 * Quadratic form == (rev(x)*x)(0).
13477 */
13478
13479 /* function exit code */
13480 __pyx_L1_error;
13481 __Pyx_XDECREF(__pyx_t_1);
13482 __Pyx_XDECREF(__pyx_t_2);
13483 __Pyx_XDECREF(__pyx_t_3);
13484 __Pyx_AddTraceback("PyClical.quad", __pyx_clineno, __pyx_lineno, __pyx_filename);
13485 __pyx_r = 0;
13486 __pyx_L0;
13487 __Pyx_XGIVEREF(__pyx_r);
13488 __Pyx_RefNannyFinishContext();
13489 return __pyx_r;
13490 }
13491
13492 /* Python wrapper */

```

```

13493 static PyObject *__pyx_pw_8PyClical_37quad(PyObject *__pyx_self, PyObject
13494 *__pyx_v_obj); /*proto*/
13494 static char __pyx_doc_8PyClical_36quad[] = "\n Quadratic form == (rev(x)*x)(0).\n\n
>> print(quad(clifford(\"1+{1}+{1,2}\"))\n 3.0\n >>
print(quad(clifford(\"1+{-1}+{1,2}+{1,2,3}\"))\n 2.0\n ";
13495 static PyObject *__pyx_pw_8PyClical_37quad(PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
13496 PyObject *__pyx_r = 0;
13497 __Pyx_RefNannyDeclarations
13498 __Pyx_RefNannySetupContext("quad (wrapper)", 0);
13499 __pyx_r = __pyx_pf_8PyClical_36quad(__pyx_self, ((PyObject *)__pyx_v_obj));
13500
13501 /* function exit code */
13502 __Pyx_RefNannyFinishContext();
13503 return __pyx_r;
13504 }
13505
13506 static PyObject *__pyx_pf_8PyClical_36quad(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj) {
13507 PyObject *__pyx_r = NULL;
13508 __Pyx_RefNannyDeclarations
13509 PyObject *__pyx_t_1 = NULL;
13510 int __pyx_lineno = 0;
13511 const char *__pyx_filename = NULL;
13512 int __pyx_clineno = 0;
13513 __Pyx_RefNannySetupContext("quad", 0);
13514 __Pyx_XDECREF(__pyx_r);
13515 __pyx_t_1 = __pyx_f_8PyClical_quad(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1500, __pyx_L1_error)
13516 __Pyx_GOTREF(__pyx_t_1);
13517 __pyx_r = __pyx_t_1;
13518 __pyx_t_1 = 0;
13519 goto __pyx_L0;
13520
13521 /* function exit code */
13522 __pyx_L1_error:;
13523 __Pyx_XDECREF(__pyx_t_1);
13524 __Pyx_AddTraceback("PyClical.quad", __pyx_clineno, __pyx_lineno, __pyx_filename);
13525 __pyx_r = NULL;
13526 __pyx_L0:;
13527 __Pyx_XGIVEREF(__pyx_r);
13528 __Pyx_RefNannyFinishContext();
13529 return __pyx_r;
13530 }
13531
13532 /* "PyClical.pyx":1511
13533 * return clifford(obj).quad()
13534 *
13535 * cpdef inline norm(obj):
13536 * """
13537 * norm == sum of squares of coordinates.
13538 */
13539
13540 static PyObject *__pyx_pw_8PyClical_39norm(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
13541 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_norm(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
13542 PyObject *__pyx_r = NULL;
13543 __Pyx_RefNannyDeclarations
13544 PyObject *__pyx_t_1 = NULL;
13545 PyObject *__pyx_t_2 = NULL;
13546 PyObject *__pyx_t_3 = NULL;
13547 int __pyx_lineno = 0;
13548 const char *__pyx_filename = NULL;
13549 int __pyx_clineno = 0;
13550 __Pyx_RefNannySetupContext("norm", 0);
13551
13552 /* "PyClical.pyx":1520
13553 * 4.0
13554 * """
13555 * return clifford(obj).norm()
13556 *
13557 * cpdef inline abs(obj):
13558 */
13559 __Pyx_XDECREF(__pyx_r);
13560 __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford),
__pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1520, __pyx_L1_error)
13561 __Pyx_GOTREF(__pyx_t_2);
13562 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_norm); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1520, __pyx_L1_error)
13563 __Pyx_GOTREF(__pyx_t_3);
13564 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
13565 __pyx_t_2 = NULL;
13566 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
13567 __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
13568 if (likely(__pyx_t_2)) {
13569 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);

```

```

13570 __Pyx_INCREF(__pyx_t_2);
13571 __Pyx_INCREF(function);
13572 __Pyx_DECREF_SET(__pyx_t_3, function);
13573 }
13574 }
13575 __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_t_2) :
__Pyx_PyObject_CallNoArg(__pyx_t_3);
13576 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
13577 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1520, __pyx_L1_error)
13578 __Pyx_GOTREF(__pyx_t_1);
13579 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
13580 __pyx_r = __pyx_t_1;
13581 __pyx_t_1 = 0;
13582 goto __pyx_L0;
13583
13584 /* "PyClical.pyx":1511
13585 * return clifford(obj).quad()
13586 *
13587 * cpdef inline norm(obj): # ««««««««
13588 * """
13589 * norm == sum of squares of coordinates.
13590 */
13591
13592 /* function exit code */
13593 __pyx_L1_error:;
13594 __Pyx_XDECREF(__pyx_t_1);
13595 __Pyx_XDECREF(__pyx_t_2);
13596 __Pyx_XDECREF(__pyx_t_3);
13597 __Pyx_AddTraceback("PyClical.norm", __pyx_clineno, __pyx_lineno, __pyx_filename);
13598 __pyx_r = 0;
13599 __pyx_L0:;
13600 __Pyx_XGIVEREF(__pyx_r);
13601 __Pyx_RefNannyFinishContext();
13602 return __pyx_r;
13603 }
13604
13605 /* Python wrapper */
13606 static PyObject * __pyx_pw_8PyClical_39norm(PyObject * __pyx_self, PyObject
*__pyx_v_obj); /*proto*/
13607 static char __pyx_doc_8PyClical_38norm[] = "\n norm == sum of squares of
coordinates.\n\n >> norm(clifford(\"1+{1}+{1,2}\"))\n 3.0\n >>
norm(clifford(\"1+{-1}+{1,2}+{1,2,3}\"))\n 4.0\n ";
13608 static PyObject * __pyx_pw_8PyClical_39norm(PyObject * __pyx_self, PyObject
*__pyx_v_obj) {
13609 PyObject * __pyx_r = 0;
13610 __Pyx_RefNannyDeclarations
13611 __Pyx_RefNannySetupContext("norm (wrapper)", 0);
13612 __pyx_r = __pyx_pf_8PyClical_38norm(__pyx_self, ((PyObject *) __pyx_v_obj));
13613
13614 /* function exit code */
13615 __Pyx_RefNannyFinishContext();
13616 return __pyx_r;
13617 }
13618
13619 static PyObject * __pyx_pf_8PyClical_38norm(CYTHON_UNUSED PyObject * __pyx_self,
PyObject * __pyx_v_obj) {
13620 PyObject * __pyx_r = NULL;
13621 __Pyx_RefNannyDeclarations
13622 PyObject * __pyx_t_1 = NULL;
13623 int __pyx_lineno = 0;
13624 const char * __pyx_filename = NULL;
13625 int __pyx_clineno = 0;
13626 __Pyx_RefNannySetupContext("norm", 0);
13627 __Pyx_XDECREF(__pyx_r);
13628 __pyx_t_1 = __pyx_f_8PyClical_norm(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1511, __pyx_L1_error)
13629 __Pyx_GOTREF(__pyx_t_1);
13630 __pyx_r = __pyx_t_1;
13631 __pyx_t_1 = 0;
13632 goto __pyx_L0;
13633
13634 /* function exit code */
13635 __pyx_L1_error:;
13636 __Pyx_XDECREF(__pyx_t_1);
13637 __Pyx_AddTraceback("PyClical.norm", __pyx_clineno, __pyx_lineno, __pyx_filename);
13638 __pyx_r = NULL;
13639 __pyx_L0:;
13640 __Pyx_XGIVEREF(__pyx_r);
13641 __Pyx_RefNannyFinishContext();
13642 return __pyx_r;
13643 }
13644
13645 /* "PyClical.pyx":1522
13646 * return clifford(obj).norm()
13647 *
13648 * cpdef inline abs(obj): # ««««««««
13649 * """

```

```

13650 * Absolute value of multivector: multivector 2-norm.
13651 */
13652
13653 static PyObject *__pyx_pw_8PyClical_4labs(PyObject *__pyx_self, PyObject
13654 *__pyx_v_obj); /*proto*/
13655 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_abs(PyObject *__pyx_v_obj,
13656 CYTHON_UNUSED int __pyx_skip_dispatch) {
13657 PyObject *__pyx_r = NULL;
13658 __Pyx_RefNannyDeclarations
13659 PyObject *__pyx_t_1 = NULL;
13660 int __pyx_lineno = 0;
13661 const char *__pyx_filename = NULL;
13662 int __pyx_clineno = 0;
13663 __Pyx_RefNannySetupContext("abs", 0);
13664
13665 /* "PyClical.pyx":1529
13666 * 2.0
13667 * """
13668 * return glucat.abs(toClifford(obj)) # ««««««««
13669 *
13670 * cpdef inline max_abs(obj):
13671 */
13672 __Pyx_XDECREF(__pyx_r);
13673 __pyx_t_1 = PyFloat_FromDouble(abs(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
13674 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1529, __pyx_L1_error)
13675 __Pyx_GOTREF(__pyx_t_1);
13676 __pyx_r = __pyx_t_1;
13677 __pyx_t_1 = 0;
13678 goto __pyx_L0;
13679
13680 /* "PyClical.pyx":1522
13681 * return clifford(obj).norm()
13682 *
13683 * cpdef inline abs(obj): # ««««««««
13684 * """
13685 * Absolute value of multivector: multivector 2-norm.
13686 */
13687
13688 /* function exit code */
13689 __pyx_L1_error:;
13690 __Pyx_XDECREF(__pyx_t_1);
13691 __Pyx_AddTraceback("PyClical.abs", __pyx_clineno, __pyx_lineno, __pyx_filename);
13692 __pyx_r = 0;
13693 __pyx_L0:;
13694 __Pyx_XGIVEREF(__pyx_r);
13695 __Pyx_RefNannyFinishContext();
13696 return __pyx_r;
13697 }
13698
13699 /* Python wrapper */
13700 static PyObject *__pyx_pw_8PyClical_4labs(PyObject *__pyx_self, PyObject
13701 *__pyx_v_obj); /*proto*/
13702 static char __pyx_doc_8PyClical_40abs[] = "\n Absolute value of multivector:
13703 multivector 2-norm.\n >> abs(clifford(\"1+{-1}+{1,2}+{1,2,3}\"))\n 2.0\n ";
13704 static PyObject *__pyx_pw_8PyClical_4labs(PyObject *__pyx_self, PyObject *__pyx_v_obj)
13705 {
13706 PyObject *__pyx_r = 0;
13707 __Pyx_RefNannyDeclarations
13708 PyObject *__pyx_t_1 = NULL;
13709 int __pyx_lineno = 0;
13710 const char *__pyx_filename = NULL;
13711 int __pyx_clineno = 0;
13712 __Pyx_RefNannySetupContext("abs (wrapper)", 0);
13713 __pyx_r = __pyx_f_8PyClical_40abs(__pyx_self, ((PyObject *)__pyx_v_obj));
13714
13715 /* function exit code */
13716 __Pyx_RefNannyFinishContext();
13717 return __pyx_r;
13718 }
13719
13720 static PyObject *__pyx_pf_8PyClical_40abs(CYTHON_UNUSED PyObject *__pyx_self, PyObject
13721 *__pyx_v_obj) {
13722 PyObject *__pyx_r = NULL;
13723 __Pyx_RefNannyDeclarations
13724 PyObject *__pyx_t_1 = NULL;
13725 int __pyx_lineno = 0;
13726 const char *__pyx_filename = NULL;
13727 int __pyx_clineno = 0;
13728 __Pyx_RefNannySetupContext("abs", 0);
13729 __Pyx_XDECREF(__pyx_r);
13730 __pyx_t_1 = __pyx_f_8PyClical_abs(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
13731 __PYX_ERR(0, 1522, __pyx_L1_error)
13732 __Pyx_GOTREF(__pyx_t_1);
13733 __pyx_r = __pyx_t_1;
13734 __pyx_t_1 = 0;
13735 goto __pyx_L0;
13736
13737 /* function exit code */
13738 __pyx_L1_error:;
13739 __Pyx_XDECREF(__pyx_t_1);
13740 __Pyx_AddTraceback("PyClical.abs", __pyx_clineno, __pyx_lineno, __pyx_filename);

```

```

13729 __pyx_r = NULL;
13730 __pyx_L0:;
13731 __Pyx_XGIVEREF(__pyx_r);
13732 __Pyx_RefNannyFinishContext();
13733 return __pyx_r;
13734 }
13735
13736 /* "PyClical.pyx":1531
13737 * return glucat.abs(toClifford(obj))
13738 *
13739 * cpdef inline max_abs(obj):
13740 * """
13741 * Maximum absolute value of coordinates multivector: multivector infinity-norm.
13742 */
13743
13744 static PyObject *__pyx_pw_8PyClical_43max_abs(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
13745 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_max_abs(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
13746 PyObject *__pyx_r = NULL;
13747 __Pyx_RefNannyDeclarations
13748 PyObject *__pyx_t_1 = NULL;
13749 int __pyx_lineno = 0;
13750 const char *__pyx_filename = NULL;
13751 int __pyx_clineno = 0;
13752 __Pyx_RefNannySetupContext("max_abs", 0);
13753
13754 /* "PyClical.pyx":1541
13755 *
13756 * """
13757 * return glucat.max_abs(toClifford(obj))
13758 *
13759 * cpdef inline pow(obj, m):
13760 */
13761 __Pyx_XDECREF(__pyx_r);
13762 __pyx_t_1 = PyFloat_FromDouble(max_abs(__pyx_f_8PyClical_toClifford(__pyx_v_obj)));
13763 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1541, __pyx_L1_error)
13764 __Pyx_GOTREF(__pyx_t_1);
13765 __pyx_r = __pyx_t_1;
13766 __pyx_t_1 = 0;
13767 goto __pyx_L0;
13768
13769 /* "PyClical.pyx":1531
13770 * return glucat.abs(toClifford(obj))
13771 *
13772 * cpdef inline max_abs(obj):
13773 * """
13774 * Maximum absolute value of coordinates multivector: multivector infinity-norm.
13775 */
13776 /* function exit code */
13777 __pyx_L1_error:;
13778 __Pyx_XDECREF(__pyx_t_1);
13779 __Pyx_AddTraceback("PyClical.max_abs", __pyx_clineno, __pyx_lineno, __pyx_filename);
13780 __pyx_r = 0;
13781 __pyx_L0:;
13782 __Pyx_XGIVEREF(__pyx_r);
13783 __Pyx_RefNannyFinishContext();
13784 return __pyx_r;
13785 }
13786
13787 /* Python wrapper */
13788 static PyObject *__pyx_pw_8PyClical_43max_abs(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
13789 static char __pyx_doc_8PyClical_42max_abs[] = "\n Maximum absolute value of
coordinates multivector: multivector infinity-norm.\n >>
max_abs(clifford(\"1+{-1}+{1,2}+{1,2,3}\")\n 1.0\n >> max_abs(clifford(\"3+2{1}+{1,2}\")\n
3.0\n\n ";
13790 static PyObject *__pyx_pw_8PyClical_43max_abs(PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
13791 PyObject *__pyx_r = 0;
13792 __Pyx_RefNannyDeclarations
13793 __Pyx_RefNannySetupContext("max_abs (wrapper)", 0);
13794 __pyx_r = __pyx_pf_8PyClical_42max_abs(__pyx_self, ((PyObject *)__pyx_v_obj));
13795
13796 /* function exit code */
13797 __Pyx_RefNannyFinishContext();
13798 return __pyx_r;
13799 }
13800
13801 static PyObject *__pyx_pf_8PyClical_42max_abs(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj) {
13802 PyObject *__pyx_r = NULL;
13803 __Pyx_RefNannyDeclarations
13804 PyObject *__pyx_t_1 = NULL;
13805 int __pyx_lineno = 0;
13806 const char *__pyx_filename = NULL;

```

```

13807 int __pyx_clineno = 0;
13808 __Pyx_RefNannySetupContext("max_abs", 0);
13809 __Pyx_XDECREF(__pyx_r);
13810 __pyx_t_1 = __pyx_f_8PyClical_max_abs(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1531, __pyx_L1_error)
13811 __Pyx_GOTREF(__pyx_t_1);
13812 __pyx_r = __pyx_t_1;
13813 __pyx_t_1 = 0;
13814 goto __pyx_L0;
13815
13816 /* function exit code */
13817 __pyx_L1_error:;
13818 __Pyx_XDECREF(__pyx_t_1);
13819 __Pyx_AddTraceback("PyClical.max_abs", __pyx_clineno, __pyx_lineno, __pyx_filename);
13820 __pyx_r = NULL;
13821 __pyx_L0:;
13822 __Pyx_XGIVEREF(__pyx_r);
13823 __Pyx_RefNannyFinishContext();
13824 return __pyx_r;
13825 }
13826
13827 /* "PyClical.pyx":1543
13828 * return glucat.max_abs(toClifford(obj))
13829 *
13830 * cpdef inline pow(obj, m): # ««««««««
13831 * """
13832 * Integer power of multivector: obj to the m.
13833 */
13834
13835 static PyObject * __pyx_pw_8PyClical_45pow(PyObject * __pyx_self, PyObject * __pyx_args,
PyObject * __pyx_kwds); /*proto*/
13836 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_pow(PyObject * __pyx_v_obj, PyObject
__pyx_v_m, CYTHON_UNUSED int __pyx_skip_dispatch) {
13837 PyObject * __pyx_r = NULL;
13838 __Pyx_RefNannyDeclarations
13839 PyObject * __pyx_t_1 = NULL;
13840 PyObject * __pyx_t_2 = NULL;
13841 PyObject * __pyx_t_3 = NULL;
13842 PyObject * __pyx_t_4 = NULL;
13843 PyObject * __pyx_t_5 = NULL;
13844 PyObject * __pyx_t_6 = NULL;
13845 int __pyx_t_7;
13846 PyObject * __pyx_t_8 = NULL;
13847 PyObject * __pyx_t_9 = NULL;
13848 PyObject * __pyx_t_10 = NULL;
13849 int __pyx_lineno = 0;
13850 const char * __pyx_filename = NULL;
13851 int __pyx_clineno = 0;
13852 __Pyx_RefNannySetupContext("pow", 0);
13853
13854 /* "PyClical.pyx":1562
13855 * 1
13856 * """
13857 * try: # ««««««««
13858 * math.pow(obj, m)
13859 * except:
13860 */
13861 {
13862 __Pyx_PyThreadState_declare
13863 __Pyx_PyThreadState_assign
13864 __Pyx_ExceptionSave(&__pyx_t_1, &__pyx_t_2, &__pyx_t_3);
13865 __Pyx_XGOTREF(__pyx_t_1);
13866 __Pyx_XGOTREF(__pyx_t_2);
13867 __Pyx_XGOTREF(__pyx_t_3);
13868 /*try:*/ {
13869
13870 /* "PyClical.pyx":1563
13871 * """
13872 * try:
13873 * math.pow(obj, m) # ««««««««
13874 * except:
13875 * return clifford(obj).pow(m)
13876 */
13877 __Pyx_GetModuleGlobalName(__pyx_t_5, __pyx_n_s_math); if (unlikely(!__pyx_t_5))
__PYX_ERR(0, 1563, __pyx_L3_error)
13878 __Pyx_GOTREF(__pyx_t_5);
13879 __pyx_t_6 = __Pyx_PyObject_GetAttrStr(__pyx_t_5, __pyx_n_s_pow); if
(unlikely(!__pyx_t_6)) __PYX_ERR(0, 1563, __pyx_L3_error)
13880 __Pyx_GOTREF(__pyx_t_6);
13881 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
13882 __pyx_t_5 = NULL;
13883 __pyx_t_7 = 0;
13884 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_6))) {
13885 __pyx_t_5 = PyMethod_GET_SELF(__pyx_t_6);
13886 if (likely(__pyx_t_5)) {
13887 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_6);
13888 __Pyx_INCREF(__pyx_t_5);

```



```

13889 __Pyx_INCREF(function);
13890 __Pyx_DECREF_SET(__pyx_t_6, function);
13891 __pyx_t_7 = 1;
13892 }
13893 }
13894 #if CYTHON_FAST_PYCALL
13895 if (PyFunction_Check(__pyx_t_6)) {
13896 PyObject *__pyx_temp[3] = {__pyx_t_5, __pyx_v_obj, __pyx_v_m};
13897 __pyx_t_4 = __Pyx_PyFunction_FastCall(__pyx_t_6, __pyx_temp+1-__pyx_t_7,
2+__pyx_t_7); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1563, __pyx_L3_error)
13898 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
13899 __Pyx_GOTREF(__pyx_t_4);
13900 } else
13901 #endif
13902 #if CYTHON_FAST_PYCCALL
13903 if (__Pyx_PyFastCFunction_Check(__pyx_t_6)) {
13904 PyObject *__pyx_temp[3] = {__pyx_t_5, __pyx_v_obj, __pyx_v_m};
13905 __pyx_t_4 = __Pyx_PyCFunction_FastCall(__pyx_t_6, __pyx_temp+1-__pyx_t_7,
2+__pyx_t_7); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1563, __pyx_L3_error)
13906 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
13907 __Pyx_GOTREF(__pyx_t_4);
13908 } else
13909 #endif
13910 {
13911 __pyx_t_8 = PyTuple_New(2+__pyx_t_7); if (unlikely(!__pyx_t_8)) __PYX_ERR(0,
1563, __pyx_L3_error)
13912 __Pyx_GOTREF(__pyx_t_8);
13913 if (__pyx_t_5) {
13914 __Pyx_GIVEREF(__pyx_t_5); PyTuple_SET_ITEM(__pyx_t_8, 0, __pyx_t_5);
__pyx_t_5 = NULL;
13915 }
13916 __Pyx_INCREF(__pyx_v_obj);
13917 __Pyx_GIVEREF(__pyx_v_obj);
13918 PyTuple_SET_ITEM(__pyx_t_8, 0+__pyx_t_7, __pyx_v_obj);
13919 __Pyx_INCREF(__pyx_v_m);
13920 __Pyx_GIVEREF(__pyx_v_m);
13921 PyTuple_SET_ITEM(__pyx_t_8, 1+__pyx_t_7, __pyx_v_m);
13922 __pyx_t_4 = __Pyx_PyObject_Call(__pyx_t_6, __pyx_t_8, NULL); if
(unlikely(!__pyx_t_4)) __PYX_ERR(0, 1563, __pyx_L3_error)
13923 __Pyx_GOTREF(__pyx_t_4);
13924 __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
13925 }
13926 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
13927 __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
13928
13929 /* "PyClical.pyx":1562
13930 * 1
13931 * """
13932 * try: # ««««««««
13933 * math.pow(obj, m)
13934 * except:
13935 */
13936 }
13937 __Pyx_XDECREF(__pyx_t_1); __pyx_t_1 = 0;
13938 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
13939 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
13940 goto __pyx_L8_try_end;
13941 __pyx_L3_error:;
13942 __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
13943 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
13944 __Pyx_XDECREF(__pyx_t_6); __pyx_t_6 = 0;
13945 __Pyx_XDECREF(__pyx_t_8); __pyx_t_8 = 0;
13946
13947 /* "PyClical.pyx":1564
13948 * try:
13949 * math.pow(obj, m)
13950 * except: # ««««««««
13951 * return clifford(obj).pow(m)
13952 *
13953 */
13954 /*except:*/ {
13955 __Pyx_AddTraceback("PyClical.pow", __pyx_clineno, __pyx_lineno, __pyx_filename);
13956 if (__Pyx_GetException(&__pyx_t_4, &__pyx_t_6, &__pyx_t_8) < 0) __PYX_ERR(0,
1564, __pyx_L5_except_error)
13957 __Pyx_GOTREF(__pyx_t_4);
13958 __Pyx_GOTREF(__pyx_t_6);
13959 __Pyx_GOTREF(__pyx_t_8);
13960
13961 /* "PyClical.pyx":1565
13962 * math.pow(obj, m)
13963 * except:
13964 * return clifford(obj).pow(m) # ««««««««
13965 *
13966 * cpdef inline outer_pow(obj, m):
13967 */
13968 __Pyx_XDECREF(__pyx_r);
13969 __pyx_t_9 = __Pyx_PyObject_CallOneArg((PyObject)

```

```

*)__pyx_ptype_8PyClical_clifford), __pyx_v_obj); if (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1565,
__pyx_L5_except_error)
13970 __Pyx_GOTREF(__pyx_t_9);
13971 __pyx_t_10 = __Pyx_PyObject_GetAttrStr(__pyx_t_9, __pyx_n_s_pow); if
(unlikely(!__pyx_t_10)) __PYX_ERR(0, 1565, __pyx_L5_except_error)
13972 __Pyx_GOTREF(__pyx_t_10);
13973 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
13974 __pyx_t_9 = NULL;
13975 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_10))) {
13976 __pyx_t_9 = PyMethod_GET_SELF(__pyx_t_10);
13977 if (likely(__pyx_t_9)) {
13978 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_10);
13979 __Pyx_INCREF(__pyx_t_9);
13980 __Pyx_INCREF(function);
13981 __Pyx_DECREF_SET(__pyx_t_10, function);
13982 }
13983 }
13984 __pyx_t_5 = (__pyx_t_9) ? __Pyx_PyObject_Call2Args(__pyx_t_10, __pyx_t_9,
__pyx_v_m) : __Pyx_PyObject_CallOneArg(__pyx_t_10, __pyx_v_m);
13985 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
13986 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1565, __pyx_L5_except_error)
13987 __Pyx_GOTREF(__pyx_t_5);
13988 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
13989 __pyx_r = __pyx_t_5;
13990 __pyx_t_5 = 0;
13991 __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
13992 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
13993 __Pyx_DECREF(__pyx_t_8); __pyx_t_8 = 0;
13994 goto __pyx_L6_except_return;
13995 }
13996 __pyx_L5_except_error:;
13997
13998 /* "PyClical.pyx":1562
13999 * 1
14000 * """
14001 * try: # ««««««««
14002 * math.pow(obj, m)
14003 * except:
14004 */
14005 __Pyx_XGIVEREF(__pyx_t_1);
14006 __Pyx_XGIVEREF(__pyx_t_2);
14007 __Pyx_XGIVEREF(__pyx_t_3);
14008 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
14009 goto __pyx_L1_error;
14010 __pyx_L6_except_return:;
14011 __Pyx_XGIVEREF(__pyx_t_1);
14012 __Pyx_XGIVEREF(__pyx_t_2);
14013 __Pyx_XGIVEREF(__pyx_t_3);
14014 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
14015 goto __pyx_L0;
14016 __pyx_L8_try_end:;
14017 }
14018
14019 /* "PyClical.pyx":1543
14020 * return glucat.max_abs(toClifford(obj))
14021 *
14022 * cpdef inline pow(obj, m): # ««««««««
14023 * """
14024 * Integer power of multivector: obj to the m.
14025 */
14026
14027 /* function exit code */
14028 __pyx_r = Py_None; __Pyx_INCREF(Py_None);
14029 goto __pyx_L0;
14030 __pyx_L1_error:;
14031 __Pyx_XDECREF(__pyx_t_4);
14032 __Pyx_XDECREF(__pyx_t_5);
14033 __Pyx_XDECREF(__pyx_t_6);
14034 __Pyx_XDECREF(__pyx_t_8);
14035 __Pyx_XDECREF(__pyx_t_9);
14036 __Pyx_XDECREF(__pyx_t_10);
14037 __Pyx_AddTraceback("PyClical.pow", __pyx_clineno, __pyx_lineno, __pyx_filename);
14038 __pyx_r = 0;
14039 __pyx_L0:;
14040 __Pyx_XGIVEREF(__pyx_r);
14041 __Pyx_RefNannyFinishContext();
14042 return __pyx_r;
14043 }
14044
14045 /* Python wrapper */
14046 static PyObject * __pyx_pw_8PyClical_45pow(PyObject * __pyx_self, PyObject * __pyx_args,
PyObject * __pyx_kwds); /*proto*/
14047 static char __pyx_doc_8PyClical_44pow[] = "\n Integer power of multivector: obj to
the m.\n\n >> x=clifford(\"{1}\"); print(pow(x,2))\n 1\n >> x=clifford(\"2\");
print(pow(x,2))\n 4\n >> x=clifford(\"2+{1}\"); print(pow(x,0))\n 1\n >>
x=clifford(\"2+{1}\"); print(pow(x,1))\n 2+{1}\n >> x=clifford(\"2+{1}\"); print(pow(x,2))\n
5+4{1}\n >> print(pow(clifford(\"1+{1}+{1,2}\"),3))\n 1+3{1}+3{1,2}\n >>";

```

```

i=clifford("{1,2}"); print(exp(pi/2) * pow(i, i))\n 1\n ";
14048 static PyObject *__pyx_pw_8PyClical_45pow(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds) {
14049 PyObject *__pyx_v_obj = 0;
14050 PyObject *__pyx_v_m = 0;
14051 int __pyx_lineno = 0;
14052 const char *__pyx_filename = NULL;
14053 int __pyx_clineno = 0;
14054 PyObject *__pyx_r = 0;
14055 __Pyx_RefNannyDeclarations
14056 __Pyx_RefNannySetupContext("pow (wrapper)", 0);
14057 {
14058 static PyObject *__pyx_pyargnames[] = {&__pyx_n_s_obj, &__pyx_n_s_m, 0};
14059 PyObject* values[2] = {0, 0};
14060 if (unlikely(__pyx_kwds)) {
14061 Py_ssize_t kw_args;
14062 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
14063 switch (pos_args) {
14064 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
14065 CYTHON_FALLTHROUGH;
14066 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
14067 CYTHON_FALLTHROUGH;
14068 case 0: break;
14069 default: goto __pyx_L5_argtuple_error;
14070 }
14071 kw_args = PyDict_Size(__pyx_kwds);
14072 switch (pos_args) {
14073 case 0:
14074 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
0)) kw_args--;
14075 else goto __pyx_L5_argtuple_error;
14076 CYTHON_FALLTHROUGH;
14077 case 1:
14078 if (likely((values[1] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_m)) !=
0)) kw_args--;
14079 else {
14080 __Pyx_RaiseArgtupleInvalid("pow", 1, 2, 2, 1); __PYX_ERR(0, 1543,
__pyx_L3_error)
14081 }
14082 if (unlikely(kw_args > 0)) {
14083 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "pow") < 0)) __PYX_ERR(0, 1543, __pyx_L3_error)
14084 }
14085 } else if (PyTuple_GET_SIZE(__pyx_args) != 2) {
14086 goto __pyx_L5_argtuple_error;
14087 } else {
14088 values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
14089 values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
14090 }
14091 __pyx_v_obj = values[0];
14092 __pyx_v_m = values[1];
14093 }
14094 goto __pyx_L4_argument_unpacking_done;
14095 __pyx_L5_argtuple_error:;
14096 __Pyx_RaiseArgtupleInvalid("pow", 1, 2, 2, PyTuple_GET_SIZE(__pyx_args));
14097 __PYX_ERR(0, 1543, __pyx_L3_error)
14098 __pyx_L3_error:;
14099 __Pyx_AddTraceback("PyClical.pow", __pyx_clineno, __pyx_lineno, __pyx_filename);
14100 __Pyx_RefNannyFinishContext();
14101 return NULL;
14102 __pyx_L4_argument_unpacking_done:;
14103 __pyx_r = __pyx_pf_8PyClical_44pow(__pyx_self, __pyx_v_obj, __pyx_v_m);
14104
14105 /* function exit code */
14106 __Pyx_RefNannyFinishContext();
14107 return __pyx_r;
14108 }
14109
14110 static PyObject *__pyx_pf_8PyClical_44pow(CYTHON_UNUSED PyObject *__pyx_self, PyObject
*__pyx_v_obj, PyObject *__pyx_v_m) {
14111 PyObject *__pyx_r = NULL;
14112 __Pyx_RefNannyDeclarations
14113 PyObject *__pyx_t_1 = NULL;
14114 int __pyx_lineno = 0;
14115 const char *__pyx_filename = NULL;
14116 int __pyx_clineno = 0;
14117 __Pyx_RefNannySetupContext("pow", 0);
14118 __Pyx_XDECREF(__pyx_r);
14119 __pyx_t_1 = __pyx_pf_8PyClical_pow(__pyx_v_obj, __pyx_v_m, 0); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1543, __pyx_L1_error)
14120 __Pyx_GOTREF(__pyx_t_1);
14121 __pyx_r = __pyx_t_1;
14122 __pyx_t_1 = 0;
14123 goto __pyx_L0;
14124
14125 /* function exit code */

```

```

14126 __pyx_L1_error;;
14127 __Pyx_XDECREF(__pyx_t_1);
14128 __Pyx_AddTraceback("PyClical.pow", __pyx_clineno, __pyx_lineno, __pyx_filename);
14129 __pyx_r = NULL;
14130 __pyx_L0;;
14131 __Pyx_XGIVEREF(__pyx_r);
14132 __Pyx_RefNannyFinishContext();
14133 return __pyx_r;
14134 }
14135
14136 /* "PyClical.pyx":1567
14137 * return clifford(obj).pow(m)
14138 *
14139 * cpdef inline outer_pow(obj, m):
14140 * """
14141 * Outer product power of multivector.
14142 */
14143
14144 static PyObject *__pyx_pw_8PyClical_47outer_pow(PyObject *__pyx_self, PyObject
14145 *__pyx_args, PyObject *__pyx_kwds); /*proto*/
14146 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_outer_pow(PyObject *__pyx_v_obj,
14147 PyObject *__pyx_v_m, CYTHON_UNUSED int __pyx_skip_dispatch) {
14148 PyObject *__pyx_r = NULL;
14149 __Pyx_RefNannyDeclarations
14150 PyObject *__pyx_t_1 = NULL;
14151 PyObject *__pyx_t_2 = NULL;
14152 PyObject *__pyx_t_3 = NULL;
14153 int __pyx_lineno = 0;
14154 const char *__pyx_filename = NULL;
14155 int __pyx_clineno = 0;
14156 __Pyx_RefNannySetupContext("outer_pow", 0);
14157
14158 /* "PyClical.pyx":1574
14159 * 1+3{1}+3{1,2}
14160 * return clifford(obj).outer_pow(m)
14161 * """
14162 * cpdef inline complexifier(obj):
14163 */
14164 __Pyx_XDECREF(__pyx_r);
14165 __pyx_t_2 = __Pyx_PyObject_CallOneArg((PyObject *)__pyx_ptype_8PyClical_clifford),
14166 __pyx_v_obj); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1574, __pyx_L1_error)
14167 __Pyx_GOTREF(__pyx_t_2);
14168 __pyx_t_3 = __Pyx_PyObject_GetAttrStr(__pyx_t_2, __pyx_n_s_outer_pow); if
14169 (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1574, __pyx_L1_error)
14170 __Pyx_GOTREF(__pyx_t_3);
14171 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
14172 __pyx_t_2 = NULL;
14173 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_3))) {
14174 __pyx_t_2 = PyMethod_GET_SELF(__pyx_t_3);
14175 if (likely(__pyx_t_2)) {
14176 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_3);
14177 __Pyx_INCREF(__pyx_t_2);
14178 __Pyx_INCREF(function);
14179 __Pyx_DECREF_SET(__pyx_t_3, function);
14180 }
14181 }
14182 __pyx_t_1 = (__pyx_t_2) ? __Pyx_PyObject_Call2Args(__pyx_t_3, __pyx_t_2, __pyx_v_m)
14183 : __Pyx_PyObject_CallOneArg(__pyx_t_3, __pyx_v_m);
14184 __Pyx_XDECREF(__pyx_t_2); __pyx_t_2 = 0;
14185 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1574, __pyx_L1_error)
14186 __Pyx_GOTREF(__pyx_t_1);
14187 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
14188 __pyx_r = __pyx_t_1;
14189 __pyx_t_1 = 0;
14190 goto __pyx_L0;
14191
14192 /* "PyClical.pyx":1567
14193 * return clifford(obj).pow(m)
14194 *
14195 * cpdef inline outer_pow(obj, m):
14196 * """
14197 * Outer product power of multivector.
14198 */
14199
14200 /* function exit code */
14201 __pyx_L1_error;;
14202 __Pyx_XDECREF(__pyx_t_1);
14203 __Pyx_XDECREF(__pyx_t_2);
14204 __Pyx_XDECREF(__pyx_t_3);
14205 __Pyx_AddTraceback("PyClical.outer_pow", __pyx_clineno, __pyx_lineno,
14206 __pyx_filename);
14207 __pyx_r = 0;
14208 __pyx_L0;;
14209 __Pyx_XGIVEREF(__pyx_r);
14210 __Pyx_RefNannyFinishContext();
14211 return __pyx_r;

```

```

14207 }
14208
14209 /* Python wrapper */
14210 static PyObject *__pyx_pw_8PyClical_47outer_pow(PyObject *__pyx_self, PyObject
14211 *__pyx_args, PyObject *__pyx_kwds); /*proto*/
14212 static char __pyx_doc_8PyClical_46outer_pow[] = "\n Outer product power of
multivector.\n\n >> print(outer_pow(clifford(\"1+{1}+{1,2}\"),3))\n 1+3{1}+3{1,2}\n ";
14213 static PyObject *__pyx_pw_8PyClical_47outer_pow(PyObject *__pyx_self, PyObject
14214 *__pyx_args, PyObject *__pyx_kwds) {
14215 PyObject *__pyx_v_obj = 0;
14216 PyObject *__pyx_v_m = 0;
14217 int __pyx_lineno = 0;
14218 const char *__pyx_filename = NULL;
14219 int __pyx_clineno = 0;
14220 PyObject *__pyx_r = 0;
14221 __Pyx_RefNannyDeclarations
14222 __Pyx_RefNannySetupContext("outer_pow (wrapper)", 0);
14223 {
14224 static PyObject *__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_m,0};
14225 PyObject* values[2] = {0,0};
14226 if (unlikely(__pyx_kwds)) {
14227 Py_ssize_t kw_args;
14228 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
14229 switch (pos_args) {
14230 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
14231 CYTHON_FALLTHROUGH;
14232 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
14233 CYTHON_FALLTHROUGH;
14234 case 0: break;
14235 default: goto __pyx_L5_argtuple_error;
14236 }
14237 kw_args = PyDict_Size(__pyx_kwds);
14238 switch (pos_args) {
14239 case 0:
14240 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
0)) kw_args--;
14241 else goto __pyx_L5_argtuple_error;
14242 CYTHON_FALLTHROUGH;
14243 case 1:
14244 if (likely((values[1] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_m)) !=
0)) kw_args--;
14245 else {
14246 __Pyx_RaiseArgtupleInvalid("outer_pow", 1, 2, 2, 1); __PYX_ERR(0, 1567,
__pyx_L3_error)
14247 }
14248 if (unlikely(kw_args > 0)) {
14249 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "outer_pow") < 0)) __PYX_ERR(0, 1567, __pyx_L3_error)
14250 } else if (PyTuple_GET_SIZE(__pyx_args) != 2) {
14251 goto __pyx_L5_argtuple_error;
14252 } else {
14253 values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
14254 values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
14255 }
14256 __pyx_v_obj = values[0];
14257 __pyx_v_m = values[1];
14258 }
14259 goto __pyx_L4_argument_unpacking_done;
14260 __pyx_L5_argtuple_error:;
14261 __Pyx_RaiseArgtupleInvalid("outer_pow", 1, 2, 2, PyTuple_GET_SIZE(__pyx_args));
14262 __PYX_ERR(0, 1567, __pyx_L3_error)
14263 __pyx_L3_error:;
14264 __Pyx_AddTraceback("PyClical.outer_pow", __pyx_clineno, __pyx_lineno,
__pyx_filename);
14265 __Pyx_RefNannyFinishContext();
14266 return NULL;
14267 __pyx_L4_argument_unpacking_done:;
14268 __pyx_r = __pyx_pf_8PyClical_46outer_pow(__pyx_self, __pyx_v_obj, __pyx_v_m);
14269
14270 /* function exit code */
14271 __Pyx_RefNannyFinishContext();
14272 return __pyx_r;
14273 }
14274 static PyObject *__pyx_pf_8PyClical_46outer_pow(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj, PyObject *__pyx_v_m) {
14275 PyObject *__pyx_r = NULL;
14276 __Pyx_RefNannyDeclarations
14277 PyObject *__pyx_t_1 = NULL;
14278 int __pyx_lineno = 0;
14279 const char *__pyx_filename = NULL;
14280 int __pyx_clineno = 0;
14281 __Pyx_RefNannySetupContext("outer_pow", 0);
14282 __Pyx_XDECREF(__pyx_r);
14283 __pyx_t_1 = __pyx_f_8PyClical_outer_pow(__pyx_v_obj, __pyx_v_m, 0); if

```

```

(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1567, __pyx_L1_error)
14284 __Pyx_GOTREF(__pyx_t_1);
14285 __pyx_r = __pyx_t_1;
14286 __pyx_t_1 = 0;
14287 goto __pyx_L0;
14288
14289 /* function exit code */
14290 __pyx_L1_error;
14291 __Pyx_XDECREF(__pyx_t_1);
14292 __Pyx_AddTraceback("PyClical.outer_pow", __pyx_clineno, __pyx_lineno,
__pyx_filename);
14293 __pyx_r = NULL;
14294 __pyx_L0;
14295 __Pyx_XGIVEREF(__pyx_r);
14296 __Pyx_RefNannyFinishContext();
14297 return __pyx_r;
14298 }
14299
14300 /* "PyClical.pyx":1576
14301 * return clifford(obj).outer_pow(m)
14302 *
14303 * cpdef inline complexifier(obj): # ««««««««
14304 * """
14305 * Square root of -1 which commutes with all members of the frame of the given multivector.
14306 */
14307
14308 static PyObject * __pyx_pw_8PyClical_49complexifier(PyObject * __pyx_self, PyObject
* __pyx_v_obj); /*proto*/
14309 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_complexifier(PyObject * __pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
14310 PyObject * __pyx_r = NULL;
14311 __Pyx_RefNannyDeclarations
14312 PyObject * __pyx_t_1 = NULL;
14313 PyObject * __pyx_t_2 = NULL;
14314 int __pyx_lineno = 0;
14315 const char * __pyx_filename = NULL;
14316 int __pyx_clineno = 0;
14317 __Pyx_RefNannySetupContext("complexifier", 0);
14318
14319 /* "PyClical.pyx":1589
14320 * {-1}
14321 * """
14322 * return clifford().wrap(glucat.complexifier(toClifford(obj))) # ««««««««
14323 *
14324 * cpdef inline sqrt(obj, i = None):
14325 */
14326 __Pyx_XDECREF(__pyx_r);
14327 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *) __pyx_ptype_8PyClical_clifford));
14328 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1589, __pyx_L1_error)
14329 __Pyx_GOTREF(__pyx_t_1);
14330 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*) __pyx_t_1), complexifier(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_2))
__PYX_ERR(0, 1589, __pyx_L1_error)
14331 __Pyx_GOTREF(__pyx_t_2);
14332 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
14333 __pyx_r = __pyx_t_2;
14334 __pyx_t_2 = 0;
14335 goto __pyx_L0;
14336
14337 /* "PyClical.pyx":1576
14338 * return clifford(obj).outer_pow(m)
14339 *
14340 * cpdef inline complexifier(obj): # ««««««««
14341 * """
14342 * Square root of -1 which commutes with all members of the frame of the given multivector.
14343 */
14344
14345 /* function exit code */
14346 __pyx_L1_error;
14347 __Pyx_XDECREF(__pyx_t_1);
14348 __Pyx_XDECREF(__pyx_t_2);
14349 __Pyx_AddTraceback("PyClical.complexifier", __pyx_clineno, __pyx_lineno,
__pyx_filename);
14350 __pyx_r = 0;
14351 __pyx_L0;
14352 __Pyx_XGIVEREF(__pyx_r);
14353 __Pyx_RefNannyFinishContext();
14354 return __pyx_r;
14355 }
14356
14357 /* Python wrapper */
14358 static PyObject * __pyx_pw_8PyClical_49complexifier(PyObject * __pyx_self, PyObject
* __pyx_v_obj); /*proto*/
14359 static char __pyx_doc_8PyClical_48complexifier[] = "\n Square root of -1 which
commutes with all members of the frame of the given multivector.\n\n >>
print(complexifier(clifford(index_set({1})))\n {1,2,3}\n >>
print(complexifier(clifford(index_set({-1})))\n {-1}\n >> print(complexifier(index_set({1})))\n

```

```

 {1,2,3}\n >> print(complexifier(index_set({-1}))\n {-1}\n ";
14359 static PyObject *__pyx_pw_8PyClical_49complexifier(PyObject *__pyx_self, PyObject
 *__pyx_v_obj) {
14360 PyObject *__pyx_r = 0;
14361 __Pyx_RefNannyDeclarations
14362 __Pyx_RefNannySetupContext("complexifier (wrapper)", 0);
14363 __pyx_r = __pyx_pf_8PyClical_48complexifier(__pyx_self, ((PyObject *)__pyx_v_obj));
14364
14365 /* function exit code */
14366 __Pyx_RefNannyFinishContext();
14367 return __pyx_r;
14368 }
14369
14370 static PyObject *__pyx_pf_8PyClical_48complexifier(CYTHON_UNUSED PyObject *__pyx_self,
 PyObject *__pyx_v_obj) {
14371 PyObject *__pyx_r = NULL;
14372 __Pyx_RefNannyDeclarations
14373 PyObject *__pyx_t_1 = NULL;
14374 int __pyx_lineno = 0;
14375 const char *__pyx_filename = NULL;
14376 int __pyx_clineno = 0;
14377 __Pyx_RefNannySetupContext("complexifier", 0);
14378 __Pyx_XDECREF(__pyx_r);
14379 __pyx_t_1 = __pyx_f_8PyClical_complexifier(__pyx_v_obj, 0); if
 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1576, __pyx_L1_error)
14380 __Pyx_GOTREF(__pyx_t_1);
14381 __pyx_r = __pyx_t_1;
14382 __pyx_t_1 = 0;
14383 goto __pyx_L0;
14384
14385 /* function exit code */
14386 __pyx_L1_error;
14387 __Pyx_XDECREF(__pyx_t_1);
14388 __Pyx_AddTraceback("PyClical.complexifier", __pyx_clineno, __pyx_lineno,
 __pyx_filename);
14389 __pyx_r = NULL;
14390 __pyx_L0;
14391 __Pyx_XGIVEREF(__pyx_r);
14392 __Pyx_RefNannyFinishContext();
14393 return __pyx_r;
14394 }
14395
14396 /* "PyClical.pyx":1591
14397 * return clifford().wrap(glucat.complexifier(toClifford(obj)))
14398 *
14399 * cpdef inline sqrt(obj, i = None):
14400 * """
14401 * Square root of multivector with optional complexifier.
14402 */
14403
14404 static PyObject *__pyx_pw_8PyClical_51sqrt(PyObject *__pyx_self, PyObject *__pyx_args,
 PyObject *__pyx_kwds); /*proto*/
14405 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_sqrt(PyObject *__pyx_v_obj,
 CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_sqrt *__pyx_optional_args) {
14406 PyObject *__pyx_v_i = ((PyObject *)Py_None);
14407 PyObject *__pyx_r = NULL;
14408 __Pyx_RefNannyDeclarations
14409 int __pyx_t_1;
14410 int __pyx_t_2;
14411 PyObject *__pyx_t_3 = NULL;
14412 Clifford __pyx_t_4;
14413 PyObject *__pyx_t_5 = NULL;
14414 PyObject *__pyx_t_6 = NULL;
14415 PyObject *__pyx_t_7 = NULL;
14416 PyObject *__pyx_t_8 = NULL;
14417 PyObject *__pyx_t_9 = NULL;
14418 PyObject *__pyx_t_10 = NULL;
14419 PyObject *__pyx_t_11 = NULL;
14420 int __pyx_lineno = 0;
14421 const char *__pyx_filename = NULL;
14422 int __pyx_clineno = 0;
14423 __Pyx_RefNannySetupContext("sqrt", 0);
14424 if (__pyx_optional_args) {
14425 if (__pyx_optional_args->__pyx_n > 0) {
14426 __pyx_v_i = __pyx_optional_args->1;
14427 }
14428 }
14429
14430 /* "PyClical.pyx":1606
14431 * -1
14432 * """
14433 * if not (i is None):
14434 * return clifford().wrap(glucat.sqrt(toClifford(obj), toClifford(i)))
14435 * else:
14436 */
14437 __pyx_t_1 = (__pyx_v_i != Py_None);
14438 __pyx_t_2 = (__pyx_t_1 != 0);

```

```

14439 if (__pyx_t_2) {
14440
14441 /* "PyClicl.pyx":1607
14442 *
14443 * if not (i is None):
14444 * return clifford().wrap(glucat.sqrt(toClifford(obj), toClifford(i))) # ««««««««
14445 * else:
14446 * try:
14447 */
14448 __Pyx_XDECREF(__pyx_r);
14449 __pyx_t_3 = __Pyx_PyObject_CallNoArg((PyObject
*)__pyx_ptype_8PyClicl_clifford); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1607, __pyx_L1_error)
14450 __Pyx_GOTREF(__pyx_t_3);
14451 try {
14452 __pyx_t_4 = sqrt(__pyx_f_8PyClicl_toClifford(__pyx_v_obj),
__pyx_f_8PyClicl_toClifford(__pyx_v_i));
14453 } catch (...) {
14454 __Pyx_CppExn2PyErr();
14455 __PYX_ERR(0, 1607, __pyx_L1_error)
14456 }
14457 __pyx_t_5 = __pyx_f_8PyClicl_8clifford_wrap(((struct __pyx_obj_8PyClicl_clifford
*)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1607, __pyx_L1_error)
14458 __Pyx_GOTREF(__pyx_t_5);
14459 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
14460 __pyx_r = __pyx_t_5;
14461 __pyx_t_5 = 0;
14462 goto __pyx_L0;
14463
14464 /* "PyClicl.pyx":1606
14465 *
14466 * -1
14467 *
14468 * if not (i is None): # ««««««««
14469 * return clifford().wrap(glucat.sqrt(toClifford(obj), toClifford(i)))
14470 * else:
14471 */
14472 }
14473
14474 /* "PyClicl.pyx":1609
14475 * return clifford().wrap(glucat.sqrt(toClifford(obj), toClifford(i)))
14476 * else:
14477 * try: # ««««««««
14478 * return math.sqrt(obj)
14479 * except:
14480 */
14481 /*else*/ {
14482 {
14483 __Pyx_PyThreadState_declare
14484 __Pyx_PyThreadState_assign
14485 __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
14486 __Pyx_XGOTREF(__pyx_t_6);
14487 __Pyx_XGOTREF(__pyx_t_7);
14488 __Pyx_XGOTREF(__pyx_t_8);
14489 /*try:*/ {
14490
14491 /* "PyClicl.pyx":1610
14492 *
14493 * else:
14494 * try:
14495 * return math.sqrt(obj) # ««««««««
14496 * except:
14497 * return clifford().wrap(glucat.sqrt(toClifford(obj)))
14498 */
14499 __Pyx_XDECREF(__pyx_r);
14500 __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1610, __pyx_L4_error)
14501 __Pyx_GOTREF(__pyx_t_3);
14502 __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_sqrt); if
(unlikely(!__pyx_t_9)) __PYX_ERR(0, 1610, __pyx_L4_error)
14503 __Pyx_GOTREF(__pyx_t_9);
14504 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
14505 __pyx_t_3 = NULL;
14506 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
14507 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
14508 if (likely(__pyx_t_3)) {
14509 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
14510 __Pyx_INCREF(__pyx_t_3);
14511 __Pyx_INCREF(function);
14512 __Pyx_DECREF_SET(__pyx_t_9, function);
14513 }
14514 }
14515 __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
14516 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
14517 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1610, __pyx_L4_error)
14518 __Pyx_GOTREF(__pyx_t_5);
14519 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
14520 __pyx_r = __pyx_t_5;
14521 __pyx_t_5 = 0;

```



```

14520 goto __pyx_L8_try_return;
14521
14522 /* "PyClical.pyx":1609
14523 * return clifford().wrap(glucat.sqrt(toClifford(obj)), toClifford(i)))
14524 * else:
14525 * try:
14526 * # ««««««««
14527 * return math.sqrt(obj)
14528 * except:
14529 */
14530 }
14531 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
14532 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
14533 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
14534
14535 /* "PyClical.pyx":1611
14536 * try:
14537 * return math.sqrt(obj)
14538 * except:
14539 * # ««««««««
14540 * return clifford().wrap(glucat.sqrt(toClifford(obj)))
14541 */
14542
14543 /*except:*/ {
14544 __Pyx_AddTraceback("PyClical.sqrt", __pyx_clineno, __pyx_lineno,
__pyx_filename);
14545 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
1611, __pyx_L6_except_error)
14546 __Pyx_GOTREF(__pyx_t_5);
14547 __Pyx_GOTREF(__pyx_t_9);
14548 __Pyx_GOTREF(__pyx_t_3);
14549
14550 /* "PyClical.pyx":1612
14551 * return math.sqrt(obj)
14552 * except:
14553 * return clifford().wrap(glucat.sqrt(toClifford(obj)))
14554 * # ««««««««
14555 * cpdef inline exp(obj):
14556 */
14557 __Pyx_XDECREF(__pyx_r);
14558 __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1612,
__pyx_L6_except_error)
14559 __Pyx_GOTREF(__pyx_t_10);
14560 __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct
__pyx_obj_8PyClical_clifford *)__pyx_t_10), sqrt(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 1612, __pyx_L6_except_error)
14561 __Pyx_GOTREF(__pyx_t_11);
14562 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
14563 __pyx_r = __pyx_t_11;
14564 __pyx_t_11 = 0;
14565 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
14566 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
14567 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
14568 goto __pyx_L7_except_return;
14569 }
14570 __pyx_L6_except_error:;
14571
14572 /* "PyClical.pyx":1609
14573 * return clifford().wrap(glucat.sqrt(toClifford(obj)), toClifford(i)))
14574 * else:
14575 * try:
14576 * # ««««««««
14577 * return math.sqrt(obj)
14578 * except:
14579 */
14580 __Pyx_XGIVEREF(__pyx_t_6);
14581 __Pyx_XGIVEREF(__pyx_t_7);
14582 __Pyx_XGIVEREF(__pyx_t_8);
14583 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
14584 goto __pyx_L1_error;
14585 __pyx_L8_try_return:;
14586 __Pyx_XGIVEREF(__pyx_t_6);
14587 __Pyx_XGIVEREF(__pyx_t_7);
14588 __Pyx_XGIVEREF(__pyx_t_8);
14589 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
14590 goto __pyx_L0;
14591 __pyx_L7_except_return:;
14592 __Pyx_XGIVEREF(__pyx_t_6);
14593 __Pyx_XGIVEREF(__pyx_t_7);
14594 __Pyx_XGIVEREF(__pyx_t_8);
14595 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
14596 goto __pyx_L0;
14597 }
14598
14599 /* "PyClical.pyx":1591
14600 * return clifford().wrap(glucat.complexifier(toClifford(obj)))
14601 */

```

```

14601 * cpdef inline sqrt(obj, i = None): # ««««««««
14602 * """
14603 * Square root of multivector with optional complexifier.
14604 */
14605
14606 /* function exit code */
14607 __pyx_L1_error++;
14608 __Pyx_XDECREF(__pyx_t_3);
14609 __Pyx_XDECREF(__pyx_t_5);
14610 __Pyx_XDECREF(__pyx_t_9);
14611 __Pyx_XDECREF(__pyx_t_10);
14612 __Pyx_XDECREF(__pyx_t_11);
14613 __Pyx_AddTraceback("PyClical.sqrt", __pyx_clineno, __pyx_lineno, __pyx_filename);
14614 __pyx_r = 0;
14615 __pyx_L0:;
14616 __Pyx_XGIVEREF(__pyx_r);
14617 __Pyx_RefNannyFinishContext();
14618 return __pyx_r;
14619 }
14620
14621 /* Python wrapper */
14622 static PyObject * __pyx_pw_8PyClical_5lsqrt(PyObject * __pyx_self, PyObject * __pyx_args,
14623 PyObject * __pyx_kwds); /*proto*/
14624 static char __pyx_doc_8PyClical_50sqrt[] = "\n Square root of multivector with
optional complexifier.\n\n >> print(sqrt(-1))\n {-1}\n >> print(sqrt(clifford(\"2{-1}\")))\n
1+{-1}\n >> j=sqrt(-1,complexifier(index_set({1})))\n {j}\n {j*j}\n {1,2,3}\n -1\n
>> j=sqrt(-1,\"{1,2,3}\"); print(j); print(j*j)\n {1,2,3}\n -1\n ";
14625 static PyObject * __pyx_pw_8PyClical_5lsqrt(PyObject * __pyx_self, PyObject * __pyx_args,
14626 PyObject * __pyx_kwds) {
14627 PyObject * __pyx_v_obj = 0;
14628 PyObject * __pyx_v_i = 0;
14629 int __pyx_lineno = 0;
14630 const char * __pyx_filename = NULL;
14631 int __pyx_clineno = 0;
14632 PyObject * __pyx_r = 0;
14633 __Pyx_RefNannyDeclarations
14634 __Pyx_RefNannySetupContext("sqrt (wrapper)", 0);
14635 {
14636 static PyObject * __pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
14637 PyObject* values[2] = {0,0};
14638 values[1] = ((PyObject *)Py_None);
14639 if (unlikely(__pyx_kwds)) {
14640 Py_ssize_t kw_args;
14641 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
14642 switch (pos_args) {
14643 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
14644 CYTHON_FALLTHROUGH;
14645 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
14646 CYTHON_FALLTHROUGH;
14647 case 0: break;
14648 default: goto __pyx_L5_argtuple_error;
14649 }
14650 kw_args = PyDict_Size(__pyx_kwds);
14651 switch (pos_args) {
14652 case 0:
14653 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
14654 0)) kw_args--;
14655 else goto __pyx_L5_argtuple_error;
14656 CYTHON_FALLTHROUGH;
14657 case 1:
14658 if (kw_args > 0) {
14659 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
14660 if (value) { values[1] = value; kw_args--; }
14661 }
14662 if (unlikely(kw_args > 0)) {
14663 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
14664 values, pos_args, "sqrt") < 0)) __PYX_ERR(0, 1591, __pyx_L3_error)
14665 }
14666 } else {
14667 switch (PyTuple_GET_SIZE(__pyx_args)) {
14668 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
14669 CYTHON_FALLTHROUGH;
14670 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
14671 break;
14672 default: goto __pyx_L5_argtuple_error;
14673 }
14674 }
14675 __pyx_v_obj = values[0];
14676 __pyx_v_i = values[1];
14677 }
14678 goto __pyx_L4_argument_unpacking_done;
14679 __pyx_L5_argtuple_error:;
14680 __Pyx_RaiseArgtupleInvalid("sqrt", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
14681 __PYX_ERR(0, 1591, __pyx_L3_error)
14682 __pyx_L3_error:;
14683 __Pyx_AddTraceback("PyClical.sqrt", __pyx_clineno, __pyx_lineno, __pyx_filename);

```

```

14680 __Pyx_RefNannyFinishContext();
14681 return NULL;
14682 __pyx_L4_argument_unpacking_done:;
14683 __pyx_r = __pyx_pf_8PyClical_50sqrt(__pyx_self, __pyx_v_obj, __pyx_v_i);
14684
14685 /* function exit code */
14686 __Pyx_RefNannyFinishContext();
14687 return __pyx_r;
14688 }
14689
14690 static PyObject *__pyx_pf_8PyClical_50sqrt(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj, PyObject *__pyx_v_i) {
14691 PyObject *__pyx_r = NULL;
14692 __Pyx_RefNannyDeclarations
14693 PyObject *__pyx_t_1 = NULL;
14694 struct __pyx_opt_args_8PyClical_sqrt __pyx_t_2;
14695 int __pyx_lineno = 0;
14696 const char *__pyx_filename = NULL;
14697 int __pyx_clineno = 0;
14698 __Pyx_RefNannySetupContext("sqrt", 0);
14699 __Pyx_XDECREF(__pyx_r);
14700 __pyx_t_2.__pyx_n = 1;
14701 __pyx_t_2.i = __pyx_v_i;
14702 __pyx_t_1 = __pyx_f_8PyClical_sqrt(__pyx_v_obj, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1591, __pyx_L1_error)
14703 __Pyx_GOTREF(__pyx_t_1);
14704 __pyx_r = __pyx_t_1;
14705 __pyx_t_1 = 0;
14706 goto __pyx_L0;
14707
14708 /* function exit code */
14709 __pyx_L1_error:;
14710 __Pyx_XDECREF(__pyx_t_1);
14711 __Pyx_AddTraceback("PyClical.sqrt", __pyx_clineno, __pyx_lineno, __pyx_filename);
14712 __pyx_r = NULL;
14713 __pyx_L0:;
14714 __Pyx_XGIVEREF(__pyx_r);
14715 __Pyx_RefNannyFinishContext();
14716 return __pyx_r;
14717 }
14718
14719 /* "PyClical.pyx":1614
14720 *
14721 * return clifford().wrap(glucat.sqrt(toClifford(obj)))
14722 * cpdef inline exp(obj):
14723 * """
14724 * Exponential of multivector.
14725 */
14726
14727 static PyObject *__pyx_pw_8PyClical_53exp(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
14728 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_exp(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
14729 PyObject *__pyx_r = NULL;
14730 __Pyx_RefNannyDeclarations
14731 PyObject *__pyx_t_1 = NULL;
14732 PyObject *__pyx_t_2 = NULL;
14733 PyObject *__pyx_t_3 = NULL;
14734 PyObject *__pyx_t_4 = NULL;
14735 PyObject *__pyx_t_5 = NULL;
14736 PyObject *__pyx_t_6 = NULL;
14737 PyObject *__pyx_t_7 = NULL;
14738 PyObject *__pyx_t_8 = NULL;
14739 int __pyx_lineno = 0;
14740 const char *__pyx_filename = NULL;
14741 int __pyx_clineno = 0;
14742 __Pyx_RefNannySetupContext("exp", 0);
14743
14744 /* "PyClical.pyx":1623
14745 * {1,2}
14746 * """
14747 * try:
14748 * return math.exp(obj)
14749 * except:
14750 */
14751 {
14752 __Pyx_PyThreadState_declare
14753 __Pyx_PyThreadState_assign
14754 __Pyx_ExceptionSave(&__pyx_t_1, &__pyx_t_2, &__pyx_t_3);
14755 __Pyx_XGOTREF(__pyx_t_1);
14756 __Pyx_XGOTREF(__pyx_t_2);
14757 __Pyx_XGOTREF(__pyx_t_3);
14758 /*try:*/ {
14759
14760 /* "PyClical.pyx":1624
14761 * """
14762 * try:

```

```

14763 * return math.exp(obj) # ««««««««
14764 * except:
14765 * return clifford().wrap(glucat.exp(toClifford(obj)))
14766 */
14767 __Pyx_XDECREF(__pyx_r);
14768 __Pyx_GetModuleGlobalName(__pyx_t_5, __pyx_n_s_math); if (unlikely(!__pyx_t_5))
__PYX_ERR(0, 1624, __pyx_L3_error)
14769 __Pyx_GOTREF(__pyx_t_5);
14770 __pyx_t_6 = __Pyx_PyObject_GetAttrStr(__pyx_t_5, __pyx_n_s_exp); if
(unlikely(!__pyx_t_6)) __PYX_ERR(0, 1624, __pyx_L3_error)
14771 __Pyx_GOTREF(__pyx_t_6);
14772 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
14773 __pyx_t_5 = NULL;
14774 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_6))) {
14775 __pyx_t_5 = PyMethod_GET_SELF(__pyx_t_6);
14776 if (likely(__pyx_t_5)) {
14777 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_6);
14778 __Pyx_INCREF(__pyx_t_5);
14779 __Pyx_INCREF(function);
14780 __Pyx_DECREF_SET(__pyx_t_6, function);
14781 }
14782 }
14783 __pyx_t_4 = (__pyx_t_5) ? __Pyx_PyObject_Call2Args(__pyx_t_6, __pyx_t_5,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_6, __pyx_v_obj);
14784 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
14785 if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1624, __pyx_L3_error)
14786 __Pyx_GOTREF(__pyx_t_4);
14787 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
14788 __pyx_r = __pyx_t_4;
14789 __pyx_t_4 = 0;
14790 goto __pyx_L7_try_return;
14791
14792 /* "PyClicl.pyx":1623
14793 * {1,2}
14794 * """
14795 * try: # ««««««««
14796 * return math.exp(obj)
14797 * except:
14798 */
14799 }
14800 __pyx_L3_error:;
14801 __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
14802 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
14803 __Pyx_XDECREF(__pyx_t_6); __pyx_t_6 = 0;
14804
14805 /* "PyClicl.pyx":1625
14806 * try:
14807 * return math.exp(obj)
14808 * except: # ««««««««
14809 * return clifford().wrap(glucat.exp(toClifford(obj)))
14810 *
14811 */
14812 /*except:*/ {
14813 __Pyx_AddTraceback("PyClicl.exp", __pyx_clineno, __pyx_lineno, __pyx_filename);
14814 if (__Pyx_GetException(&__pyx_t_4, &__pyx_t_6, &__pyx_t_5) < 0) __PYX_ERR(0,
1625, __pyx_L5_except_error)
14815 __Pyx_GOTREF(__pyx_t_4);
14816 __Pyx_GOTREF(__pyx_t_6);
14817 __Pyx_GOTREF(__pyx_t_5);
14818
14819 /* "PyClicl.pyx":1626
14820 * return math.exp(obj)
14821 * except:
14822 * return clifford().wrap(glucat.exp(toClifford(obj))) # ««««««««
14823 *
14824 * cpdef inline log(obj,i = None):
14825 */
14826 __Pyx_XDECREF(__pyx_r);
14827 __pyx_t_7 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClicl_clifford)); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 1626,
__pyx_L5_except_error)
14828 __Pyx_GOTREF(__pyx_t_7);
14829 __pyx_t_8 = __pyx_f_8PyClicl_8clifford_wrap(((struct
__pyx_obj_8PyClicl_clifford *)__pyx_t_7), exp(__pyx_f_8PyClicl_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_8)) __PYX_ERR(0, 1626, __pyx_L5_except_error)
14830 __Pyx_GOTREF(__pyx_t_8);
14831 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
14832 __pyx_r = __pyx_t_8;
14833 __pyx_t_8 = 0;
14834 __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
14835 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
14836 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
14837 goto __pyx_L6_except_return;
14838 }
14839 __pyx_L5_except_error:;
14840
14841 /* "PyClicl.pyx":1623

```

```

14842 * {1,2}
14843 * """
14844 * try: # ««««««««
14845 * return math.exp(obj)
14846 * except:
14847 */
14848 __Pyx_XGIVEREF(__pyx_t_1);
14849 __Pyx_XGIVEREF(__pyx_t_2);
14850 __Pyx_XGIVEREF(__pyx_t_3);
14851 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
14852 goto __pyx_L1_error;
14853 __pyx_L7_try_return:;
14854 __Pyx_XGIVEREF(__pyx_t_1);
14855 __Pyx_XGIVEREF(__pyx_t_2);
14856 __Pyx_XGIVEREF(__pyx_t_3);
14857 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
14858 goto __pyx_L0;
14859 __pyx_L6_except_return:;
14860 __Pyx_XGIVEREF(__pyx_t_1);
14861 __Pyx_XGIVEREF(__pyx_t_2);
14862 __Pyx_XGIVEREF(__pyx_t_3);
14863 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
14864 goto __pyx_L0;
14865 }
14866
14867 /* "PyClical.pyx":1614
14868 * return clifford().wrap(glucat.sqrt(toClifford(obj)))
14869 *
14870 * cpdef inline exp(obj): # ««««««««
14871 * """
14872 * Exponential of multivector.
14873 */
14874
14875 /* function exit code */
14876 __pyx_L1_error:;
14877 __Pyx_XDECREF(__pyx_t_4);
14878 __Pyx_XDECREF(__pyx_t_5);
14879 __Pyx_XDECREF(__pyx_t_6);
14880 __Pyx_XDECREF(__pyx_t_7);
14881 __Pyx_XDECREF(__pyx_t_8);
14882 __Pyx_AddTraceback("PyClical.exp", __pyx_clineno, __pyx_lineno, __pyx_filename);
14883 __pyx_r = 0;
14884 __pyx_L0:;
14885 __Pyx_XGIVEREF(__pyx_r);
14886 __Pyx_RefNannyFinishContext();
14887 return __pyx_r;
14888 }
14889
14890 /* Python wrapper */
14891 static PyObject * __pyx_pw_8PyClical_53exp(PyObject * __pyx_self, PyObject
*__pyx_v_obj); /*proto*/
14892 static char __pyx_doc_8PyClical_52exp[] = "\n Exponential of multivector.\n\n >>
x=clifford(\"{1,2}\") * pi/4; print(exp(x))\n 0.7071+0.7071j{1,2}\n >> x=clifford(\"{1,2}\") *
pi/2; print(exp(x))\n {1,2}\n ";
14893 static PyObject * __pyx_pw_8PyClical_53exp(PyObject * __pyx_self, PyObject * __pyx_v_obj)
{
14894 PyObject * __pyx_r = 0;
14895 __Pyx_RefNannyDeclarations
14896 __Pyx_RefNannySetupContext("exp (wrapper)", 0);
14897 __pyx_r = __pyx_pf_8PyClical_52exp(__pyx_self, ((PyObject *) __pyx_v_obj));
14898
14899 /* function exit code */
14900 __Pyx_RefNannyFinishContext();
14901 return __pyx_r;
14902 }
14903
14904 static PyObject * __pyx_pf_8PyClical_52exp(CYTHON_UNUSED PyObject * __pyx_self, PyObject
*__pyx_v_obj) {
14905 PyObject * __pyx_r = NULL;
14906 __Pyx_RefNannyDeclarations
14907 PyObject * __pyx_t_1 = NULL;
14908 int __pyx_lineno = 0;
14909 const char * __pyx_filename = NULL;
14910 int __pyx_clineno = 0;
14911 __Pyx_RefNannySetupContext("exp", 0);
14912 __Pyx_XDECREF(__pyx_r);
14913 __pyx_t_1 = __pyx_f_8PyClical_exp(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1614, __pyx_L1_error)
14914 __Pyx_GOTREF(__pyx_t_1);
14915 __pyx_r = __pyx_t_1;
14916 __pyx_t_1 = 0;
14917 goto __pyx_L0;
14918
14919 /* function exit code */
14920 __pyx_L1_error:;
14921 __Pyx_XDECREF(__pyx_t_1);
14922 __Pyx_AddTraceback("PyClical.exp", __pyx_clineno, __pyx_lineno, __pyx_filename);

```

```

14923 __pyx_r = NULL;
14924 __pyx_L0:;
14925 __Pyx_XGIVEREF(__pyx_r);
14926 __Pyx_RefNannyFinishContext();
14927 return __pyx_r;
14928 }
14929
14930 /* "PyClicl.pyx":1628
14931 * return clifford().wrap(glucat.exp(toClifford(obj)))
14932 *
14933 * cpdef inline log(obj,i = None):
14934 * """
14935 * Natural logarithm of multivector with optional complexifier.
14936 */
14937
14938 static PyObject *__pyx_pw_8PyClicl_55log(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds); /*proto*/
14939 static CYTHON_INLINE PyObject *__pyx_f_8PyClicl_log(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClicl_log *__pyx_optional_args) {
14940 PyObject *__pyx_v_i = ((PyObject *)Py_None);
14941 PyObject *__pyx_r = NULL;
14942 __Pyx_RefNannyDeclarations
14943 int __pyx_t_1;
14944 int __pyx_t_2;
14945 PyObject *__pyx_t_3 = NULL;
14946 Clifford __pyx_t_4;
14947 PyObject *__pyx_t_5 = NULL;
14948 PyObject *__pyx_t_6 = NULL;
14949 PyObject *__pyx_t_7 = NULL;
14950 PyObject *__pyx_t_8 = NULL;
14951 PyObject *__pyx_t_9 = NULL;
14952 PyObject *__pyx_t_10 = NULL;
14953 PyObject *__pyx_t_11 = NULL;
14954 int __pyx_lineno = 0;
14955 const char *__pyx_filename = NULL;
14956 int __pyx_clineno = 0;
14957 __Pyx_RefNannySetupContext("log", 0);
14958 if (__pyx_optional_args) {
14959 if (__pyx_optional_args->__pyx_n > 0) {
14960 __pyx_v_i = __pyx_optional_args->i;
14961 }
14962 }
14963
14964 /* "PyClicl.pyx":1643
14965 * RuntimeError: check_complex(val, i): i is not a valid complexifier for val
14966 *
14967 * if not (i is None):
14968 * return clifford().wrap(glucat.log(toClifford(obj), toClifford(i)))
14969 * else:
14970 */
14971 __pyx_t_1 = (__pyx_v_i != Py_None);
14972 __pyx_t_2 = (__pyx_t_1 != 0);
14973 if (__pyx_t_2) {
14974
14975 /* "PyClicl.pyx":1644
14976 *
14977 * if not (i is None):
14978 * return clifford().wrap(glucat.log(toClifford(obj), toClifford(i)))
14979 * else:
14980 * try:
14981 */
14982 __Pyx_XDECREF(__pyx_r);
14983 __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClicl_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1644, __pyx_L1_error)
14984 __Pyx_GOTREF(__pyx_t_3);
14985 try {
14986 __pyx_t_4 = log(__pyx_f_8PyClicl_toClifford(__pyx_v_obj),
__pyx_f_8PyClicl_toClifford(__pyx_v_i));
14987 } catch (...) {
14988 __Pyx_CppExn2PyErr();
14989 __PYX_ERR(0, 1644, __pyx_L1_error)
14990 }
14991 __pyx_t_5 = __pyx_f_8PyClicl_8clifford_wrap(((struct __pyx_obj_8PyClicl_clifford
*)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1644, __pyx_L1_error)
14992 __Pyx_GOTREF(__pyx_t_5);
14993 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
14994 __pyx_r = __pyx_t_5;
14995 __pyx_t_5 = 0;
14996 goto __pyx_L0;
14997
14998 /* "PyClicl.pyx":1643
14999 * RuntimeError: check_complex(val, i): i is not a valid complexifier for val
15000 *
15001 * if not (i is None):
15002 * return clifford().wrap(glucat.log(toClifford(obj), toClifford(i)))
15003 * else:
15004 */

```

```

15005 }
15006
15007 /* "PyClical.pyx":1646
15008 * return clifford().wrap(glucat.log(toClifford(obj), toClifford(i)))
15009 * else:
15010 * try: # ««««««««
15011 * return math.log(obj)
15012 * except:
15013 */
15014 /*else*/ {
15015 {
15016 __Pyx_PyThreadState_declare
15017 __Pyx_PyThreadState_assign
15018 __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
15019 __Pyx_XGOTREF(__pyx_t_6);
15020 __Pyx_XGOTREF(__pyx_t_7);
15021 __Pyx_XGOTREF(__pyx_t_8);
15022 }/*try*/ {
15023
15024 /* "PyClical.pyx":1647
15025 * else:
15026 * try:
15027 * return math.log(obj) # ««««««««
15028 * except:
15029 * return clifford().wrap(glucat.log(toClifford(obj)))
15030 */
15031 __Pyx_XDECREF(__pyx_r);
15032 __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1647, __pyx_L4_error)
15033 __Pyx_GOTREF(__pyx_t_3);
15034 __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_log); if
(unlikely(!__pyx_t_9)) __PYX_ERR(0, 1647, __pyx_L4_error)
15035 __Pyx_GOTREF(__pyx_t_9);
15036 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
15037 __pyx_t_3 = NULL;
15038 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
15039 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
15040 if (likely(__pyx_t_3)) {
15041 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
15042 __Pyx_INCREF(__pyx_t_3);
15043 __Pyx_INCREF(function);
15044 __Pyx_DECREF_SET(__pyx_t_9, function);
15045 }
15046 }
15047 __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
15048 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
15049 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1647, __pyx_L4_error)
15050 __Pyx_GOTREF(__pyx_t_5);
15051 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
15052 __pyx_r = __pyx_t_5;
15053 __pyx_t_5 = 0;
15054 goto __pyx_L8_try_return;
15055
15056 /* "PyClical.pyx":1646
15057 * return clifford().wrap(glucat.log(toClifford(obj), toClifford(i)))
15058 * else:
15059 * try: # ««««««««
15060 * return math.log(obj)
15061 * except:
15062 */
15063 }
15064 __pyx_L4_error:;
15065 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
15066 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
15067 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
15068
15069 /* "PyClical.pyx":1648
15070 * try:
15071 * return math.log(obj)
15072 * except: # ««««««««
15073 * return clifford().wrap(glucat.log(toClifford(obj)))
15074 *
15075 */
15076 /*except*/ {
15077 __Pyx_AddTraceback("PyClical.log", __pyx_clineno, __pyx_lineno,
__pyx_filename);
15078 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
15079 1648, __pyx_L6_except_error)
15080 __Pyx_GOTREF(__pyx_t_5);
15081 __Pyx_GOTREF(__pyx_t_9);
15082 __Pyx_GOTREF(__pyx_t_3);
15083
15084 /* "PyClical.pyx":1649
15085 * return math.log(obj)
15086 * except:
15087 * return clifford().wrap(glucat.log(toClifford(obj))) # ««««««««

```

```

15087 *
15088 * cpdef inline cos(obj,i = None):
15089 */
15090
15091 __Pyx_XDECREF(__pyx_r);
15092 __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
15093 *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1649,
__pyx_L6_except_error)
15092 __Pyx_GOTREF(__pyx_t_10);
15093 __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct
__pyx_obj_8PyClical_clifford *)__pyx_t_10), log(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 1649, __pyx_L6_except_error)
15094 __Pyx_GOTREF(__pyx_t_11);
15095 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
15096 __pyx_r = __pyx_t_11;
15097 __pyx_t_11 = 0;
15098 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
15099 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
15100 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
15101 goto __pyx_L7_except_return;
15102 }
15103 __pyx_L6_except_error;;
15104
15105 /* "PyClical.pyx":1646
15106 return clifford().wrap(glucat.log(toClifford(obj), toClifford(i)))
15107 else:
15108 try:
15109 # ««««««««
15109 return math.log(obj)
15110 except:
15111 */
15112
15113 __Pyx_XGIVEREF(__pyx_t_6);
15114 __Pyx_XGIVEREF(__pyx_t_7);
15115 __Pyx_XGIVEREF(__pyx_t_8);
15116 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
15117 goto __pyx_L1_error;
15118 __pyx_L8_try_return;;
15119 __Pyx_XGIVEREF(__pyx_t_6);
15120 __Pyx_XGIVEREF(__pyx_t_7);
15121 __Pyx_XGIVEREF(__pyx_t_8);
15122 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
15123 goto __pyx_L0;
15124 __pyx_L7_except_return;;
15125 __Pyx_XGIVEREF(__pyx_t_6);
15126 __Pyx_XGIVEREF(__pyx_t_7);
15127 __Pyx_XGIVEREF(__pyx_t_8);
15128 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
15129 goto __pyx_L0;
15130 }
15131
15132 /* "PyClical.pyx":1628
15133 return clifford().wrap(glucat.exp(toClifford(obj)))
15134
15135 * cpdef inline log(obj,i = None):
15136 * """
15137 * Natural logarithm of multivector with optional complexifier.
15138 */
15139
15140 /* function exit code */
15141 __pyx_L1_error;;
15142 __Pyx_XDECREF(__pyx_t_3);
15143 __Pyx_XDECREF(__pyx_t_5);
15144 __Pyx_XDECREF(__pyx_t_9);
15145 __Pyx_XDECREF(__pyx_t_10);
15146 __Pyx_XDECREF(__pyx_t_11);
15147 __Pyx_AddTraceback("PyClical.log", __pyx_clineno, __pyx_lineno, __pyx_filename);
15148 __pyx_r = 0;
15149 __pyx_L0;;
15150 __Pyx_XGIVEREF(__pyx_r);
15151 __Pyx_RefNannyFinishContext();
15152 return __pyx_r;
15153 }
15154
15155 /* Python wrapper */
15156 static PyObject *__pyx_pw_8PyClical_55log(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kws); /*proto*/
15157 static char __pyx_doc_8PyClical_54log[] = "\n Natural logarithm of multivector with
optional complexifier.\n\n >> x=clifford(\"{-1}\"); print((log(x,\"{-1}\") * 2/pi))\n {-1}\n
>> x=clifford(\"{1,2}\"); print((log(x,\"{1,2,3}\") * 2/pi))\n {1,2}\n >> x=clifford(\"{1,2}\");
print((log(x) * 2/pi))\n {1,2}\n >> x=clifford(\"{1,2}\"); print((log(x,\"{1,2}\") * 2/pi))\n
Traceback (most recent call last):\n ...\n RuntimeError: check_complex(val, i): i is not a valid
complexifier for val\n ";
15158 static PyObject *__pyx_pw_8PyClical_55log(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kws) {
15159 PyObject *__pyx_v_obj = 0;
15160 PyObject *__pyx_v_i = 0;
15161 int __pyx_lineno = 0;
15162 const char *__pyx_filename = NULL;

```



```

15163 int __pyx_clineno = 0;
15164 PyObject *__pyx_r = 0;
15165 __Pyx_RefNannyDeclarations
15166 __Pyx_RefNannySetupContext("log (wrapper)", 0);
15167 {
15168 static PyObject *__pyx_pyargnames[] = {&__pyx_n_s_obj, &__pyx_n_s_i, 0};
15169 PyObject* values[2] = {0, 0};
15170 values[1] = ((PyObject *)Py_None);
15171 if (unlikely(__pyx_kwds)) {
15172 Py_ssize_t kw_args;
15173 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
15174 switch (pos_args) {
15175 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
15176 CYTHON_FALLTHROUGH;
15177 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
15178 CYTHON_FALLTHROUGH;
15179 case 0: break;
15180 default: goto __pyx_L5_argtuple_error;
15181 }
15182 kw_args = PyDict_Size(__pyx_kwds);
15183 switch (pos_args) {
15184 case 0:
15185 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
0)) kw_args--;
15186 else goto __pyx_L5_argtuple_error;
15187 CYTHON_FALLTHROUGH;
15188 case 1:
15189 if (kw_args > 0) {
15190 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
15191 if (value) { values[1] = value; kw_args--; }
15192 }
15193 }
15194 if (unlikely(kw_args > 0)) {
15195 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "log") < 0)) __PYX_ERR(0, 1628, __pyx_L3_error)
15196 }
15197 else {
15198 switch (PyTuple_GET_SIZE(__pyx_args)) {
15199 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
15200 CYTHON_FALLTHROUGH;
15201 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
15202 break;
15203 default: goto __pyx_L5_argtuple_error;
15204 }
15205 }
15206 __pyx_v_obj = values[0];
15207 __pyx_v_i = values[1];
15208 }
15209 goto __pyx_L4_argument_unpacking_done;
15210 __pyx_L5_argtuple_error:;
15211 __Pyx_RaiseArgtupleInvalid("log", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
15212 __PYX_ERR(0, 1628, __pyx_L3_error)
15213 __pyx_L3_error:;
15214 __Pyx_AddTraceback("PyClical.log", __pyx_clineno, __pyx_lineno, __pyx_filename);
15215 __Pyx_RefNannyFinishContext();
15216 return NULL;
15217 __pyx_L4_argument_unpacking_done:;
15218 __pyx_r = __pyx_pf_8PyClical_54log(__pyx_self, __pyx_v_obj, __pyx_v_i);
15219
15220 /* function exit code */
15221 __Pyx_RefNannyFinishContext();
15222 return __pyx_r;
15223 }
15224 static PyObject *__pyx_pf_8PyClical_54log(CYTHON_UNUSED PyObject *__pyx_self, PyObject
*__pyx_v_obj, PyObject *__pyx_v_i) {
15225 PyObject *__pyx_r = NULL;
15226 __Pyx_RefNannyDeclarations
15227 PyObject *__pyx_t_1 = NULL;
15228 struct __pyx_opt_args_8PyClical_log __pyx_t_2;
15229 int __pyx_lineno = 0;
15230 const char *__pyx_filename = NULL;
15231 int __pyx_clineno = 0;
15232 __Pyx_RefNannySetupContext("log", 0);
15233 __Pyx_XDECREF(__pyx_r);
15234 __pyx_t_2.__pyx_n = 1;
15235 __pyx_t_2.i = __pyx_v_i;
15236 __pyx_t_1 = __pyx_pf_8PyClical_log(__pyx_v_obj, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1628, __pyx_L1_error)
15237 __Pyx_GOTREF(__pyx_t_1);
15238 __pyx_r = __pyx_t_1;
15239 __pyx_t_1 = 0;
15240 goto __pyx_L0;
15241
15242 /* function exit code */
15243 __pyx_L1_error:;
15244 __Pyx_XDECREF(__pyx_t_1);

```

```

15245 __Pyx_AddTraceback("PyCliclcal.log", __pyx_clineno, __pyx_lineno, __pyx_filename);
15246 __pyx_r = NULL;
15247 __pyx_L0;
15248 __Pyx_XGIVEREF(__pyx_r);
15249 __Pyx_RefNannyFinishContext();
15250 return __pyx_r;
15251 }
15252
15253 /* "PyCliclcal.pyx":1651
15254 *
15255 * return clifford().wrap(glucat.log(toClifford(obj)))
15256 *
15257 * cpdef inline cos(obj,i = None):
15258 * # ««««««««
15259 * """
15260 * Cosine of multivector with optional complexifier.
15261 */
15262
15263 static PyObject *__pyx_pw_8PyCliclcal_57cos(PyObject *__pyx_self, PyObject *__pyx_args,
15264 PyObject *__pyx_kwds); /*proto*/
15265 static CYTHON_INLINE PyObject *__pyx_f_8PyCliclcal_cos(PyObject *__pyx_v_obj,
15266 CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyCliclcal_cos *__pyx_optional_args) {
15267 PyObject *__pyx_v_i = ((PyObject *)Py_None);
15268 PyObject *__pyx_r = NULL;
15269 __Pyx_RefNannyDeclarations
15270 int __pyx_t_1;
15271 int __pyx_t_2;
15272 PyObject *__pyx_t_3 = NULL;
15273 Clifford __pyx_t_4;
15274 PyObject *__pyx_t_5 = NULL;
15275 PyObject *__pyx_t_6 = NULL;
15276 PyObject *__pyx_t_7 = NULL;
15277 PyObject *__pyx_t_8 = NULL;
15278 PyObject *__pyx_t_9 = NULL;
15279 PyObject *__pyx_t_10 = NULL;
15280 PyObject *__pyx_t_11 = NULL;
15281 int __pyx_lineno = 0;
15282 const char *__pyx_filename = NULL;
15283 int __pyx_clineno = 0;
15284 __Pyx_RefNannySetupContext("cos", 0);
15285 if (__pyx_optional_args) {
15286 if (__pyx_optional_args->__pyx_n > 0) {
15287 __pyx_v_i = __pyx_optional_args->i;
15288 }
15289 }
15290
15291 /* "PyCliclcal.pyx":1660
15292 *
15293 * {1,2}
15294 * """
15295 * if not (i is None):
15296 * # ««««««««
15297 * return clifford().wrap(glucat.cos(toClifford(obj), toClifford(i)))
15298 *
15299 * else:
15300 */
15301 __pyx_t_1 = (__pyx_v_i != Py_None);
15302 __pyx_t_2 = (__pyx_t_1 != 0);
15303 if (__pyx_t_2) {
15304
15305 /* "PyCliclcal.pyx":1661
15306 *
15307 * if not (i is None):
15308 * return clifford().wrap(glucat.cos(toClifford(obj), toClifford(i)))
15309 *
15310 * else:
15311 * try:
15312 */
15313 __Pyx_XDECREF(__pyx_r);
15314 __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject *)
15315 *)__pyx_ptype_8PyCliclcal_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1661, __pyx_L1_error)
15316 __Pyx_GOTREF(__pyx_t_3);
15317 try {
15318 __pyx_t_4 = cos(__pyx_f_8PyCliclcal_toClifford(__pyx_v_obj),
15319 __pyx_f_8PyCliclcal_toClifford(__pyx_v_i));
15320 } catch (...) {
15321 __Pyx_CppExn2PyErr();
15322 __PYX_ERR(0, 1661, __pyx_L1_error)
15323 }
15324 __pyx_t_5 = __pyx_f_8PyCliclcal_8clifford_wrap(((struct __pyx_obj_8PyCliclcal_clifford
15325 *)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1661, __pyx_L1_error)
15326 __Pyx_GOTREF(__pyx_t_5);
15327 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
15328 __pyx_r = __pyx_t_5;
15329 __pyx_t_5 = 0;
15330 goto __pyx_L0;
15331
15332 /* "PyCliclcal.pyx":1660
15333 *
15334 * {1,2}
15335 * """
15336 * if not (i is None):
15337 * # ««««««««
15338 * return clifford().wrap(glucat.cos(toClifford(obj), toClifford(i)))
15339 *
15340 * else:
15341 */

```

```

15327 */
15328 }
15329
15330 /* "PyClical.pyx":1663
15331 * return clifford().wrap(glucat.cos(toClifford(obj), toClifford(i)))
15332 * else:
15333 * try:
15334 * # ««««««««
15335 * return math.cos(obj)
15336 * except:
15337 */
15338 /*else*/ {
15339 {
15340 __Pyx_PyThreadState_declare
15341 __Pyx_PyThreadState_assign
15342 __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
15343 __Pyx_XGOTREF(__pyx_t_6);
15344 __Pyx_XGOTREF(__pyx_t_7);
15345 __Pyx_XGOTREF(__pyx_t_8);
15346 /*try:*/ {
15347
15348 /* "PyClical.pyx":1664
15349 * try:
15350 * return math.cos(obj)
15351 * # ««««««««
15352 * except:
15353 * return clifford().wrap(glucat.cos(toClifford(obj)))
15354 */
15355 __Pyx_XDECREF(__pyx_r);
15356 __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
15357 (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1664, __pyx_L4_error)
15358 __Pyx_GOTREF(__pyx_t_3);
15359 __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_cos); if
15360 (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1664, __pyx_L4_error)
15361 __Pyx_GOTREF(__pyx_t_9);
15362 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
15363 __pyx_t_3 = NULL;
15364 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
15365 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
15366 if (likely(__pyx_t_3)) {
15367 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
15368 __Pyx_INCREF(__pyx_t_3);
15369 __Pyx_INCREF(function);
15370 __Pyx_DECREF_SET(__pyx_t_9, function);
15371 }
15372 }
15373 __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
15374 __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
15375 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
15376 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1664, __pyx_L4_error)
15377 __Pyx_GOTREF(__pyx_t_5);
15378 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
15379 __pyx_r = __pyx_t_5;
15380 __pyx_t_5 = 0;
15381 goto __pyx_L8_try_return;
15382
15383 /* "PyClical.pyx":1663
15384 * return clifford().wrap(glucat.cos(toClifford(obj), toClifford(i)))
15385 * else:
15386 * try:
15387 * # ««««««««
15388 * return math.cos(obj)
15389 * except:
15390 */
15391 }
15392 __pyx_L4_error:;
15393 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
15394 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
15395 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
15396
15397 /* "PyClical.pyx":1665
15398 * try:
15399 * return math.cos(obj)
15400 * except:
15401 * # ««««««««
15402 * return clifford().wrap(glucat.cos(toClifford(obj)))
15403 */
15404 /*except:*/ {
15405 __Pyx_AddTraceback("PyClical.cos", __pyx_clineno, __pyx_lineno,
15406 __pyx_filename);
15407 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
15408 1665, __pyx_L6_except_error)
15409 __Pyx_GOTREF(__pyx_t_5);
15410 __Pyx_GOTREF(__pyx_t_9);
15411 __Pyx_GOTREF(__pyx_t_3);
15412
15413 /* "PyClical.pyx":1666
15414 * return math.cos(obj)
15415 * except:

```

```

15409 * return clifford().wrap(glucat.cos(toClifford(obj))) # ««««««««
15410 *
15411 * cpdef inline acos(obj,i = None): # ««««««««
15412 */
15413 __Pyx_XDECREF(__pyx_r);
15414 __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1666,
__pyx_L6_except_error)
15415 __Pyx_GOTREF(__pyx_t_10);
15416 __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct
__pyx_obj_8PyClical_clifford *)__pyx_t_10), cos(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 1666, __pyx_L6_except_error)
15417 __Pyx_GOTREF(__pyx_t_11);
15418 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
15419 __pyx_r = __pyx_t_11;
15420 __pyx_t_11 = 0;
15421 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
15422 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
15423 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
15424 goto __pyx_L7_except_return;
15425 }
15426 __pyx_L6_except_error;;
15427
15428 /* "PyClical.pyx":1663
15429 * return clifford().wrap(glucat.cos(toClifford(obj), toClifford(i)))
15430 * else:
15431 * try: # ««««««««
15432 * return math.cos(obj)
15433 * except:
15434 */
15435 __Pyx_XGIVEREF(__pyx_t_6);
15436 __Pyx_XGIVEREF(__pyx_t_7);
15437 __Pyx_XGIVEREF(__pyx_t_8);
15438 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
15439 goto __pyx_L1_error;
15440 __pyx_L8_try_return;;
15441 __Pyx_XGIVEREF(__pyx_t_6);
15442 __Pyx_XGIVEREF(__pyx_t_7);
15443 __Pyx_XGIVEREF(__pyx_t_8);
15444 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
15445 goto __pyx_L0;
15446 __pyx_L7_except_return;;
15447 __Pyx_XGIVEREF(__pyx_t_6);
15448 __Pyx_XGIVEREF(__pyx_t_7);
15449 __Pyx_XGIVEREF(__pyx_t_8);
15450 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
15451 goto __pyx_L0;
15452 }
15453 }
15454
15455 /* "PyClical.pyx":1651
15456 * return clifford().wrap(glucat.log(toClifford(obj)))
15457 *
15458 * cpdef inline cos(obj,i = None): # ««««««««
15459 * """
15460 * Cosine of multivector with optional complexifier.
15461 */
15462
15463 /* function exit code */
15464 __pyx_L1_error;;
15465 __Pyx_XDECREF(__pyx_t_3);
15466 __Pyx_XDECREF(__pyx_t_5);
15467 __Pyx_XDECREF(__pyx_t_9);
15468 __Pyx_XDECREF(__pyx_t_10);
15469 __Pyx_XDECREF(__pyx_t_11);
15470 __Pyx_AddTraceback("PyClical.cos", __pyx_clineno, __pyx_lineno, __pyx_filename);
15471 __pyx_r = 0;
15472 __pyx_L0;;
15473 __Pyx_XGIVEREF(__pyx_r);
15474 __Pyx_RefNannyFinishContext();
15475 return __pyx_r;
15476 }
15477
15478 /* Python wrapper */
15479 static PyObject *__pyx_pw_8PyClical_57cos(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kws); /*proto*/
15480 static char __pyx_doc_8PyClical_56cos[] = "\n Cosine of multivector with optional
complexifier.\n\n >> x=clifford('{1,2}'); print(cos(acos(x), '{1,2,3}')\n {1,2}\n >>
x=clifford('{1,2}'); print(cos(acos(x))\n {1,2}\n ";
15481 static PyObject *__pyx_pw_8PyClical_57cos(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kws) {
15482 PyObject *__pyx_v_obj = 0;
15483 PyObject *__pyx_v_i = 0;
15484 int __pyx_lineno = 0;
15485 const char *__pyx_filename = NULL;
15486 int __pyx_clineno = 0;
15487 PyObject *__pyx_r = 0;

```

```

15488 __Pyx_RefNannyDeclarations
15489 __Pyx_RefNannySetupContext("cos (wrapper)", 0);
15490 {
15491 static PyObject* __pyx_pyargnames[] = {&__pyx_n_s_obj, &__pyx_n_s_i, 0};
15492 PyObject* values[2] = {0, 0};
15493 values[1] = (PyObject*)Py_None;
15494 if (unlikely(__pyx_kwds)) {
15495 Py_ssize_t kw_args;
15496 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
15497 switch (pos_args) {
15498 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
15499 CYTHON_FALLTHROUGH;
15500 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
15501 CYTHON_FALLTHROUGH;
15502 case 0: break;
15503 default: goto __pyx_L5_argtuple_error;
15504 }
15505 kw_args = PyDict_Size(__pyx_kwds);
15506 switch (pos_args) {
15507 case 0:
15508 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
15509 0)) kw_args--;
15509 else goto __pyx_L5_argtuple_error;
15510 CYTHON_FALLTHROUGH;
15511 case 1:
15512 if (kw_args > 0) {
15513 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
15514 if (value) { values[1] = value; kw_args--; }
15515 }
15516 }
15517 if (unlikely(kw_args > 0)) {
15518 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
15519 values, pos_args, "cos") < 0)) __PYX_ERR(0, 1651, __pyx_L3_error)
15520 }
15521 else {
15522 switch (PyTuple_GET_SIZE(__pyx_args)) {
15523 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
15524 CYTHON_FALLTHROUGH;
15525 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
15526 break;
15527 default: goto __pyx_L5_argtuple_error;
15528 }
15529 __pyx_v_obj = values[0];
15530 __pyx_v_i = values[1];
15531 }
15532 goto __pyx_L4_argument_unpacking_done;
15533 __pyx_L5_argtuple_error:;
15534 __Pyx_RaiseArgtupleInvalid("cos", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
15535 __PYX_ERR(0, 1651, __pyx_L3_error)
15536 __pyx_L3_error:;
15537 __Pyx_AddTraceback("PyClical.cos", __pyx_clineno, __pyx_lineno, __pyx_filename);
15538 __Pyx_RefNannyFinishContext();
15539 return NULL;
15540 __pyx_L4_argument_unpacking_done:;
15541 __pyx_r = __pyx_pf_8PyClical_56cos(__pyx_self, __pyx_v_obj, __pyx_v_i);
15542 /* function exit code */
15543 __Pyx_RefNannyFinishContext();
15544 return __pyx_r;
15545 }
15546 }
15547 static PyObject* __pyx_pf_8PyClical_56cos(CYTHON_UNUSED PyObject* __pyx_self, PyObject
__pyx_v_obj, PyObject __pyx_v_i) {
15548 PyObject* __pyx_r = NULL;
15549 __Pyx_RefNannyDeclarations
15550 PyObject* __pyx_t_1 = NULL;
15551 struct __pyx_opt_args_8PyClical_cos __pyx_t_2;
15552 int __pyx_lineno = 0;
15553 const char* __pyx_filename = NULL;
15554 int __pyx_clineno = 0;
15555 __Pyx_RefNannySetupContext("cos", 0);
15556 __Pyx_XDECREF(__pyx_r);
15557 __pyx_t_2.__pyx_n = 1;
15558 __pyx_t_2.i = __pyx_v_i;
15559 __pyx_t_1 = __pyx_f_8PyClical_cos(__pyx_v_obj, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1651, __pyx_L1_error)
15560 __Pyx_GOTREF(__pyx_t_1);
15561 __pyx_r = __pyx_t_1;
15562 __pyx_t_1 = 0;
15563 goto __pyx_L0;
15564 }
15565 /* function exit code */
15566 __pyx_L1_error:;
15567 __Pyx_XDECREF(__pyx_t_1);
15568 __Pyx_AddTraceback("PyClical.cos", __pyx_clineno, __pyx_lineno, __pyx_filename);
15569 __pyx_r = NULL;

```

```

15570 __pyx_L0;
15571 __Pyx_XGIVEREF(__pyx_r);
15572 __Pyx_RefNannyFinishContext();
15573 return __pyx_r;
15574 }
15575
15576 /* "PyClicl.pyx":1668
15577 * return clifford().wrap(glucat.cos(toClifford(obj)))
15578 *
15579 * cpdef inline acos(obj,i = None):
15580 * """
15581 * Inverse cosine of multivector with optional complexifier.
15582 */
15583
15584 static PyObject * __pyx_pw_8PyClicl_59acos(PyObject * __pyx_self, PyObject * __pyx_args,
PyObject * __pyx_kwds); /*proto*/
15585 static CYTHON_INLINE PyObject * __pyx_f_8PyClicl_acos(PyObject * __pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClicl_acos * __pyx_optional_args) {
15586 PyObject * __pyx_v_i = (PyObject *)Py_None;
15587 PyObject * __pyx_r = NULL;
15588 __Pyx_RefNannyDeclarations
15589 int __pyx_t_1;
15590 int __pyx_t_2;
15591 PyObject * __pyx_t_3 = NULL;
15592 Clifford __pyx_t_4;
15593 PyObject * __pyx_t_5 = NULL;
15594 PyObject * __pyx_t_6 = NULL;
15595 PyObject * __pyx_t_7 = NULL;
15596 PyObject * __pyx_t_8 = NULL;
15597 PyObject * __pyx_t_9 = NULL;
15598 PyObject * __pyx_t_10 = NULL;
15599 PyObject * __pyx_t_11 = NULL;
15600 int __pyx_lineno = 0;
15601 const char * __pyx_filename = NULL;
15602 int __pyx_clineno = 0;
15603 __Pyx_RefNannySetupContext("acos", 0);
15604 if (__pyx_optional_args) {
15605 if (__pyx_optional_args->__pyx_n > 0) {
15606 __pyx_v_i = __pyx_optional_args->i;
15607 }
15608 }
15609
15610 /* "PyClicl.pyx":1681
15611 * {1,2}
15612 * """
15613 * if not (i is None):
15614 * return clifford().wrap(glucat.acos(toClifford(obj), toClifford(i)))
15615 * else:
15616 */
15617 __pyx_t_1 = (__pyx_v_i != Py_None);
15618 __pyx_t_2 = (__pyx_t_1 != 0);
15619 if (__pyx_t_2) {
15620
15621 /* "PyClicl.pyx":1682
15622 * """
15623 * if not (i is None):
15624 * return clifford().wrap(glucat.acos(toClifford(obj), toClifford(i)))
15625 * else:
15626 * try:
15627 */
15628 __Pyx_XDECREF(__pyx_r);
15629 __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClicl_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1682, __pyx_L1_error)
15630 __Pyx_GOTREF(__pyx_t_3);
15631 try {
15632 __pyx_t_4 = acos(__pyx_f_8PyClicl_toClifford(__pyx_v_obj),
__pyx_f_8PyClicl_toClifford(__pyx_v_i));
15633 } catch (...) {
15634 __Pyx_CppExn2PyErr();
15635 __PYX_ERR(0, 1682, __pyx_L1_error)
15636 }
15637 __pyx_t_5 = __pyx_f_8PyClicl_8clifford_wrap(((struct __pyx_obj_8PyClicl_clifford
*)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1682, __pyx_L1_error)
15638 __Pyx_GOTREF(__pyx_t_5);
15639 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
15640 __pyx_r = __pyx_t_5;
15641 __pyx_t_5 = 0;
15642 goto __pyx_L0;
15643
15644 /* "PyClicl.pyx":1681
15645 * {1,2}
15646 * """
15647 * if not (i is None):
15648 * return clifford().wrap(glucat.acos(toClifford(obj), toClifford(i)))
15649 * else:
15650 */
15651 }

```

```

15652
15653 /* "PyClical.pyx":1684
15654 * return clifford().wrap(glucat.acos(toClifford(obj), toClifford(i)))
15655 * else:
15656 * try:
15657 * # ««««««««
15658 * return math.acos(obj)
15659 * except:
15660 */
15661 /*else*/ {
15662 {
15663 __Pyx_PyThreadState_declare
15664 __Pyx_PyThreadState_assign
15665 __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
15666 __Pyx_XGOTREF(__pyx_t_6);
15667 __Pyx_XGOTREF(__pyx_t_7);
15668 __Pyx_XGOTREF(__pyx_t_8);
15669 /*try:*/ {
15670
15671 /* "PyClical.pyx":1685
15672 * else:
15673 * try:
15674 * return math.acos(obj)
15675 * # ««««««««
15676 * except:
15677 * return clifford().wrap(glucat.acos(toClifford(obj)))
15678 */
15679 __Pyx_XDECREF(__pyx_r);
15680 __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
15681 (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1685, __pyx_L4_error)
15682 __Pyx_GOTREF(__pyx_t_3);
15683 __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_acos); if
15684 (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1685, __pyx_L4_error)
15685 __Pyx_GOTREF(__pyx_t_9);
15686 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
15687 __pyx_t_3 = NULL;
15688 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
15689 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
15690 if (likely(__pyx_t_3)) {
15691 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
15692 __Pyx_INCREF(__pyx_t_3);
15693 __Pyx_INCREF(function);
15694 __Pyx_DECREF_SET(__pyx_t_9, function);
15695 }
15696 }
15697 __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
15698 __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
15699 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
15700 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1685, __pyx_L4_error)
15701 __Pyx_GOTREF(__pyx_t_5);
15702 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
15703 __pyx_r = __pyx_t_5;
15704 __pyx_t_5 = 0;
15705 goto __pyx_L8_try_return;
15706
15707 /* "PyClical.pyx":1684
15708 * return clifford().wrap(glucat.acos(toClifford(obj), toClifford(i)))
15709 * else:
15710 * try:
15711 * # ««««««««
15712 * return math.acos(obj)
15713 * except:
15714 */
15715 }
15716 __pyx_L4_error:;
15717 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
15718 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
15719 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
15720
15721 /* "PyClical.pyx":1686
15722 * try:
15723 * return math.acos(obj)
15724 * except:
15725 * # ««««««««
15726 * return clifford().wrap(glucat.acos(toClifford(obj)))
15727 */
15728 /*except:*/ {
15729 __Pyx_AddTraceback("PyClical.acos", __pyx_clineno, __pyx_lineno,
15730 __pyx_filename);
15731 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
15732 1686, __pyx_L6_except_error)
15733 __Pyx_GOTREF(__pyx_t_5);
15734 __Pyx_GOTREF(__pyx_t_9);
15735 __Pyx_GOTREF(__pyx_t_3);
15736
15737 /* "PyClical.pyx":1687
15738 * return math.acos(obj)
15739 * except:
15740 * return clifford().wrap(glucat.acos(toClifford(obj)))
15741 * # ««««««««
15742 */

```

```

15734 * cpdef inline cosh(obj):
15735 */
15736 __Pyx_XDECREF(__pyx_r);
15737 __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1687,
__pyx_L6_except_error)
15738 __Pyx_GOTREF(__pyx_t_10);
15739 __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct
__pyx_obj_8PyClical_clifford *)__pyx_t_10), acos(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 1687, __pyx_L6_except_error)
15740 __Pyx_GOTREF(__pyx_t_11);
15741 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
15742 __pyx_r = __pyx_t_11;
15743 __pyx_t_11 = 0;
15744 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
15745 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
15746 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
15747 goto __pyx_L7_except_return;
15748 }
15749 __pyx_L6_except_error;;
15750
15751 /* "PyClical.pyx":1684
15752 * return clifford().wrap(glucat.acos(toClifford(obj), toClifford(i)))
15753 * else:
15754 * try: # ««««««
15755 * return math.acos(obj)
15756 * except:
15757 */
15758 __Pyx_XGIVEREF(__pyx_t_6);
15759 __Pyx_XGIVEREF(__pyx_t_7);
15760 __Pyx_XGIVEREF(__pyx_t_8);
15761 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
15762 goto __pyx_L1_error;
15763 __pyx_L8_try_return;;
15764 __Pyx_XGIVEREF(__pyx_t_6);
15765 __Pyx_XGIVEREF(__pyx_t_7);
15766 __Pyx_XGIVEREF(__pyx_t_8);
15767 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
15768 goto __pyx_L0;
15769 __pyx_L7_except_return;;
15770 __Pyx_XGIVEREF(__pyx_t_6);
15771 __Pyx_XGIVEREF(__pyx_t_7);
15772 __Pyx_XGIVEREF(__pyx_t_8);
15773 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
15774 goto __pyx_L0;
15775 }
15776 }
15777
15778 /* "PyClical.pyx":1668
15779 * return clifford().wrap(glucat.cos(toClifford(obj)))
15780 */
15781 * cpdef inline acos(obj, i = None): # ««««««
15782 * """
15783 * Inverse cosine of multivector with optional complexifier.
15784 */
15785
15786 /* function exit code */
15787 __pyx_L1_error;;
15788 __Pyx_XDECREF(__pyx_t_3);
15789 __Pyx_XDECREF(__pyx_t_5);
15790 __Pyx_XDECREF(__pyx_t_9);
15791 __Pyx_XDECREF(__pyx_t_10);
15792 __Pyx_XDECREF(__pyx_t_11);
15793 __Pyx_AddTraceback("PyClical.acos", __pyx_clineno, __pyx_lineno, __pyx_filename);
15794 __pyx_r = 0;
15795 __pyx_L0;;
15796 __Pyx_XGIVEREF(__pyx_r);
15797 __Pyx_RefNannyFinishContext();
15798 return __pyx_r;
15799 }
15800
15801 /* Python wrapper */
15802 static PyObject *__pyx_pw_8PyClical_59acos(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kws); /*proto*/
15803 static char __pyx_doc_8PyClical_58acos[] = "\n Inverse cosine of multivector with
optional complexifier.\n\n >> x=clifford(\"{1,2}\"); print(cos(acos(x), \"{1,2,3}\")\n {1,2}\n
>> x=clifford(\"{1,2}\"); print(cos(acos(x), \"{1,2,3,4}\")\n {1,2}\n >> print(acos(0) /
pi)\n 0.5\n >> x=clifford(\"{1,2}\"); print(cos(acos(x)))\n {1,2}\n ";
15804 static PyObject *__pyx_pw_8PyClical_59acos(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kws) {
15805 PyObject *__pyx_v_obj = 0;
15806 PyObject *__pyx_v_i = 0;
15807 int __pyx_lineno = 0;
15808 const char *__pyx_filename = NULL;
15809 int __pyx_clineno = 0;
15810 PyObject *__pyx_r = 0;
15811 __Pyx_RefNannyDeclarations

```



```

15812 __Pyx_RefNannySetupContext("acos (wrapper)", 0);
15813 {
15814 static PyObject* *__pyx_pyargnames[] = {&__pyx_n_s_obj, &__pyx_n_s_i, 0};
15815 PyObject* values[2] = {0, 0};
15816 values[1] = ((PyObject*)Py_None);
15817 if (unlikely(__pyx_kwds)) {
15818 Py_ssize_t kw_args;
15819 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
15820 switch (pos_args) {
15821 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
15822 CYTHON_FALLTHROUGH;
15823 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
15824 CYTHON_FALLTHROUGH;
15825 case 0: break;
15826 default: goto __pyx_L5_argtuple_error;
15827 }
15828 kw_args = PyDict_Size(__pyx_kwds);
15829 switch (pos_args) {
15830 case 0:
15831 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
15832 0)) kw_args--;
15833 else goto __pyx_L5_argtuple_error;
15834 CYTHON_FALLTHROUGH;
15835 case 1:
15836 if (kw_args > 0) {
15837 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
15838 if (value) { values[1] = value; kw_args--; }
15839 }
15840 if (unlikely(kw_args > 0)) {
15841 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
15842 values, pos_args, "acos") < 0)) __PYX_ERR(0, 1668, __pyx_L3_error)
15843 }
15844 } else {
15845 switch (PyTuple_GET_SIZE(__pyx_args)) {
15846 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
15847 CYTHON_FALLTHROUGH;
15848 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
15849 break;
15850 default: goto __pyx_L5_argtuple_error;
15851 }
15852 }
15853 __pyx_v_obj = values[0];
15854 __pyx_v_i = values[1];
15855 }
15856 goto __pyx_L4_argument_unpacking_done;
15857 __pyx_L5_argtuple_error:;
15858 __Pyx_RaiseArgtupleInvalid("acos", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
15859 __PYX_ERR(0, 1668, __pyx_L3_error)
15860 __pyx_L3_error:;
15861 __Pyx_AddTraceback("PyClical.acos", __pyx_clineno, __pyx_lineno, __pyx_filename);
15862 __Pyx_RefNannyFinishContext();
15863 return NULL;
15864 __pyx_L4_argument_unpacking_done:;
15865 __pyx_r = __pyx_pf_8PyClical_58acos(__pyx_self, __pyx_v_obj, __pyx_v_i);
15866
15867 /* function exit code */
15868 __Pyx_RefNannyFinishContext();
15869 return __pyx_r;
15870 }
15871
15872 static PyObject* __pyx_pf_8PyClical_58acos(CYTHON_UNUSED PyObject* __pyx_self,
15873 PyObject* __pyx_v_obj, PyObject* __pyx_v_i) {
15874 PyObject* __pyx_r = NULL;
15875 __Pyx_RefNannyDeclarations
15876 PyObject* __pyx_t_1 = NULL;
15877 struct __pyx_opt_args_8PyClical_acos __pyx_t_2;
15878 int __pyx_lineno = 0;
15879 const char *__pyx_filename = NULL;
15880 int __pyx_clineno = 0;
15881 __Pyx_RefNannySetupContext("acos", 0);
15882 __Pyx_XDECREF(__pyx_r);
15883 __pyx_t_2.__pyx_n = 1;
15884 __pyx_t_2.i = __pyx_v_i;
15885 __pyx_t_1 = __pyx_pf_8PyClical_acos(__pyx_v_obj, 0, &__pyx_t_2); if
15886 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1668, __pyx_L1_error)
15887 __Pyx_GOTREF(__pyx_t_1);
15888 __pyx_r = __pyx_t_1;
15889 __pyx_t_1 = 0;
15890 goto __pyx_L0;
15891
15892 /* function exit code */
15893 __pyx_L1_error:;
15894 __Pyx_XDECREF(__pyx_t_1);
15895 __Pyx_AddTraceback("PyClical.acos", __pyx_clineno, __pyx_lineno, __pyx_filename);
15896 __pyx_r = NULL;
15897 __pyx_L0:;

```

```

15894 __Pyx_XGIVEREF(__pyx_r);
15895 __Pyx_RefNannyFinishContext();
15896 return __pyx_r;
15897 }
15898
15899 /* "PyClical.pyx":1689
15900 * return clifford().wrap(glucat.acos(toClifford(obj)))
15901 *
15902 * cpdef inline cosh(obj): # ««««««««
15903 * """
15904 * Hyperbolic cosine of multivector.
15905 */
15906
15907 static PyObject *__pyx_pw_8PyClical_61cosh(PyObject *__pyx_self, PyObject
15908 *__pyx_v_obj); /*proto*/
15909 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_cosh(PyObject *__pyx_v_obj,
15910 CYTHON_UNUSED int __pyx_skip_dispatch) {
15911 PyObject *__pyx_r = NULL;
15912 __Pyx_RefNannyDeclarations
15913 PyObject *__pyx_t_1 = NULL;
15914 PyObject *__pyx_t_2 = NULL;
15915 PyObject *__pyx_t_3 = NULL;
15916 PyObject *__pyx_t_4 = NULL;
15917 PyObject *__pyx_t_5 = NULL;
15918 PyObject *__pyx_t_6 = NULL;
15919 PyObject *__pyx_t_7 = NULL;
15920 PyObject *__pyx_t_8 = NULL;
15921 int __pyx_lineno = 0;
15922 const char *__pyx_filename = NULL;
15923 int __pyx_clineno = 0;
15924 __Pyx_RefNannySetupContext("cosh", 0);
15925
15926 /* "PyClical.pyx":1700
15927 * {1,2}
15928 * """
15929 * try: # ««««««««
15930 * return math.cosh(obj)
15931 * except:
15932 */
15933 {
15934 __Pyx_PyThreadState_declare
15935 __Pyx_PyThreadState_assign
15936 __Pyx_ExceptionSave(&__pyx_t_1, &__pyx_t_2, &__pyx_t_3);
15937 __Pyx_XGOTREF(__pyx_t_1);
15938 __Pyx_XGOTREF(__pyx_t_2);
15939 __Pyx_XGOTREF(__pyx_t_3);
15940 /*try:*/ {
15941
15942 /* "PyClical.pyx":1701
15943 * """
15944 * try:
15945 * return math.cosh(obj) # ««««««««
15946 * except:
15947 * return clifford().wrap(glucat.cosh(toClifford(obj)))
15948 */
15949
15950 __Pyx_XDECREF(__pyx_r);
15951 __Pyx_GetModuleGlobalName(__pyx_t_5, __pyx_n_s_math); if (unlikely(!__pyx_t_5))
15952 __PYX_ERR(0, 1701, __pyx_L3_error)
15953 __Pyx_GOTREF(__pyx_t_5);
15954 __pyx_t_6 = __Pyx_PyObject_GetAttrStr(__pyx_t_5, __pyx_n_s_cosh); if
15955 (unlikely(!__pyx_t_6)) __PYX_ERR(0, 1701, __pyx_L3_error)
15956 __Pyx_GOTREF(__pyx_t_6);
15957 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
15958 __pyx_t_5 = NULL;
15959 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_6))) {
15960 __pyx_t_5 = PyMethod_GET_SELF(__pyx_t_6);
15961 if (likely(__pyx_t_5)) {
15962 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_6);
15963 __Pyx_INCREF(__pyx_t_5);
15964 __Pyx_DECREF(function);
15965 __Pyx_DECREF_SET(__pyx_t_6, function);
15966 }
15967 }
15968 __pyx_t_4 = (__pyx_t_5) ? __Pyx_PyObject_Call2Args(__pyx_t_6, __pyx_t_5,
15969 __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_6, __pyx_v_obj);
15970 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
15971 if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1701, __pyx_L3_error)
15972 __Pyx_GOTREF(__pyx_t_4);
15973 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
15974 __pyx_r = __pyx_t_4;
15975 __pyx_t_4 = 0;
15976 goto __pyx_L7_try_return;
15977
15978 /* "PyClical.pyx":1700
15979 * {1,2}
15980 * """
15981 * try: # ««««««««

```

```

15976 * return math.cosh(obj)
15977 * except:
15978 */
15979 }
15980 __pyx_L3_error;;
15981 __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
15982 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
15983 __Pyx_XDECREF(__pyx_t_6); __pyx_t_6 = 0;
15984
15985 /* "PyClical.pyx":1702
15986 * try:
15987 * return math.cosh(obj)
15988 * except: # ««««««««
15989 * return clifford().wrap(glucat.cosh(toClifford(obj)))
15990 *
15991 */
15992
15993 /*except:*/ {
15994 __Pyx_AddTraceback("PyClical.cosh", __pyx_clineno, __pyx_lineno,
__pyx_filename);
15994 if (__Pyx_GetException(&__pyx_t_4, &__pyx_t_6, &__pyx_t_5) < 0) __PYX_ERR(0,
1702, __pyx_L5_except_error)
15995 __Pyx_GOTREF(__pyx_t_4);
15996 __Pyx_GOTREF(__pyx_t_6);
15997 __Pyx_GOTREF(__pyx_t_5);
15998
15999 /* "PyClical.pyx":1703
16000 * return math.cosh(obj)
16001 * except:
16002 * return clifford().wrap(glucat.cosh(toClifford(obj)))
16003 *
16004 * cpdef inline acosh(obj,i = None):
16005 */
16006 __Pyx_XDECREF(__pyx_r);
16007 __pyx_t_7 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 1703,
__pyx_L5_except_error)
16008 __Pyx_GOTREF(__pyx_t_7);
16009 __pyx_t_8 = __pyx_f_8PyClical_8clifford_wrap(((struct
__pyx_obj_8PyClical_clifford *)__pyx_t_7), cosh(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_8)) __PYX_ERR(0, 1703, __pyx_L5_except_error)
16010 __Pyx_GOTREF(__pyx_t_8);
16011 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
16012 __pyx_r = __pyx_t_8;
16013 __pyx_t_8 = 0;
16014 __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
16015 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
16016 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
16017 goto __pyx_L6_except_return;
16018 }
16019 __pyx_L5_except_error;;
16020
16021 /* "PyClical.pyx":1700
16022 * {1,2}
16023 * """
16024 * try: # ««««««««
16025 * return math.cosh(obj)
16026 * except:
16027 */
16028 __Pyx_XGIVEREF(__pyx_t_1);
16029 __Pyx_XGIVEREF(__pyx_t_2);
16030 __Pyx_XGIVEREF(__pyx_t_3);
16031 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
16032 goto __pyx_L1_error;
16033 __pyx_L7_try_return;
16034 __Pyx_XGIVEREF(__pyx_t_1);
16035 __Pyx_XGIVEREF(__pyx_t_2);
16036 __Pyx_XGIVEREF(__pyx_t_3);
16037 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
16038 goto __pyx_L0;
16039 __pyx_L6_except_return;
16040 __Pyx_XGIVEREF(__pyx_t_1);
16041 __Pyx_XGIVEREF(__pyx_t_2);
16042 __Pyx_XGIVEREF(__pyx_t_3);
16043 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
16044 goto __pyx_L0;
16045 }
16046
16047 /* "PyClical.pyx":1689
16048 * return clifford().wrap(glucat.acos(toClifford(obj)))
16049 *
16050 * cpdef inline cosh(obj): # ««««««««
16051 * """
16052 * Hyperbolic cosine of multivector.
16053 */
16054
16055 /* function exit code */
16056 __pyx_L1_error;;

```

```

16057 __Pyx_XDECREF(__pyx_t_4);
16058 __Pyx_XDECREF(__pyx_t_5);
16059 __Pyx_XDECREF(__pyx_t_6);
16060 __Pyx_XDECREF(__pyx_t_7);
16061 __Pyx_XDECREF(__pyx_t_8);
16062 __Pyx_AddTraceback("PyClical.cosh", __pyx_clineno, __pyx_lineno, __pyx_filename);
16063 __pyx_r = 0;
16064 __pyx_L0:;
16065 __Pyx_XGIVEREF(__pyx_r);
16066 __Pyx_RefNannyFinishContext();
16067 return __pyx_r;
16068 }
16069
16070 /* Python wrapper */
16071 static PyObject * __pyx_pw_8PyClical_61cosh(PyObject * __pyx_self, PyObject
*__pyx_v_obj); /*proto*/
16072 static char __pyx_doc_8PyClical_60cosh[] = "\n Hyperbolic cosine of
multivector.\n\n >> x=clifford(\"{1,2}\") * pi; print(cosh(x))\n -1\n >>
x=clifford(\"{1,2,3}\"); print(cosh(acosh(x)))\n {1,2,3}\n >> x=clifford(\"{1,2}\");
print(cosh(acosh(x)))\n {1,2}\n ";
16073 static PyObject * __pyx_pf_8PyClical_61cosh(PyObject * __pyx_self, PyObject
*__pyx_v_obj) {
16074 PyObject * __pyx_r = 0;
16075 __Pyx_RefNannyDeclarations
16076 __Pyx_RefNannySetupContext("cosh (wrapper)", 0);
16077 __pyx_r = __pyx_pf_8PyClical_60cosh(__pyx_self, ((PyObject *) __pyx_v_obj));
16078
16079 /* function exit code */
16080 __Pyx_RefNannyFinishContext();
16081 return __pyx_r;
16082 }
16083
16084 static PyObject * __pyx_pf_8PyClical_60cosh(CYTHON_UNUSED PyObject * __pyx_self,
PyObject * __pyx_v_obj) {
16085 PyObject * __pyx_r = NULL;
16086 __Pyx_RefNannyDeclarations
16087 PyObject * __pyx_t_1 = NULL;
16088 int __pyx_lineno = 0;
16089 const char * __pyx_filename = NULL;
16090 int __pyx_clineno = 0;
16091 __Pyx_RefNannySetupContext("cosh", 0);
16092 __Pyx_XDECREF(__pyx_r);
16093 __pyx_t_1 = __pyx_f_8PyClical_cosh(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1689, __pyx_L1_error)
16094 __Pyx_GOTREF(__pyx_t_1);
16095 __pyx_r = __pyx_t_1;
16096 __pyx_t_1 = 0;
16097 goto __pyx_L0;
16098
16099 /* function exit code */
16100 __pyx_L1_error:;
16101 __Pyx_XDECREF(__pyx_t_1);
16102 __Pyx_AddTraceback("PyClical.cosh", __pyx_clineno, __pyx_lineno, __pyx_filename);
16103 __pyx_r = NULL;
16104 __pyx_L0:;
16105 __Pyx_XGIVEREF(__pyx_r);
16106 __Pyx_RefNannyFinishContext();
16107 return __pyx_r;
16108 }
16109
16110 /* "PyClical.pyx":1705
16111 * return clifford().wrap(glucat.cosh(toClifford(obj)))
16112 *
16113 * cpdef inline acosh(obj,i = None):
16114 * """
16115 * Inverse hyperbolic cosine of multivector with optional complexifier.
16116 */
16117
16118 static PyObject * __pyx_pw_8PyClical_63acosh(PyObject * __pyx_self, PyObject
*__pyx_args, PyObject * __pyx_kwds); /*proto*/
16119 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_acosh(PyObject * __pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_acosh * __pyx_optional_args) {
16120 PyObject * __pyx_v_i = ((PyObject *) Py_None);
16121 PyObject * __pyx_r = NULL;
16122 __Pyx_RefNannyDeclarations
16123 int __pyx_t_1;
16124 int __pyx_t_2;
16125 PyObject * __pyx_t_3 = NULL;
16126 Clifford __pyx_t_4;
16127 PyObject * __pyx_t_5 = NULL;
16128 PyObject * __pyx_t_6 = NULL;
16129 PyObject * __pyx_t_7 = NULL;
16130 PyObject * __pyx_t_8 = NULL;
16131 PyObject * __pyx_t_9 = NULL;
16132 PyObject * __pyx_t_10 = NULL;
16133 PyObject * __pyx_t_11 = NULL;
16134 int __pyx_lineno = 0;

```

```

16135 const char *__pyx_filename = NULL;
16136 int __pyx_clineno = 0;
16137 __Pyx_RefNannySetupContext("acosh", 0);
16138 if (__pyx_optional_args) {
16139 if (__pyx_optional_args->__pyx_n > 0) {
16140 __pyx_v_i = __pyx_optional_args->i;
16141 }
16142 }
16143
16144 /* "PyClical.pyx":1720
16145 * {1,2}
16146 * """
16147 * if not (i is None): # ««««««««
16148 * return clifford().wrap(glucat.acosh(toClifford(obj), toClifford(i)))
16149 * else:
16150 */
16151 __pyx_t_1 = (__pyx_v_i != Py_None);
16152 __pyx_t_2 = (__pyx_t_1 != 0);
16153 if (__pyx_t_2) {
16154
16155 /* "PyClical.pyx":1721
16156 * """
16157 * if not (i is None):
16158 * return clifford().wrap(glucat.acosh(toClifford(obj), toClifford(i))) #
16159 * else:
16160 * try:
16161 */
16162 __Pyx_XDECREF(__pyx_r);
16163 __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
16164 *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1721, __pyx_L1_error)
16165 __Pyx_GOTREF(__pyx_t_3);
16166 try {
16167 __pyx_t_4 = acosh(__pyx_f_8PyClical_toClifford(__pyx_v_obj),
16168 __pyx_f_8PyClical_toClifford(__pyx_v_i));
16169 } catch (...) {
16170 __Pyx_CppExn2PyErr();
16171 __PYX_ERR(0, 1721, __pyx_L1_error)
16172 }
16173 __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
16174 *)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1721, __pyx_L1_error)
16175 __Pyx_GOTREF(__pyx_t_5);
16176 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
16177 __pyx_r = __pyx_t_5;
16178 __pyx_t_5 = 0;
16179 goto __pyx_L0;
16180
16181 /* "PyClical.pyx":1720
16182 * {1,2}
16183 * """
16184 * if not (i is None): # ««««««««
16185 * return clifford().wrap(glucat.acosh(toClifford(obj), toClifford(i)))
16186 * else:
16187 */
16188 }
16189
16190 /* "PyClical.pyx":1723
16191 * return clifford().wrap(glucat.acosh(toClifford(obj), toClifford(i)))
16192 * else:
16193 * try: # ««««««««
16194 * return math.acosh(obj)
16195 * except:
16196 */
16197 /*else*/ {
16198 {
16199 __Pyx_PyThreadState_declare
16200 __Pyx_PyThreadState_assign
16201 __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
16202 __Pyx_XGOTREF(__pyx_t_6);
16203 __Pyx_XGOTREF(__pyx_t_7);
16204 __Pyx_XGOTREF(__pyx_t_8);
16205 }
16206 /*try:*/ {
16207
16208 /* "PyClical.pyx":1724
16209 * else:
16210 * try:
16211 * return math.acosh(obj) # ««««««««
16212 * except:
16213 * return clifford().wrap(glucat.acosh(toClifford(obj)))
16214 */
16215 __Pyx_XDECREF(__pyx_r);
16216 __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
16217 (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1724, __pyx_L4_error)
16218 __Pyx_GOTREF(__pyx_t_3);
16219 __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_acosh); if
16220 (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1724, __pyx_L4_error)
16221 __Pyx_GOTREF(__pyx_t_9);

```

```

16216 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
16217 __pyx_t_3 = NULL;
16218 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
16219 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
16220 if (likely(__pyx_t_3)) {
16221 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
16222 __Pyx_INCREF(__pyx_t_3);
16223 __Pyx_INCREF(function);
16224 __Pyx_DECREF_SET(__pyx_t_9, function);
16225 }
16226 }
16227 __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
16228 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
16229 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1724, __pyx_L4_error)
16230 __Pyx_GOTREF(__pyx_t_5);
16231 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
16232 __pyx_r = __pyx_t_5;
16233 __pyx_t_5 = 0;
16234 goto __pyx_L8_try_return;
16235
16236 /* "PyClicl.pyx":1723
16237 * return clifford().wrap(glucat.acosh(toClifford(obj)), toClifford(i)))
16238 * else:
16239 * try:
16240 * # ««««««««
16241 * return math.acosh(obj)
16242 * except:
16243 */
16244 __pyx_L4_error:;
16245 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
16246 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
16247 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
16248
16249 /* "PyClicl.pyx":1725
16250 * try:
16251 * return math.acosh(obj)
16252 * except:
16253 * # ««««««««
16254 * return clifford().wrap(glucat.acosh(toClifford(obj)))
16255 *
16256 */
16257 /*except:*/ {
 __Pyx_AddTraceback("PyClicl.acosh", __pyx_clineno, __pyx_lineno,
__pyx_filename);
16258 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
1725, __pyx_L6_except_error)
16259 __Pyx_GOTREF(__pyx_t_5);
16260 __Pyx_GOTREF(__pyx_t_9);
16261 __Pyx_GOTREF(__pyx_t_3);
16262
16263 /* "PyClicl.pyx":1726
16264 * return math.acosh(obj)
16265 * except:
16266 * return clifford().wrap(glucat.acosh(toClifford(obj)))
16267 *
16268 * cpdef inline sin(obj,i = None):
16269 */
16270 __Pyx_XDECREF(__pyx_r);
16271 __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClicl_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1726,
__pyx_L6_except_error)
16272 __Pyx_GOTREF(__pyx_t_10);
16273 __pyx_t_11 = __pyx_f_8PyClicl_8clifford_wrap(((struct
__pyx_obj_8PyClicl_clifford *)__pyx_t_10), acosh(__pyx_f_8PyClicl_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 1726, __pyx_L6_except_error)
16274 __Pyx_GOTREF(__pyx_t_11);
16275 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
16276 __pyx_r = __pyx_t_11;
16277 __pyx_t_11 = 0;
16278 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
16279 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
16280 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
16281 goto __pyx_L7_except_return;
16282 }
16283 __pyx_L6_except_error:;
16284
16285 /* "PyClicl.pyx":1723
16286 * return clifford().wrap(glucat.acosh(toClifford(obj)), toClifford(i)))
16287 * else:
16288 * try:
16289 * # ««««««««
16290 * return math.acosh(obj)
16291 * except:
16292 */
16293 __Pyx_XGIVEREF(__pyx_t_6);
16294 __Pyx_XGIVEREF(__pyx_t_7);
16295 __Pyx_XGIVEREF(__pyx_t_8);
 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);

```

```

16296 goto __pyx_L1_error;
16297 __pyx_L8_try_return:;
16298 __Pyx_XGIVEREF(__pyx_t_6);
16299 __Pyx_XGIVEREF(__pyx_t_7);
16300 __Pyx_XGIVEREF(__pyx_t_8);
16301 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
16302 goto __pyx_L0;
16303 __pyx_L7_except_return:;
16304 __Pyx_XGIVEREF(__pyx_t_6);
16305 __Pyx_XGIVEREF(__pyx_t_7);
16306 __Pyx_XGIVEREF(__pyx_t_8);
16307 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
16308 goto __pyx_L0;
16309 }
16310 }
16311
16312 /* "PyClical.pyx":1705
16313 * return clifford().wrap(gluecat.cosh(toClifford(obj)))
16314 *
16315 * cpdef inline acosh(obj,i = None): # ««««««««
16316 * """
16317 * Inverse hyperbolic cosine of multivector with optional complexifier.
16318 */
16319
16320 /* function exit code */
16321 __pyx_L1_error:;
16322 __Pyx_XDECREF(__pyx_t_3);
16323 __Pyx_XDECREF(__pyx_t_5);
16324 __Pyx_XDECREF(__pyx_t_9);
16325 __Pyx_XDECREF(__pyx_t_10);
16326 __Pyx_XDECREF(__pyx_t_11);
16327 __Pyx_AddTraceback("PyClical.acosh", __pyx_clineno, __pyx_lineno, __pyx_filename);
16328 __pyx_r = 0;
16329 __pyx_L0:;
16330 __Pyx_XGIVEREF(__pyx_r);
16331 __Pyx_RefNannyFinishContext();
16332 return __pyx_r;
16333 }
16334
16335 /* Python wrapper */
16336 static PyObject* __pyx_pw_8PyClical_63acosh(PyObject* __pyx_self, PyObject
16337 *__pyx_args, PyObject* __pyx_kwds); /*proto*/
16338 static char __pyx_doc_8PyClical_62acosh[] = "\n Inverse hyperbolic cosine of
16339 multivector with optional complexifier.\n\n >> print(acosh(0, \"{-2,-1,1}\")\n 1.571{-2,-1,1}\n
16340 >> x=clifford(\"{1,2,3}\"); print(cosh(acosh(x, \"{-1,1,2,3,4}\")\n {1,2,3}\n >>
16341 print(acosh(0))\n 1.571{-1}\n >> x=clifford(\"{1,2,3}\"); print(cosh(acosh(x))\n {1,2,3}\n
16342 >> x=clifford(\"{1,2}\"); print(cosh(acosh(x))\n {1,2}\n ";
16343 static PyObject* __pyx_pw_8PyClical_63acosh(PyObject* __pyx_self, PyObject
16344 *__pyx_args, PyObject* __pyx_kwds) {
16345 PyObject* __pyx_v_obj = 0;
16346 PyObject* __pyx_v_i = 0;
16347 int __pyx_lineno = 0;
16348 const char* __pyx_filename = NULL;
16349 int __pyx_clineno = 0;
16350 PyObject* __pyx_r = 0;
16351 __Pyx_RefNannyDeclarations
16352 __Pyx_RefNannySetupContext("acosh (wrapper)", 0);
16353 {
16354 static PyObject* __pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
16355 PyObject* values[2] = {0,0};
16356 values[1] = ((PyObject*)Py_None);
16357 if (unlikely(__pyx_kwds)) {
16358 Py_ssize_t kw_args;
16359 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
16360 switch (pos_args) {
16361 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
16362 CYTHON_FALLTHROUGH;
16363 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
16364 CYTHON_FALLTHROUGH;
16365 case 0: break;
16366 default: goto __pyx_L5_argtuple_error;
16367 }
16368 kw_args = PyDict_Size(__pyx_kwds);
16369 switch (pos_args) {
16370 case 0:
16371 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
16372 0)) kw_args--;
16373 else goto __pyx_L5_argtuple_error;
16374 CYTHON_FALLTHROUGH;
16375 case 1:
16376 if (kw_args > 0) {
16377 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
16378 if (value) { values[1] = value; kw_args--; }
16379 }
16380 }
16381 if (unlikely(kw_args > 0)) {
16382 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,

```

```

values, pos_args, "acosh") < 0)) __PYX_ERR(0, 1705, __pyx_L3_error)
16376 }
16377 } else {
16378 switch (PyTuple_GET_SIZE(__pyx_args)) {
16379 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
16380 CYTHON_FALLTHROUGH;
16381 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
16382 break;
16383 default: goto __pyx_L5_argtuple_error;
16384 }
16385 }
16386 __pyx_v_obj = values[0];
16387 __pyx_v_i = values[1];
16388 }
16389 goto __pyx_L4_argument_unpacking_done;
16390 __pyx_L5_argtuple_error:;
16391 __Pyx_RaiseArgtupleInvalid("acosh", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
__PYX_ERR(0, 1705, __pyx_L3_error)
16392 __pyx_L3_error:;
16393 __Pyx_AddTraceback("PyClicl.acosh", __pyx_clineno, __pyx_lineno, __pyx_filename);
16394 __Pyx_RefNannyFinishContext();
16395 return NULL;
16396 __pyx_L4_argument_unpacking_done:;
16397 __pyx_r = __pyx_pf_8PyClicl_62acosh(__pyx_self, __pyx_v_obj, __pyx_v_i);
16398
16399 /* function exit code */
16400 __Pyx_RefNannyFinishContext();
16401 return __pyx_r;
16402 }
16403
16404 static PyObject *__pyx_pf_8PyClicl_62acosh(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj, PyObject *__pyx_v_i) {
16405 PyObject *__pyx_r = NULL;
16406 __Pyx_RefNannyDeclarations
16407 PyObject *__pyx_t_1 = NULL;
16408 struct __pyx_opt_args_8PyClicl_acosh __pyx_t_2;
16409 int __pyx_lineno = 0;
16410 const char *__pyx_filename = NULL;
16411 int __pyx_clineno = 0;
16412 __Pyx_RefNannySetupContext("acosh", 0);
16413 __Pyx_XDECREF(__pyx_r);
16414 __pyx_t_2.__pyx_n = 1;
16415 __pyx_t_2.i = __pyx_v_i;
16416 __pyx_t_1 = __pyx_f_8PyClicl_acosh(__pyx_v_obj, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1705, __pyx_L1_error)
16417 __Pyx_GOTREF(__pyx_t_1);
16418 __pyx_r = __pyx_t_1;
16419 __pyx_t_1 = 0;
16420 goto __pyx_L0;
16421
16422 /* function exit code */
16423 __pyx_L1_error:;
16424 __Pyx_XDECREF(__pyx_t_1);
16425 __Pyx_AddTraceback("PyClicl.acosh", __pyx_clineno, __pyx_lineno, __pyx_filename);
16426 __pyx_r = NULL;
16427 __pyx_L0:;
16428 __Pyx_XGIVEREF(__pyx_r);
16429 __Pyx_RefNannyFinishContext();
16430 return __pyx_r;
16431 }
16432
16433 /* "PyClicl.pyx":1728
16434 * return clifford().wrap(glucat.acosh(toClifford(obj)))
16435 *
16436 * cpdef inline sin(obj,i = None): # ««««««««
16437 * """
16438 * Sine of multivector with optional complexifier.
16439 */
16440
16441 static PyObject *__pyx_pw_8PyClicl_65sin(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kws); /*proto*/
16442 static CYTHON_INLINE PyObject *__pyx_f_8PyClicl_sin(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClicl_sin *__pyx_optional_args) {
16443 PyObject *__pyx_v_i = ((PyObject *)Py_None);
16444 PyObject *__pyx_r = NULL;
16445 __Pyx_RefNannyDeclarations
16446 int __pyx_t_1;
16447 int __pyx_t_2;
16448 PyObject *__pyx_t_3 = NULL;
16449 Clifford __pyx_t_4;
16450 PyObject *__pyx_t_5 = NULL;
16451 PyObject *__pyx_t_6 = NULL;
16452 PyObject *__pyx_t_7 = NULL;
16453 PyObject *__pyx_t_8 = NULL;
16454 PyObject *__pyx_t_9 = NULL;
16455 PyObject *__pyx_t_10 = NULL;
16456 PyObject *__pyx_t_11 = NULL;

```



```

16457 int __pyx_lineno = 0;
16458 const char *__pyx_filename = NULL;
16459 int __pyx_clineno = 0;
16460 __Pyx_RefNannySetupContext("sin", 0);
16461 if (__pyx_optional_args) {
16462 if (__pyx_optional_args->__pyx_n > 0) {
16463 __pyx_v_i = __pyx_optional_args->i;
16464 }
16465 }
16466
16467 /* "PyClical.pyx":1739
16468 * {1,2,3}
16469 * """
16470 * if not (i is None): # ««««««««
16471 * return clifford().wrap(glucat.sin(toClifford(obj), toClifford(i)))
16472 * else:
16473 */
16474 __pyx_t_1 = (__pyx_v_i != Py_None);
16475 __pyx_t_2 = (__pyx_t_1 != 0);
16476 if (__pyx_t_2) {
16477 /* "PyClical.pyx":1740
16478 * """
16479 * if not (i is None):
16480 * return clifford().wrap(glucat.sin(toClifford(obj), toClifford(i))) # ««««««««
16481 * else:
16482 * try:
16483 */
16484 /*
16485 * __Pyx_XDECREF(__pyx_r);
16486 * __pyx_t_3 = __Pyx_PyObject_CallNoArg((PyObject
16487 *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1740, __pyx_L1_error)
16488 * __Pyx_GOTREF(__pyx_t_3);
16489 * try {
16490 * __pyx_t_4 = sin(__pyx_f_8PyClical_toClifford(__pyx_v_obj),
16491 * __pyx_f_8PyClical_toClifford(__pyx_v_i));
16492 * } catch (...) {
16493 * __Pyx_CppExn2PyErr();
16494 * __PYX_ERR(0, 1740, __pyx_L1_error)
16495 * }
16496 * __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
16497 *)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1740, __pyx_L1_error)
16498 * __Pyx_GOTREF(__pyx_t_5);
16499 * __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
16500 * __pyx_r = __pyx_t_5;
16501 * __pyx_t_5 = 0;
16502 * goto __pyx_L0;
16503 */
16504 /* "PyClical.pyx":1739
16505 * {1,2,3}
16506 * """
16507 * if not (i is None): # ««««««««
16508 * return clifford().wrap(glucat.sin(toClifford(obj), toClifford(i)))
16509 * else:
16510 */
16511 /*
16512 * __Pyx_XDECREF(__pyx_r);
16513 * __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
16514 (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1743, __pyx_L4_error)
16515 * __Pyx_GOTREF(__pyx_t_3);
16516 * __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_sin); if
16517 (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1743, __pyx_L4_error)
16518 * __Pyx_GOTREF(__pyx_t_9);
16519 */
16520 /* "PyClical.pyx":1742
16521 * return clifford().wrap(glucat.sin(toClifford(obj), toClifford(i)))
16522 * else:
16523 * try: # ««««««««
16524 * return math.sin(obj)
16525 * except:
16526 */
16527 /*else*/ {
16528 {
16529 __Pyx_PyThreadState_declare
16530 __Pyx_PyThreadState_assign
16531 __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
16532 __Pyx_XGOTREF(__pyx_t_6);
16533 __Pyx_XGOTREF(__pyx_t_7);
16534 __Pyx_XGOTREF(__pyx_t_8);
16535 }
16536 /*try:*/ {
16537 /* "PyClical.pyx":1743
16538 * else:
16539 * try:
16540 * return math.sin(obj) # ««««««««
16541 * except:
16542 * return clifford().wrap(glucat.sin(toClifford(obj)))
16543 */
16544 /*
16545 * __Pyx_XDECREF(__pyx_r);
16546 * __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
16547 (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1743, __pyx_L4_error)
16548 * __Pyx_GOTREF(__pyx_t_3);
16549 * __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_sin); if
16550 (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1743, __pyx_L4_error)
16551 * __Pyx_GOTREF(__pyx_t_9);
16552 */

```

```

16539 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
16540 __pyx_t_3 = NULL;
16541 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
16542 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
16543 if (likely(__pyx_t_3)) {
16544 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
16545 __Pyx_INCREF(__pyx_t_3);
16546 __Pyx_INCREF(function);
16547 __Pyx_DECREF_SET(__pyx_t_9, function);
16548 }
16549 }
16550 __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
16551 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
16552 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1743, __pyx_L4_error)
16553 __Pyx_GOTREF(__pyx_t_5);
16554 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
16555 __pyx_r = __pyx_t_5;
16556 __pyx_t_5 = 0;
16557 goto __pyx_L8_try_return;
16558
16559 /* "PyClicl.pyx":1742
16560 * return clifford().wrap(glucat.sin(toClifford(obj), toClifford(i)))
16561 * else:
16562 * try:
16563 * # ««««««««
16564 * return math.sin(obj)
16565 * except:
16566 */
16567 __pyx_L4_error:;
16568 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
16569 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
16570 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
16571
16572 /* "PyClicl.pyx":1744
16573 * try:
16574 * return math.sin(obj)
16575 * except:
16576 * # ««««««««
16577 * return clifford().wrap(glucat.sin(toClifford(obj)))
16578 */
16579 /*except:*/ {
16580 __Pyx_AddTraceback("PyClicl.sin", __pyx_clineno, __pyx_lineno,
__pyx_filename);
16581 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
1744, __pyx_L6_except_error)
16582 __Pyx_GOTREF(__pyx_t_5);
16583 __Pyx_GOTREF(__pyx_t_9);
16584 __Pyx_GOTREF(__pyx_t_3);
16585
16586 /* "PyClicl.pyx":1745
16587 * return math.sin(obj)
16588 * except:
16589 * return clifford().wrap(glucat.sin(toClifford(obj)))
16590 * # ««««««««
16591 * cpdef inline asin(obj,i = None):
16592 */
16593 __Pyx_XDECREF(__pyx_r);
16594 __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClicl_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1745,
__pyx_L6_except_error)
16595 __Pyx_GOTREF(__pyx_t_10);
16596 __pyx_t_11 = __pyx_f_8PyClicl_8clifford_wrap(((struct
__pyx_obj_8PyClicl_clifford *)__pyx_t_10), sin(__pyx_f_8PyClicl_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 1745, __pyx_L6_except_error)
16597 __Pyx_GOTREF(__pyx_t_11);
16598 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
16599 __pyx_r = __pyx_t_11;
16600 __pyx_t_11 = 0;
16601 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
16602 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
16603 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
16604 goto __pyx_L7_except_return;
16605 }
16606 __pyx_L6_except_error:;
16607
16608 /* "PyClicl.pyx":1742
16609 * return clifford().wrap(glucat.sin(toClifford(obj), toClifford(i)))
16610 * else:
16611 * try:
16612 * # ««««««««
16613 * return math.sin(obj)
16614 * except:
16615 */
16616 __Pyx_XGIVEREF(__pyx_t_6);
16617 __Pyx_XGIVEREF(__pyx_t_7);
16618 __Pyx_XGIVEREF(__pyx_t_8);
16619 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);

```

```

16619 goto __pyx_L1_error;
16620 __pyx_L8_try_return:;
16621 __Pyx_XGIVEREF(__pyx_t_6);
16622 __Pyx_XGIVEREF(__pyx_t_7);
16623 __Pyx_XGIVEREF(__pyx_t_8);
16624 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
16625 goto __pyx_L0;
16626 __pyx_L7_except_return:;
16627 __Pyx_XGIVEREF(__pyx_t_6);
16628 __Pyx_XGIVEREF(__pyx_t_7);
16629 __Pyx_XGIVEREF(__pyx_t_8);
16630 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
16631 goto __pyx_L0;
16632 }
16633 }
16634
16635 /* "PyClical.pyx":1728
16636 *
16637 *
16638 * cpdef inline sin(obj,i = None):
16639 * """
16640 * Sine of multivector with optional complexifier.
16641 */
16642
16643 /* function exit code */
16644 __pyx_L1_error:;
16645 __Pyx_XDECREF(__pyx_t_3);
16646 __Pyx_XDECREF(__pyx_t_5);
16647 __Pyx_XDECREF(__pyx_t_9);
16648 __Pyx_XDECREF(__pyx_t_10);
16649 __Pyx_XDECREF(__pyx_t_11);
16650 __Pyx_AddTraceback("PyClical.sin", __pyx_clineno, __pyx_lineno, __pyx_filename);
16651 __pyx_r = 0;
16652 __pyx_L0:;
16653 __Pyx_XGIVEREF(__pyx_r);
16654 __Pyx_RefNannyFinishContext();
16655 return __pyx_r;
16656 }
16657
16658 /* Python wrapper */
16659 static PyObject *__pyx_pw_8PyClical_65sin(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds); /*proto*/
16660 static char __pyx_doc_8PyClical_64sin[] = "\n Sine of multivector with optional
complexifier.\n\n >> s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),s))\n {-1}\n >>
s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),\"{-2,-1,1}\"))\n {-1}\n >>
x=clifford(\"{1,2,3}\"); print(asin(sin(x))\n {1,2,3}\n ";
16661 static PyObject *__pyx_pw_8PyClical_65sin(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds) {
16662 PyObject *__pyx_v_obj = 0;
16663 PyObject *__pyx_v_i = 0;
16664 int __pyx_lineno = 0;
16665 const char *__pyx_filename = NULL;
16666 int __pyx_clineno = 0;
16667 PyObject *__pyx_r = 0;
16668 __Pyx_RefNannyDeclarations
16669 __Pyx_RefNannySetupContext("sin (wrapper)", 0);
16670 {
16671 static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
16672 PyObject* values[2] = {0,0};
16673 values[1] = ((PyObject *)Py_None);
16674 if (unlikely(__pyx_kwds)) {
16675 Py_ssize_t kw_args;
16676 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
16677 switch (pos_args) {
16678 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
16679 CYTHON_FALLTHROUGH;
16680 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
16681 CYTHON_FALLTHROUGH;
16682 case 0: break;
16683 default: goto __pyx_L5_argtuple_error;
16684 }
16685 kw_args = PyDict_Size(__pyx_kwds);
16686 switch (pos_args) {
16687 case 0:
16688 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
0)) kw_args--;
16689 else goto __pyx_L5_argtuple_error;
16690 CYTHON_FALLTHROUGH;
16691 case 1:
16692 if (kw_args > 0) {
16693 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
16694 if (value) { values[1] = value; kw_args--; }
16695 }
16696 }
16697 if (unlikely(kw_args > 0)) {
16698 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "sin") < 0)) __PYX_ERR(0, 1728, __pyx_L3_error)

```

```

16699 }
16700 } else {
16701 switch (PyTuple_GET_SIZE(__pyx_args)) {
16702 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
16703 CYTHON_FALLTHROUGH;
16704 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
16705 break;
16706 default: goto __pyx_L5_argtuple_error;
16707 }
16708 }
16709 __pyx_v_obj = values[0];
16710 __pyx_v_i = values[1];
16711 }
16712 goto __pyx_L4_argument_unpacking_done;
16713 __pyx_L5_argtuple_error:;
16714 __Pyx_RaiseArgtupleInvalid("sin", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
16715 __PYX_ERR(0, 1728, __pyx_L3_error)
16716 __pyx_L3_error:;
16717 __Pyx_AddTraceback("PyClical.sin", __pyx_clineno, __pyx_lineno, __pyx_filename);
16718 __Pyx_RefNannyFinishContext();
16719 return NULL;
16720 __pyx_L4_argument_unpacking_done:;
16721 __pyx_r = __pyx_pf_8PyClical_64sin(__pyx_self, __pyx_v_obj, __pyx_v_i);
16722
16723 /* function exit code */
16724 __Pyx_RefNannyFinishContext();
16725 return __pyx_r;
16726 }
16727
16728 static PyObject *__pyx_pf_8PyClical_64sin(CYTHON_UNUSED PyObject *__pyx_self, PyObject
16729 *__pyx_v_obj, PyObject *__pyx_v_i) {
16730 PyObject *__pyx_r = NULL;
16731 __Pyx_RefNannyDeclarations
16732 PyObject *__pyx_t_1 = NULL;
16733 struct __pyx_opt_args_8PyClical_sin __pyx_t_2;
16734 int __pyx_lineno = 0;
16735 const char *__pyx_filename = NULL;
16736 int __pyx_clineno = 0;
16737 __Pyx_RefNannySetupContext("sin", 0);
16738 __Pyx_XDECREF(__pyx_r);
16739 __pyx_t_2.__pyx_n = 1;
16740 __pyx_t_2.i = __pyx_v_i;
16741 __pyx_t_1 = __pyx_pf_8PyClical_sin(__pyx_v_obj, 0, &__pyx_t_2); if
16742 (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1728, __pyx_L1_error)
16743 __Pyx_GOTREF(__pyx_t_1);
16744 __pyx_r = __pyx_t_1;
16745 __pyx_t_1 = 0;
16746 goto __pyx_L0;
16747
16748 /* function exit code */
16749 __pyx_L1_error:;
16750 __Pyx_XDECREF(__pyx_t_1);
16751 __Pyx_AddTraceback("PyClical.sin", __pyx_clineno, __pyx_lineno, __pyx_filename);
16752 __pyx_r = NULL;
16753 __pyx_L0:;
16754 __Pyx_XGIVEREF(__pyx_r);
16755 __Pyx_RefNannyFinishContext();
16756 return __pyx_r;
16757 }
16758
16759 /* "PyClical.pyx":1747
16760 *
16761 * cpdef inline asin(obj,i = None):
16762 * """
16763 * Inverse sine of multivector with optional complexifier.
16764 *
16765 static PyObject *__pyx_pw_8PyClical_67asin(PyObject *__pyx_self, PyObject *__pyx_args,
16766 PyObject *__pyx_kws); /*proto*/
16767 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_asin(PyObject *__pyx_v_obj,
16768 CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_asin *__pyx_optional_args) {
16769 PyObject *__pyx_v_i = (PyObject *)Py_None;
16770 PyObject *__pyx_r = NULL;
16771 __Pyx_RefNannyDeclarations
16772 int __pyx_t_1;
16773 int __pyx_t_2;
16774 PyObject *__pyx_t_3 = NULL;
16775 Clifford __pyx_t_4;
16776 PyObject *__pyx_t_5 = NULL;
16777 PyObject *__pyx_t_6 = NULL;
16778 PyObject *__pyx_t_7 = NULL;
16779 PyObject *__pyx_t_8 = NULL;
16780 PyObject *__pyx_t_9 = NULL;
16781 PyObject *__pyx_t_10 = NULL;
16782 PyObject *__pyx_t_11 = NULL;
16783 int __pyx_lineno = 0;

```

```

16781 const char *__pyx_filename = NULL;
16782 int __pyx_clineno = 0;
16783 __Pyx_RefNannySetupContext("asin", 0);
16784 if (__pyx_optional_args) {
16785 if (__pyx_optional_args->__pyx_n > 0) {
16786 __pyx_v_i = __pyx_optional_args->i;
16787 }
16788 }
16789
16790 /* "PyClical.pyx":1760
16791 * {1,2,3}
16792 * """
16793 * if not (i is None): # ««««««««
16794 * return clifford().wrap(glucat.asin(toClifford(obj), toClifford(i)))
16795 * else:
16796 */
16797 __pyx_t_1 = (__pyx_v_i != Py_None);
16798 __pyx_t_2 = (__pyx_t_1 != 0);
16799 if (__pyx_t_2) {
16800
16801 /* "PyClical.pyx":1761
16802 * """
16803 * if not (i is None):
16804 * return clifford().wrap(glucat.asin(toClifford(obj), toClifford(i))) # ««««««««
16805 * else:
16806 * try:
16807 */
16808 __Pyx_XDECREF(__pyx_r);
16809 __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1761, __pyx_L1_error)
16810 __Pyx_GOTREF(__pyx_t_3);
16811 try {
16812 __pyx_t_4 = asin(__pyx_f_8PyClical_toClifford(__pyx_v_obj),
__pyx_f_8PyClical_toClifford(__pyx_v_i));
16813 } catch (...) {
16814 __Pyx_CppExn2PyErr();
16815 __PYX_ERR(0, 1761, __pyx_L1_error)
16816 }
16817 __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1761, __pyx_L1_error)
16818 __Pyx_GOTREF(__pyx_t_5);
16819 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
16820 __pyx_r = __pyx_t_5;
16821 __pyx_t_5 = 0;
16822 goto __pyx_L0;
16823
16824 /* "PyClical.pyx":1760
16825 * {1,2,3}
16826 * """
16827 * if not (i is None): # ««««««««
16828 * return clifford().wrap(glucat.asin(toClifford(obj), toClifford(i)))
16829 * else:
16830 */
16831 }
16832
16833 /* "PyClical.pyx":1763
16834 * return clifford().wrap(glucat.asin(toClifford(obj), toClifford(i)))
16835 * else:
16836 * try: # ««««««««
16837 * return math.asin(obj)
16838 * except:
16839 */
16840 /*else*/ {
16841 {
16842 __Pyx_PyThreadState_declare
16843 __Pyx_PyThreadState_assign
16844 __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
16845 __Pyx_XGOTREF(__pyx_t_6);
16846 __Pyx_XGOTREF(__pyx_t_7);
16847 __Pyx_XGOTREF(__pyx_t_8);
16848 /*try:*/ {
16849
16850 /* "PyClical.pyx":1764
16851 * else:
16852 * try:
16853 * return math.asin(obj) # ««««««««
16854 * except:
16855 * return clifford().wrap(glucat.asin(toClifford(obj)))
16856 */
16857 __Pyx_XDECREF(__pyx_r);
16858 __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1764, __pyx_L4_error)
16859 __Pyx_GOTREF(__pyx_t_3);
16860 __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_asin); if
(unlikely(!__pyx_t_9)) __PYX_ERR(0, 1764, __pyx_L4_error)
16861 __Pyx_GOTREF(__pyx_t_9);
16862 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;

```

```

16863 __pyx_t_3 = NULL;
16864 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
16865 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
16866 if (likely(__pyx_t_3)) {
16867 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
16868 __Pyx_INCREF(__pyx_t_3);
16869 __Pyx_INCREF(function);
16870 __Pyx_DECREF_SET(__pyx_t_9, function);
16871 }
16872 }
16873 __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
16874 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
16875 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1764, __pyx_L4_error)
16876 __Pyx_GOTREF(__pyx_t_5);
16877 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
16878 __pyx_r = __pyx_t_5;
16879 __pyx_t_5 = 0;
16880 goto __pyx_L8_try_return;
16881
16882 /* "PyClicl.pyx":1763
16883 * return clifford().wrap(glucat.asin(toClifford(obj)), toClifford(i)))
16884 * else:
16885 * try:
16886 * # ««««««««
16887 * return math.asin(obj)
16888 * except:
16889 */
16890 }
16891 __pyx_L4_error:;
16892 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
16893 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
16894 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
16895
16896 /* "PyClicl.pyx":1765
16897 * try:
16898 * return math.asin(obj)
16899 * except:
16900 * # ««««««««
16901 * return clifford().wrap(glucat.asin(toClifford(obj)))
16902 */
16903 /*except:*/ {
16904 __Pyx_AddTraceback("PyClicl.asin", __pyx_clineno, __pyx_lineno,
__pyx_filename);
16905 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
1765, __pyx_L6_except_error)
16906 __Pyx_GOTREF(__pyx_t_5);
16907 __Pyx_GOTREF(__pyx_t_9);
16908 __Pyx_GOTREF(__pyx_t_3);
16909
16910 /* "PyClicl.pyx":1766
16911 * return math.asin(obj)
16912 * except:
16913 * return clifford().wrap(glucat.asin(toClifford(obj)))
16914 * # ««««««««
16915 * cpdef inline sinh(obj):
16916 */
16917 __Pyx_XDECREF(__pyx_r);
16918 __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClicl_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1766,
__pyx_L6_except_error)
16919 __Pyx_GOTREF(__pyx_t_10);
16920 __pyx_t_11 = __pyx_f_8PyClicl_8clifford_wrap(((struct
__pyx_obj_8PyClicl_clifford *)__pyx_t_10), asin(__pyx_f_8PyClicl_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 1766, __pyx_L6_except_error)
16921 __Pyx_GOTREF(__pyx_t_11);
16922 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
16923 __pyx_r = __pyx_t_11;
16924 __pyx_t_11 = 0;
16925 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
16926 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
16927 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
16928 goto __pyx_L7_except_return;
16929 }
16930 __pyx_L6_except_error:;
16931
16932 /* "PyClicl.pyx":1763
16933 * return clifford().wrap(glucat.asin(toClifford(obj)), toClifford(i)))
16934 * else:
16935 * try:
16936 * # ««««««««
16937 * return math.asin(obj)
16938 * except:
16939 */
16940 __Pyx_XGIVEREF(__pyx_t_6);
16941 __Pyx_XGIVEREF(__pyx_t_7);
16942 __Pyx_XGIVEREF(__pyx_t_8);
16943 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
16944 goto __pyx_L1_error;

```

```

16943 __pyx_L8_try_return;;
16944 __Pyx_XGIVEREF(__pyx_t_6);
16945 __Pyx_XGIVEREF(__pyx_t_7);
16946 __Pyx_XGIVEREF(__pyx_t_8);
16947 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
16948 goto __pyx_L0;
16949 __pyx_L7_except_return;;
16950 __Pyx_XGIVEREF(__pyx_t_6);
16951 __Pyx_XGIVEREF(__pyx_t_7);
16952 __Pyx_XGIVEREF(__pyx_t_8);
16953 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
16954 goto __pyx_L0;
16955 }
16956 }
16957
16958 /* "PyClical.pyx":1747
16959 * return clifford().wrap(glucat.sin(toClifford(obj)))
16960 *
16961 * cpdef inline asin(obj,i = None): # ««««««««
16962 * """
16963 * Inverse sine of multivector with optional complexifier.
16964 */
16965
16966 /* function exit code */
16967 __pyx_L1_error;;
16968 __Pyx_XDECREF(__pyx_t_3);
16969 __Pyx_XDECREF(__pyx_t_5);
16970 __Pyx_XDECREF(__pyx_t_9);
16971 __Pyx_XDECREF(__pyx_t_10);
16972 __Pyx_XDECREF(__pyx_t_11);
16973 __Pyx_AddTraceback("PyClical.asin", __pyx_clineno, __pyx_lineno, __pyx_filename);
16974 __pyx_r = 0;
16975 __pyx_L0;;
16976 __Pyx_XGIVEREF(__pyx_r);
16977 __Pyx_RefNannyFinishContext();
16978 return __pyx_r;
16979 }
16980
16981 /* Python wrapper */
16982 static PyObject* __pyx_pw_8PyClical_67asin(PyObject* __pyx_self, PyObject* __pyx_args,
16983 PyObject* __pyx_kwds); /*proto*/
16984 static char __pyx_doc_8PyClical_66asin[] = "\n Inverse sine of multivector with\n optional complexifier.\n\n >> s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),s))\n {-1}\n >>\n s=\"{-1}\"; x=clifford(s); print(asin(sin(x,s),\"{-2,-1,1}\"))\n {-1}\n >> print(asin(1) / pi)\n 0.5\n >> x=clifford(\"{1,2,3}\"); print(asin(sin(x)))\n {1,2,3}\n ";
16985 static PyObject* __pyx_pw_8PyClical_67asin(PyObject* __pyx_self, PyObject* __pyx_args,
16986 PyObject* __pyx_kwds) {
16987 PyObject* __pyx_v_obj = 0;
16988 PyObject* __pyx_v_i = 0;
16989 int __pyx_lineno = 0;
16990 const char* __pyx_filename = NULL;
16991 int __pyx_clineno = 0;
16992 PyObject* __pyx_r = 0;
16993 __Pyx_RefNannyDeclarations
16994 __Pyx_RefNannySetupContext("asin (wrapper)", 0);
16995 {
16996 static PyObject* __pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
16997 PyObject* values[2] = {0,0};
16998 values[1] = ((PyObject*)Py_None);
16999 if (unlikely(__pyx_kwds)) {
17000 Py_ssize_t kw_args;
17001 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
17002 switch (pos_args) {
17003 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
17004 CYTHON_FALLTHROUGH;
17005 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
17006 CYTHON_FALLTHROUGH;
17007 case 0: break;
17008 default: goto __pyx_L5_argtuple_error;
17009 }
17010 kw_args = PyDict_Size(__pyx_kwds);
17011 switch (pos_args) {
17012 case 0:
17013 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
17014 0)) kw_args--;
17015 else goto __pyx_L5_argtuple_error;
17016 CYTHON_FALLTHROUGH;
17017 case 1:
17018 if (kw_args > 0) {
17019 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
17020 if (value) { values[1] = value; kw_args--; }
17021 }
17022 if (unlikely(kw_args > 0)) {
17023 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
17024 values, pos_args, "asin") < 0)) __PYX_ERR(0, 1747, __pyx_L3_error)
17025 }

```

```

17023 } else {
17024 switch (PyTuple_GET_SIZE(__pyx_args)) {
17025 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
17026 CYTHON_FALLTHROUGH;
17027 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
17028 break;
17029 default: goto __pyx_L5_argtuple_error;
17030 }
17031 }
17032 __pyx_v_obj = values[0];
17033 __pyx_v_i = values[1];
17034 }
17035 goto __pyx_L4_argument_unpacking_done;
17036 __pyx_L5_argtuple_error:;
17037 __Pyx_RaiseArgtupleInvalid("asin", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
17038 __PYX_ERR(0, 1747, __pyx_L3_error)
17039 __pyx_L3_error:;
17040 __Pyx_AddTraceback("PyClical.asin", __pyx_clineno, __pyx_lineno, __pyx_filename);
17041 __Pyx_RefNannyFinishContext();
17042 return NULL;
17043 __pyx_L4_argument_unpacking_done:;
17044 __pyx_r = __pyx_pf_8PyClical_66asin(__pyx_self, __pyx_v_obj, __pyx_v_i);
17045
17046 /* function exit code */
17047 __Pyx_RefNannyFinishContext();
17048 return __pyx_r;
17049 }
17050
17051 static PyObject *__pyx_pf_8PyClical_66asin(CYTHON_UNUSED PyObject *__pyx_self,
17052 PyObject *__pyx_v_obj, PyObject *__pyx_v_i) {
17053 PyObject *__pyx_r = NULL;
17054 __Pyx_RefNannyDeclarations
17055 PyObject *__pyx_t_1 = NULL;
17056 struct __pyx_opt_args_8PyClical_asin __pyx_t_2;
17057 int __pyx_lineno = 0;
17058 const char *__pyx_filename = NULL;
17059 int __pyx_clineno = 0;
17060 __Pyx_RefNannySetupContext("asin", 0);
17061 __Pyx_XDECREF(__pyx_r);
17062 __pyx_t_2.__pyx_n = 1;
17063 __pyx_t_2.i = __pyx_v_i;
17064 __pyx_t_1 = __pyx_f_8PyClical_asin(__pyx_v_obj, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1747, __pyx_L1_error)
17065 __Pyx_GOTREF(__pyx_t_1);
17066 __pyx_r = __pyx_t_1;
17067 __pyx_t_1 = 0;
17068 goto __pyx_L0;
17069
17070 /* function exit code */
17071 __pyx_L1_error:;
17072 __Pyx_XDECREF(__pyx_t_1);
17073 __Pyx_AddTraceback("PyClical.asin", __pyx_clineno, __pyx_lineno, __pyx_filename);
17074 __pyx_r = NULL;
17075 __pyx_L0:;
17076 __Pyx_XGIVEREF(__pyx_r);
17077 __Pyx_RefNannyFinishContext();
17078 return __pyx_r;
17079 }
17080
17081 /* "PyClical.pyx":1768
17082 *
17083 * return clifford().wrap(glucat.asin(toClifford(obj)))
17084 *
17085 * cpdef inline sinh(obj):
17086 * # ««««««««
17087 * """
17088 * Hyperbolic sine of multivector.
17089 * */
17090
17091 static PyObject *__pyx_pw_8PyClical_69sinh(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
17092 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_sinh(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
17093 PyObject *__pyx_r = NULL;
17094 __Pyx_RefNannyDeclarations
17095 PyObject *__pyx_t_1 = NULL;
17096 PyObject *__pyx_t_2 = NULL;
17097 PyObject *__pyx_t_3 = NULL;
17098 PyObject *__pyx_t_4 = NULL;
17099 PyObject *__pyx_t_5 = NULL;
17100 PyObject *__pyx_t_6 = NULL;
17101 PyObject *__pyx_t_7 = NULL;
17102 PyObject *__pyx_t_8 = NULL;
17103 int __pyx_lineno = 0;
17104 const char *__pyx_filename = NULL;
17105 int __pyx_clineno = 0;
17106 __Pyx_RefNannySetupContext("sinh", 0);
17107
17108 /* "PyClical.pyx":1777

```



```

17105 * 0.5{1,2}
17106 * """
17107 * try: # ««««««««
17108 * return math.sinh(obj)
17109 * except:
17110 */
17111
17112 {
17113 __Pyx_PyThreadState_declare
17114 __Pyx_PyThreadState_assign
17115 __Pyx_ExceptionSave(&__pyx_t_1, &__pyx_t_2, &__pyx_t_3);
17116 __Pyx_XGOTREF(__pyx_t_1);
17117 __Pyx_XGOTREF(__pyx_t_2);
17118 __Pyx_XGOTREF(__pyx_t_3);
17119 /*try:*/ {
17120
17121 /* "PyClical.pyx":1778
17122
17123 try:
17124 return math.sinh(obj) # ««««««««
17125 except:
17126 return clifford().wrap(glucat.sinh(toClifford(obj)))
17127 */
17128
17129 __Pyx_XDECREF(__pyx_r);
17130 __Pyx_GetModuleGlobalName(__pyx_t_5, __pyx_n_s_math); if (unlikely(!__pyx_t_5))
17131 __PYX_ERR(0, 1778, __pyx_L3_error)
17132 __Pyx_GOTREF(__pyx_t_5);
17133 __pyx_t_6 = __Pyx_PyObject_GetAttrStr(__pyx_t_5, __pyx_n_s_sinh); if
17134 (unlikely(!__pyx_t_6)) __PYX_ERR(0, 1778, __pyx_L3_error)
17135 __Pyx_GOTREF(__pyx_t_6);
17136 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
17137 __pyx_t_5 = NULL;
17138 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_6))) {
17139 __pyx_t_5 = PyMethod_GET_SELF(__pyx_t_6);
17140 if (likely(__pyx_t_5)) {
17141 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_6);
17142 __Pyx_INCREF(__pyx_t_5);
17143 __Pyx_INCREF(function);
17144 __Pyx_DECREF_SET(__pyx_t_6, function);
17145 }
17146 }
17147 __pyx_t_4 = (__pyx_t_5) ? __Pyx_PyObject_Call2Args(__pyx_t_6, __pyx_t_5,
17148 __pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_6, __pyx_v_obj);
17149 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
17150 if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1778, __pyx_L3_error)
17151 __Pyx_GOTREF(__pyx_t_4);
17152 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
17153 __pyx_r = __pyx_t_4;
17154 __pyx_t_4 = 0;
17155 goto __pyx_L7_try_return;
17156
17157 /* "PyClical.pyx":1777
17158
17159 0.5{1,2}
17160 """
17161 try: # ««««««««
17162 return math.sinh(obj)
17163 except:
17164
17165 */
17166
17167 }
17168 __pyx_L3_error:;
17169 __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
17170 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
17171 __Pyx_XDECREF(__pyx_t_6); __pyx_t_6 = 0;
17172
17173 /* "PyClical.pyx":1779
17174
17175 try:
17176 return math.sinh(obj)
17177 except: # ««««««««
17178 return clifford().wrap(glucat.sinh(toClifford(obj)))
17179 */
17180
17181 /*except:*/ {
17182 __Pyx_AddTraceback("PyClical.sinh", __pyx_clineno, __pyx_lineno,
17183 __pyx_filename);
17184 if (__Pyx_GetException(&__pyx_t_4, &__pyx_t_6, &__pyx_t_5) < 0) __PYX_ERR(0,
17185 1779, __pyx_L5_except_error)
17186 __Pyx_GOTREF(__pyx_t_4);
17187 __Pyx_GOTREF(__pyx_t_6);
17188 __Pyx_GOTREF(__pyx_t_5);
17189
17190 /* "PyClical.pyx":1780
17191
17192 return math.sinh(obj)
17193 except:
17194 return clifford().wrap(glucat.sinh(toClifford(obj))) # ««««««««
17195 */
17196
17197 __Pyx_XDECREF(__pyx_r);

```

```

17187 __pyx_t_7 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyCliclal_clifford)); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 1780,
__pyx_L5_except_error)
17188 __Pyx_GOTREF(__pyx_t_7);
17189 __pyx_t_8 = __pyx_f_8PyCliclal_8clifford_wrap(((struct
__pyx_obj_8PyCliclal_clifford *)__pyx_t_7), sinh(__pyx_f_8PyCliclal_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_8)) __PYX_ERR(0, 1780, __pyx_L5_except_error)
17190 __Pyx_GOTREF(__pyx_t_8);
17191 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
17192 __pyx_r = __pyx_t_8;
17193 __pyx_t_8 = 0;
17194 __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
17195 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
17196 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
17197 goto __pyx_L6_except_return;
17198 }
17199 __pyx_L5_except_error::;
17200
17201 /* "PyCliclal.pyx":1777
17202 * 0.5{1,2}
17203 * """
17204 * try: # ««««««««
17205 * return math.sinh(obj)
17206 * except:
17207 */
17208 __Pyx_XGIVEREF(__pyx_t_1);
17209 __Pyx_XGIVEREF(__pyx_t_2);
17210 __Pyx_XGIVEREF(__pyx_t_3);
17211 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
17212 goto __pyx_L1_error;
17213 __pyx_L7_try_return::;
17214 __Pyx_XGIVEREF(__pyx_t_1);
17215 __Pyx_XGIVEREF(__pyx_t_2);
17216 __Pyx_XGIVEREF(__pyx_t_3);
17217 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
17218 goto __pyx_L0;
17219 __pyx_L6_except_return::;
17220 __Pyx_XGIVEREF(__pyx_t_1);
17221 __Pyx_XGIVEREF(__pyx_t_2);
17222 __Pyx_XGIVEREF(__pyx_t_3);
17223 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
17224 goto __pyx_L0;
17225 }
17226
17227 /* "PyCliclal.pyx":1768
17228 * return clifford().wrap(glucat.asin(toClifford(obj)))
17229 *
17230 * cpdef inline sinh(obj): # ««««««««
17231 * """
17232 * Hyperbolic sine of multivector.
17233 */
17234
17235 /* function exit code */
17236 __pyx_L1_error::;
17237 __Pyx_XDECREF(__pyx_t_4);
17238 __Pyx_XDECREF(__pyx_t_5);
17239 __Pyx_XDECREF(__pyx_t_6);
17240 __Pyx_XDECREF(__pyx_t_7);
17241 __Pyx_XDECREF(__pyx_t_8);
17242 __Pyx_AddTraceback("PyCliclal.sinh", __pyx_clineno, __pyx_lineno, __pyx_filename);
17243 __pyx_r = 0;
17244 __pyx_L0::;
17245 __Pyx_XGIVEREF(__pyx_r);
17246 __Pyx_RefNannyFinishContext();
17247 return __pyx_r;
17248 }
17249
17250 /* Python wrapper */
17251 static PyObject *__pyx_pw_8PyCliclal_69sinh(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
17252 static char __pyx_doc_8PyCliclal_68sinh[] = "\n Hyperbolic sine of multivector.\n\n
>> x=clifford(\"{1,2}\") * pi/2; print(sinh(x))\n {1,2}\n >> x=clifford(\"{1,2}\") * pi/6;
print(sinh(x))\n 0.5{1,2}\n ";
17253 static PyObject *__pyx_pw_8PyCliclal_69sinh(PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
17254 PyObject *__pyx_r = 0;
17255 __Pyx_RefNannyDeclarations
17256 __Pyx_RefNannySetupContext("sinh (wrapper)", 0);
17257 __pyx_r = __pyx_pf_8PyCliclal_68sinh(__pyx_self, ((PyObject *)__pyx_v_obj));
17258
17259 /* function exit code */
17260 __Pyx_RefNannyFinishContext();
17261 return __pyx_r;
17262 }
17263
17264 static PyObject *__pyx_pf_8PyCliclal_68sinh(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj) {

```

```

17265 PyObject *__pyx_r = NULL;
17266 __Pyx_RefNannyDeclarations
17267 PyObject *__pyx_t_1 = NULL;
17268 int __pyx_lineno = 0;
17269 const char *__pyx_filename = NULL;
17270 int __pyx_clineno = 0;
17271 __Pyx_RefNannySetupContext("sinh", 0);
17272 __Pyx_XDECREF(__pyx_r);
17273 __pyx_t_1 = __pyx_f_8PyClical_sinh(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1768, __pyx_L1_error)
17274 __Pyx_GOTREF(__pyx_t_1);
17275 __pyx_r = __pyx_t_1;
17276 __pyx_t_1 = 0;
17277 goto __pyx_L0;
17278
17279 /* function exit code */
17280 __pyx_L1_error:;
17281 __Pyx_XDECREF(__pyx_t_1);
17282 __Pyx_AddTraceback("PyClical.sinh", __pyx_clineno, __pyx_lineno, __pyx_filename);
17283 __pyx_r = NULL;
17284 __pyx_L0:;
17285 __Pyx_XGIVEREF(__pyx_r);
17286 __Pyx_RefNannyFinishContext();
17287 return __pyx_r;
17288 }
17289
17290 /* "PyClical.pyx":1782
17291 * return clifford().wrap(glucat.sinh(toClifford(obj)))
17292 *
17293 * cpdef inline asinh(obj,i = None): # ««««««««
17294 * """
17295 * Inverse hyperbolic sine of multivector with optional complexifier.
17296 */
17297
17298 static PyObject *__pyx_pw_8PyClical_7lasinh(PyObject *__pyx_self, PyObject
__pyx_args, PyObject *__pyx_kwds); /*proto*/
17299 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_asinh(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_asinh *__pyx_optional_args) {
17300 PyObject *__pyx_v_i = ((PyObject *)Py_None);
17301 PyObject *__pyx_r = NULL;
17302 __Pyx_RefNannyDeclarations
17303 int __pyx_t_1;
17304 int __pyx_t_2;
17305 PyObject *__pyx_t_3 = NULL;
17306 Clifford __pyx_t_4;
17307 PyObject *__pyx_t_5 = NULL;
17308 PyObject *__pyx_t_6 = NULL;
17309 PyObject *__pyx_t_7 = NULL;
17310 PyObject *__pyx_t_8 = NULL;
17311 PyObject *__pyx_t_9 = NULL;
17312 PyObject *__pyx_t_10 = NULL;
17313 PyObject *__pyx_t_11 = NULL;
17314 int __pyx_lineno = 0;
17315 const char *__pyx_filename = NULL;
17316 int __pyx_clineno = 0;
17317 __Pyx_RefNannySetupContext("asinh", 0);
17318 if (__pyx_optional_args) {
17319 if (__pyx_optional_args->__pyx_n > 0) {
17320 __pyx_v_i = __pyx_optional_args->i;
17321 }
17322 }
17323
17324 /* "PyClical.pyx":1793
17325 * {1,2}
17326 * """
17327 * if not (i is None): # ««««««««
17328 * return clifford().wrap(glucat.asinh(toClifford(obj), toClifford(i)))
17329 * else:
17330 */
17331 __pyx_t_1 = (__pyx_v_i != Py_None);
17332 __pyx_t_2 = (__pyx_t_1 != 0);
17333 if (__pyx_t_2) {
17334
17335 /* "PyClical.pyx":1794
17336 * """
17337 * if not (i is None):
17338 * return clifford().wrap(glucat.asinh(toClifford(obj), toClifford(i)))
17339 * else:
17340 * try:
17341 */
17342 __Pyx_XDECREF(__pyx_r);
17343 __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1794, __pyx_L1_error)
17344 __Pyx_GOTREF(__pyx_t_3);
17345 try {
17346 __pyx_t_4 = asinh(__pyx_f_8PyClical_toClifford(__pyx_v_obj),

```

```

__pyx_f_8PyClical_toClifford(__pyx_v_i));
17347 } catch(...) {
17348 __Pyx_CppExn2PyErr();
17349 __PYX_ERR(0, 1794, __pyx_L1_error)
17350 }
17351 __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1794, __pyx_L1_error)
17352 __Pyx_GOTREF(__pyx_t_5);
17353 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
17354 __pyx_r = __pyx_t_5;
17355 __pyx_t_5 = 0;
17356 goto __pyx_L0;
17357
17358 /* "PyClical.pyx":1793
17359 *
17360 * """
17361 * if not (i is None):
17362 * return clifford().wrap(glucat.asinh(toClifford(obj), toClifford(i)))
17363 * else:
17364 */
17365 }
17366
17367 /* "PyClical.pyx":1796
17368 * return clifford().wrap(glucat.asinh(toClifford(obj), toClifford(i)))
17369 * else:
17370 * try:
17371 * return math.asinh(obj)
17372 * except:
17373 */
17374 /*else*/ {
17375 {
17376 __Pyx_PyThreadState_declare
17377 __Pyx_PyThreadState_assign
17378 __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
17379 __Pyx_XGOTREF(__pyx_t_6);
17380 __Pyx_XGOTREF(__pyx_t_7);
17381 __Pyx_XGOTREF(__pyx_t_8);
17382 /*try:*/ {
17383
17384 /* "PyClical.pyx":1797
17385 * else:
17386 * try:
17387 * return math.asinh(obj)
17388 * except:
17389 * return clifford().wrap(glucat.asinh(toClifford(obj)))
17390 */
17391 __Pyx_XDECREF(__pyx_r);
17392 __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
17393 (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1797, __pyx_L4_error)
17394 __Pyx_GOTREF(__pyx_t_3);
17395 __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_asinh); if
17396 (unlikely(!__pyx_t_9)) __PYX_ERR(0, 1797, __pyx_L4_error)
17397 __Pyx_GOTREF(__pyx_t_9);
17398 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
17399 __pyx_t_3 = NULL;
17400 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
17401 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
17402 if (likely(__pyx_t_3)) {
17403 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
17404 __Pyx_INCREF(__pyx_t_3);
17405 __Pyx_INCREF(function);
17406 __Pyx_DECREF_SET(__pyx_t_9, function);
17407 }
17408 }
17409 __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
17410 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
17411 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1797, __pyx_L4_error)
17412 __Pyx_GOTREF(__pyx_t_5);
17413 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
17414 __pyx_r = __pyx_t_5;
17415 __pyx_t_5 = 0;
17416 goto __pyx_L8_try_return;
17417
17418 /* "PyClical.pyx":1796
17419 * return clifford().wrap(glucat.asinh(toClifford(obj), toClifford(i)))
17420 * else:
17421 * try:
17422 * return math.asinh(obj)
17423 * except:
17424 */
17425 }
17426 __pyx_L4_error:;
17427 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
17428 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
17429 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;

```

```

17429 /* "PyClical.pyx":1798
17430 * try:
17431 * return math.asinh(obj)
17432 * except:
17433 * # ««««««««
17434 * return clifford().wrap(glucat.asinh(toClifford(obj)))
17435 */
17436
17437 /*except:*/ {
17438 __Pyx_AddTraceback("PyClical.asinh", __pyx_clineno, __pyx_lineno,
__pyx_filename);
17439 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
17440 1798, __pyx_L6_except_error)
17441 __Pyx_GOTREF(__pyx_t_5);
17442 __Pyx_GOTREF(__pyx_t_9);
17443 __Pyx_GOTREF(__pyx_t_3);
17444 /* "PyClical.pyx":1799
17445 * return math.asinh(obj)
17446 * except:
17447 * return clifford().wrap(glucat.asinh(toClifford(obj)))
17448 * # ««««««««
17449 * cpdef inline tan(obj,i = None):
17450 */
17451 __Pyx_XDECREF(__pyx_r);
17452 __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1799,
__pyx_L6_except_error)
17453 __Pyx_GOTREF(__pyx_t_10);
17454 __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct
__pyx_obj_8PyClical_clifford *)__pyx_t_10), asinh(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 1799, __pyx_L6_except_error)
17455 __Pyx_GOTREF(__pyx_t_11);
17456 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
17457 __pyx_r = __pyx_t_11;
17458 __pyx_t_11 = 0;
17459 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
17460 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
17461 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
17462 goto __pyx_L7_except_return;
17463 }
17464 __pyx_L6_except_error;
17465
17466 /* "PyClical.pyx":1796
17467 * return clifford().wrap(glucat.asinh(toClifford(obj), toClifford(i)))
17468 * else:
17469 * try:
17470 * # ««««««««
17471 * return math.asinh(obj)
17472 * except:
17473 * # ««««««««
17474 * Pyx_XGIVEREF(__pyx_t_6);
17475 * Pyx_XGIVEREF(__pyx_t_7);
17476 * Pyx_XGIVEREF(__pyx_t_8);
17477 * Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
17478 * goto __pyx_L1_error;
17479 * __pyx_L8_try_return;
17480 * Pyx_XGIVEREF(__pyx_t_6);
17481 * Pyx_XGIVEREF(__pyx_t_7);
17482 * Pyx_XGIVEREF(__pyx_t_8);
17483 * Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
17484 * goto __pyx_L0;
17485 * __pyx_L7_except_return;
17486 * Pyx_XGIVEREF(__pyx_t_6);
17487 * Pyx_XGIVEREF(__pyx_t_7);
17488 * Pyx_XGIVEREF(__pyx_t_8);
17489 * Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
17490 * goto __pyx_L0;
17491 }
17492
17493 /* "PyClical.pyx":1782
17494 * return clifford().wrap(glucat.sinh(toClifford(obj)))
17495 * cpdef inline asinh(obj,i = None):
17496 * # ««««««««
17497 * """
17498 * Inverse hyperbolic sine of multivector with optional complexifier.
17499 */
17500
17501 /* function exit code */
17502 __pyx_L1_error;
17503 __Pyx_XDECREF(__pyx_t_3);
17504 __Pyx_XDECREF(__pyx_t_5);
17505 __Pyx_XDECREF(__pyx_t_9);
17506 __Pyx_XDECREF(__pyx_t_10);
17507 __Pyx_XDECREF(__pyx_t_11);
17508 __Pyx_AddTraceback("PyClical.asinh", __pyx_clineno, __pyx_lineno, __pyx_filename);
17509 __pyx_r = 0;
17510 __pyx_L0;

```

```

17510 __Pyx_XGIVEREF(__pyx_r);
17511 __Pyx_RefNannyFinishContext();
17512 return __pyx_r;
17513 }
17514
17515 /* Python wrapper */
17516 static PyObject * __pyx_pw_8PyClical_7lasinh(PyObject * __pyx_self, PyObject
*__pyx_args, PyObject * __pyx_kwds); /*proto*/
17517 static char __pyx_doc_8PyClical_70asinh[] = "\n Inverse hyperbolic sine of
multivector with optional complexifier.\n\n >> x=clifford(\"{1,2}\"); print(asinh(x,\"{1,2,3}\") *
2/pi)\n {1,2}\n >> x=clifford(\"{1,2}\"); print(asinh(x) * 2/pi)\n {1,2}\n >>
x=clifford(\"{1,2}\") / 2; print(asinh(x) * 6/pi)\n {1,2}\n ";
17518 static PyObject * __pyx_pw_8PyClical_7lasinh(PyObject * __pyx_self, PyObject
*__pyx_args, PyObject * __pyx_kwds) {
17519 PyObject * __pyx_v_obj = 0;
17520 PyObject * __pyx_v_i = 0;
17521 int __pyx_lineno = 0;
17522 const char * __pyx_filename = NULL;
17523 int __pyx_clineno = 0;
17524 PyObject * __pyx_r = 0;
17525 __Pyx_RefNannyDeclarations
17526 __Pyx_RefNannySetupContext("asinh (wrapper)", 0);
17527 {
17528 static PyObject * __pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
17529 PyObject* values[2] = {0,0};
17530 values[1] = ((PyObject *)Py_None);
17531 if (unlikely(__pyx_kwds)) {
17532 Py_ssize_t kw_args;
17533 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
17534 switch (pos_args) {
17535 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
CYTHON_FALLTHROUGH;
17536 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
CYTHON_FALLTHROUGH;
17537 case 0: break;
17538 default: goto __pyx_L5_argtuple_error;
17539 }
17540 kw_args = PyDict_Size(__pyx_kwds);
17541 switch (pos_args) {
17542 case 0:
17543 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
17544 0)) kw_args--;
17545 else goto __pyx_L5_argtuple_error;
17546 CYTHON_FALLTHROUGH;
17547 case 1:
17548 if (kw_args > 0) {
17549 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
17550 if (value) { values[1] = value; kw_args--; }
17551 }
17552 }
17553 if (unlikely(kw_args > 0)) {
17554 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
17555 values, pos_args, "asinh") < 0)) __PYX_ERR(0, 1782, __pyx_L3_error)
17556 }
17557 else {
17558 switch (PyTuple_GET_SIZE(__pyx_args)) {
17559 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
CYTHON_FALLTHROUGH;
17560 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
17561 break;
17562 default: goto __pyx_L5_argtuple_error;
17563 }
17564 }
17565 }
17566 __pyx_v_obj = values[0];
17567 __pyx_v_i = values[1];
17568 }
17569 goto __pyx_L4_argument_unpacking_done;
17570 __pyx_L5_argtuple_error:;
17571 __Pyx_RaiseArgtupleInvalid("asinh", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
17572 __PYX_ERR(0, 1782, __pyx_L3_error)
17573 __pyx_L3_error:;
17574 __Pyx_AddTraceback("PyClical.asinh", __pyx_clineno, __pyx_lineno, __pyx_filename);
17575 __Pyx_RefNannyFinishContext();
17576 return NULL;
17577 __pyx_L4_argument_unpacking_done:;
17578 __pyx_r = __pyx_pf_8PyClical_70asinh(__pyx_self, __pyx_v_obj, __pyx_v_i);
17579
17580 /* function exit code */
17581 __Pyx_RefNannyFinishContext();
17582 return __pyx_r;
17583 }
17584 static PyObject * __pyx_pf_8PyClical_70asinh(CYTHON_UNUSED PyObject * __pyx_self,
PyObject * __pyx_v_obj, PyObject * __pyx_v_i) {
17585 PyObject * __pyx_r = NULL;
17586 __Pyx_RefNannyDeclarations
17587 PyObject * __pyx_t_1 = NULL;

```

```

17588 struct __pyx_opt_args_8PyClical_asinh __pyx_t_2;
17589 int __pyx_lineno = 0;
17590 const char *__pyx_filename = NULL;
17591 int __pyx_clineno = 0;
17592 __Pyx_RefNannySetupContext("asinh", 0);
17593 __Pyx_XDECREF(__pyx_r);
17594 __pyx_t_2.__pyx_n = 1;
17595 __pyx_t_2.i = __pyx_v_i;
17596 __pyx_t_1 = __pyx_f_8PyClical_asinh(__pyx_v_obj, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1782, __pyx_L1_error)
17597 __Pyx_GOTREF(__pyx_t_1);
17598 __pyx_r = __pyx_t_1;
17599 __pyx_t_1 = 0;
17600 goto __pyx_L0;
17601
17602 /* function exit code */
17603 __pyx_L1_error:;
17604 __Pyx_XDECREF(__pyx_t_1);
17605 __Pyx_AddTraceback("PyClical.asinh", __pyx_clineno, __pyx_lineno, __pyx_filename);
17606 __pyx_r = NULL;
17607 __pyx_L0:;
17608 __Pyx_XGIVEREF(__pyx_r);
17609 __Pyx_RefNannyFinishContext();
17610 return __pyx_r;
17611 }
17612
17613 /* "PyClical.pyx":1801
17614 *
17615 * return clifford().wrap(glucat.asinh(toClifford(obj)))
17616 * cpdef inline tan(obj,i = None):
17617 * """
17618 * Tangent of multivector with optional complexifier.
17619 */
17620
17621 static PyObject *__pyx_pw_8PyClical_73tan(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds); /*proto*/
17622 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_tan(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_tan *__pyx_optional_args) {
17623 PyObject *__pyx_v_i = ((PyObject *)Py_None);
17624 PyObject *__pyx_r = NULL;
17625 __Pyx_RefNannyDeclarations
17626 int __pyx_t_1;
17627 int __pyx_t_2;
17628 PyObject *__pyx_t_3 = NULL;
17629 Clifford __pyx_t_4;
17630 PyObject *__pyx_t_5 = NULL;
17631 PyObject *__pyx_t_6 = NULL;
17632 PyObject *__pyx_t_7 = NULL;
17633 PyObject *__pyx_t_8 = NULL;
17634 PyObject *__pyx_t_9 = NULL;
17635 PyObject *__pyx_t_10 = NULL;
17636 PyObject *__pyx_t_11 = NULL;
17637 int __pyx_lineno = 0;
17638 const char *__pyx_filename = NULL;
17639 int __pyx_clineno = 0;
17640 __Pyx_RefNannySetupContext("tan", 0);
17641 if (__pyx_optional_args) {
17642 if (__pyx_optional_args->__pyx_n > 0) {
17643 __pyx_v_i = __pyx_optional_args->i;
17644 }
17645 }
17646
17647 /* "PyClical.pyx":1810
17648 * 0.7616{1,2}
17649 * """
17650 * if not (i is None):
17651 * return clifford().wrap(glucat.tan(toClifford(obj), toClifford(i)))
17652 * else:
17653 */
17654 __pyx_t_1 = (__pyx_v_i != Py_None);
17655 __pyx_t_2 = (__pyx_t_1 != 0);
17656 if (__pyx_t_2) {
17657
17658 /* "PyClical.pyx":1811
17659 *
17660 * if not (i is None):
17661 * return clifford().wrap(glucat.tan(toClifford(obj), toClifford(i)))
17662 * else:
17663 * try:
17664 */
17665 __Pyx_XDECREF(__pyx_r);
17666 __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1811, __pyx_L1_error)
17667 __Pyx_GOTREF(__pyx_t_3);
17668 try {
17669 __pyx_t_4 = tan(__pyx_f_8PyClical_toClifford(__pyx_v_obj),
__pyx_f_8PyClical_toClifford(__pyx_v_i));

```

```

17670 } catch(...) {
17671 __Pyx_CppExn2PyErr();
17672 __PYX_ERR(0, 1811, __pyx_L1_error)
17673 }
17674 __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1811, __pyx_L1_error)
17675 __Pyx_GOTREF(__pyx_t_5);
17676 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
17677 __pyx_r = __pyx_t_5;
17678 __pyx_t_5 = 0;
17679 goto __pyx_L0;
17680
17681 /* "PyClical.pyx":1810
17682 * 0.7616{1,2}
17683 * """
17684 * if not (i is None): # ««««««««
17685 * return clifford().wrap(glucat.tan(toClifford(obj)), toClifford(i)))
17686 * else:
17687 */
17688 }
17689
17690 /* "PyClical.pyx":1813
17691 * return clifford().wrap(glucat.tan(toClifford(obj)), toClifford(i)))
17692 * else:
17693 * try: # ««««««««
17694 * return math.tan(obj)
17695 * except:
17696 */
17697 /*else*/ {
17698 {
17699 __Pyx_PyThreadState_declare
17700 __Pyx_PyThreadState_assign
17701 __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
17702 __Pyx_XGOTREF(__pyx_t_6);
17703 __Pyx_XGOTREF(__pyx_t_7);
17704 __Pyx_XGOTREF(__pyx_t_8);
17705 /*try:*/ {
17706
17707 /* "PyClical.pyx":1814
17708 * else:
17709 * try:
17710 * return math.tan(obj) # ««««««««
17711 * except:
17712 * return clifford().wrap(glucat.tan(toClifford(obj)))
17713 */
17714 __Pyx_XDECREF(__pyx_r);
17715 __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1814, __pyx_L4_error)
17716 __Pyx_GOTREF(__pyx_t_3);
17717 __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_tan); if
(unlikely(!__pyx_t_9)) __PYX_ERR(0, 1814, __pyx_L4_error)
17718 __Pyx_GOTREF(__pyx_t_9);
17719 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
17720 __pyx_t_3 = NULL;
17721 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
17722 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
17723 if (likely(__pyx_t_3)) {
17724 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
17725 __Pyx_INCREF(__pyx_t_3);
17726 __Pyx_INCREF(function);
17727 __Pyx_DECREF_SET(__pyx_t_9, function);
17728 }
17729 }
17730 __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
17731 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
17732 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1814, __pyx_L4_error)
17733 __Pyx_GOTREF(__pyx_t_5);
17734 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
17735 __pyx_r = __pyx_t_5;
17736 __pyx_t_5 = 0;
17737 goto __pyx_L8_try_return;
17738
17739 /* "PyClical.pyx":1813
17740 * return clifford().wrap(glucat.tan(toClifford(obj)), toClifford(i)))
17741 * else:
17742 * try: # ««««««««
17743 * return math.tan(obj)
17744 * except:
17745 */
17746 }
17747 __pyx_L4_error:;
17748 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
17749 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
17750 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
17751
17752 /* "PyClical.pyx":1815

```



```

17753 * try:
17754 * return math.tan(obj)
17755 * except: # ««««««««
17756 * return clifford().wrap(glucat.tan(toClifford(obj)))
17757 *
17758 */
17759
17760 /*except:*/ {
17761 __Pyx_AddTraceback("PyClical.tan", __pyx_clineno, __pyx_lineno,
17762 __pyx_filename);
17763 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
17764 1815, __pyx_L6_except_error)
17765 __Pyx_GOTREF(__pyx_t_5);
17766 __Pyx_GOTREF(__pyx_t_9);
17767 __Pyx_GOTREF(__pyx_t_3);
17768
17769 /* "PyClical.pyx":1816
17770 * return math.tan(obj)
17771 * except:
17772 * return clifford().wrap(glucat.tan(toClifford(obj))) # ««««««««
17773 *
17774 * cpdef inline atan(obj,i = None):
17775 */
17776 __Pyx_XDECREF(__pyx_r);
17777 __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
17778 *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1816,
17779 __pyx_L6_except_error)
17780 __Pyx_GOTREF(__pyx_t_10);
17781 __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct
17782 __pyx_obj_8PyClical_clifford *)__pyx_t_10), tan(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
17783 (unlikely(!__pyx_t_11)) __PYX_ERR(0, 1816, __pyx_L6_except_error)
17784 __Pyx_GOTREF(__pyx_t_11);
17785 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
17786 __pyx_r = __pyx_t_11;
17787 __pyx_t_11 = 0;
17788 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
17789 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
17790 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
17791 goto __pyx_L7_except_return;
17792 }
17793 __pyx_L6_except_error:;
17794
17795 /* "PyClical.pyx":1813
17796 * return clifford().wrap(glucat.tan(toClifford(obj), toClifford(i)))
17797 *
17798 * else:
17799 * try: # ««««««««
17800 * return math.tan(obj)
17801 * except:
17802 */
17803 __Pyx_XGIVEREF(__pyx_t_6);
17804 __Pyx_XGIVEREF(__pyx_t_7);
17805 __Pyx_XGIVEREF(__pyx_t_8);
17806 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
17807 goto __pyx_L1_error;
17808 __pyx_L8_try_return:;
17809 __Pyx_XGIVEREF(__pyx_t_6);
17810 __Pyx_XGIVEREF(__pyx_t_7);
17811 __Pyx_XGIVEREF(__pyx_t_8);
17812 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
17813 goto __pyx_L0;
17814 __pyx_L7_except_return:;
17815 __Pyx_XGIVEREF(__pyx_t_6);
17816 __Pyx_XGIVEREF(__pyx_t_7);
17817 __Pyx_XGIVEREF(__pyx_t_8);
17818 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
17819 goto __pyx_L0;
17820 }
17821 }
17822
17823 /* "PyClical.pyx":1801
17824 * return clifford().wrap(glucat.asinh(toClifford(obj)))
17825 *
17826 * cpdef inline tan(obj,i = None): # ««««««««
17827 * """
17828 * Tangent of multivector with optional complexifier.
17829 */
17830
17831 /* function exit code */
17832 __pyx_L1_error:;
17833 __Pyx_XDECREF(__pyx_t_3);
17834 __Pyx_XDECREF(__pyx_t_5);
17835 __Pyx_XDECREF(__pyx_t_9);
17836 __Pyx_XDECREF(__pyx_t_10);
17837 __Pyx_XDECREF(__pyx_t_11);
17838 __Pyx_AddTraceback("PyClical.tan", __pyx_clineno, __pyx_lineno, __pyx_filename);
17839 __pyx_r = 0;
17840 __pyx_L0:;
17841 __Pyx_XGIVEREF(__pyx_r);

```

```

17834 __Pyx_RefNannyFinishContext();
17835 return __pyx_r;
17836 }
17837
17838 /* Python wrapper */
17839 static PyObject *__pyx_pw_8PyClical_73tan(PyObject *__pyx_self, PyObject *__pyx_args,
17840 PyObject *__pyx_kwds); /*proto*/
17841 static char __pyx_doc_8PyClical_72tan[] = "\n Tangent of multivector with optional
17842 complexifier.\n\n >> x=clifford(\"{1,2}\"); print(tan(x,\"{1,2,3}\"))\n 0.7616{1,2}\n >>
17843 x=clifford(\"{1,2}\"); print(tan(x))\n 0.7616{1,2}\n ";
17844 static PyObject *__pyx_pw_8PyClical_73tan(PyObject *__pyx_self, PyObject *__pyx_args,
17845 PyObject *__pyx_kwds) {
17846 PyObject *__pyx_v_obj = 0;
17847 PyObject *__pyx_v_i = 0;
17848 int __pyx_lineno = 0;
17849 const char *__pyx_filename = NULL;
17850 int __pyx_clineno = 0;
17851 PyObject *__pyx_r = 0;
17852 __Pyx_RefNannyDeclarations
17853 __Pyx_RefNannySetupContext("tan (wrapper)", 0);
17854 {
17855 static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
17856 PyObject* values[2] = {0,0};
17857 values[1] = (PyObject *)Py_None;
17858 if (unlikely(__pyx_kwds)) {
17859 Py_ssize_t kw_args;
17860 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
17861 switch (pos_args) {
17862 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
17863 CYTHON_FALLTHROUGH;
17864 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
17865 CYTHON_FALLTHROUGH;
17866 case 0: break;
17867 default: goto __pyx_L5_argtuple_error;
17868 }
17869 kw_args = PyDict_Size(__pyx_kwds);
17870 switch (pos_args) {
17871 case 0:
17872 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
17873 0)) kw_args--;
17874 else goto __pyx_L5_argtuple_error;
17875 CYTHON_FALLTHROUGH;
17876 case 1:
17877 if (kw_args > 0) {
17878 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
17879 if (value) { values[1] = value; kw_args--; }
17880 }
17881 if (unlikely(kw_args > 0)) {
17882 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
17883 values, pos_args, "tan") < 0)) __PYX_ERR(0, 1801, __pyx_L3_error)
17884 }
17885 } else {
17886 switch (PyTuple_GET_SIZE(__pyx_args)) {
17887 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
17888 CYTHON_FALLTHROUGH;
17889 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
17890 break;
17891 default: goto __pyx_L5_argtuple_error;
17892 }
17893 }
17894 __pyx_v_obj = values[0];
17895 __pyx_v_i = values[1];
17896 }
17897 goto __pyx_L4_argument_unpacking_done;
17898 __pyx_L5_argtuple_error:;
17899 __Pyx_RaiseArgtupleInvalid("tan", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
17900 __PYX_ERR(0, 1801, __pyx_L3_error)
17901 __pyx_L3_error:;
17902 __Pyx_AddTraceback("PyClical.tan", __pyx_clineno, __pyx_lineno, __pyx_filename);
17903 __Pyx_RefNannyFinishContext();
17904 return NULL;
17905 __pyx_L4_argument_unpacking_done:;
17906 __pyx_r = __pyx_pf_8PyClical_72tan(__pyx_self, __pyx_v_obj, __pyx_v_i);
17907
17908 /* function exit code */
17909 __Pyx_RefNannyFinishContext();
17910 return __pyx_r;
17911 }
17912
17913 static PyObject *__pyx_pf_8PyClical_72tan(CYTHON_UNUSED PyObject *__pyx_self, PyObject
17914 *__pyx_v_obj, PyObject *__pyx_v_i) {
17915 PyObject *__pyx_r = NULL;
17916 __Pyx_RefNannyDeclarations
17917 PyObject *__pyx_t_1 = NULL;
17918 struct __pyx_opt_args_8PyClical_tan __pyx_t_2;
17919 int __pyx_lineno = 0;

```

```

17913 const char *__pyx_filename = NULL;
17914 int __pyx_clineno = 0;
17915 __Pyx_RefNannySetupContext("tan", 0);
17916 __Pyx_XDECREF(__pyx_r);
17917 __pyx_t_2.__pyx_n = 1;
17918 __pyx_t_2.i = __pyx_v_i;
17919 __pyx_t_1 = __pyx_f_8PyClical_tan(__pyx_v_obj, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1801, __pyx_L1_error)
17920 __Pyx_GOTREF(__pyx_t_1);
17921 __pyx_r = __pyx_t_1;
17922 __pyx_t_1 = 0;
17923 goto __pyx_L0;
17924
17925 /* function exit code */
17926 __pyx_L1_error:;
17927 __Pyx_XDECREF(__pyx_t_1);
17928 __Pyx_AddTraceback("PyClical.tan", __pyx_clineno, __pyx_lineno, __pyx_filename);
17929 __pyx_r = NULL;
17930 __pyx_L0:;
17931 __Pyx_XGIVEREF(__pyx_r);
17932 __Pyx_RefNannyFinishContext();
17933 return __pyx_r;
17934 }
17935
17936 /* "PyClical.pyx":1818
17937 *
17938 *
17939 * cpdef inline atan(obj, i = None): # ««««««««
17940 * """
17941 * Inverse tangent of multivector with optional complexifier.
17942 */
17943
17944 static PyObject *__pyx_pw_8PyClical_75atan(PyObject *__pyx_self, PyObject *__pyx_args,
PyObject *__pyx_kwds); /*proto*/
17945 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_atan(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_atan *__pyx_optional_args) {
17946 PyObject *__pyx_v_i = (PyObject *)Py_None;
17947 PyObject *__pyx_r = NULL;
17948 __Pyx_RefNannyDeclarations
17949 int __pyx_t_1;
17950 int __pyx_t_2;
17951 PyObject *__pyx_t_3 = NULL;
17952 Clifford __pyx_t_4;
17953 PyObject *__pyx_t_5 = NULL;
17954 PyObject *__pyx_t_6 = NULL;
17955 PyObject *__pyx_t_7 = NULL;
17956 PyObject *__pyx_t_8 = NULL;
17957 PyObject *__pyx_t_9 = NULL;
17958 PyObject *__pyx_t_10 = NULL;
17959 PyObject *__pyx_t_11 = NULL;
17960 int __pyx_lineno = 0;
17961 const char *__pyx_filename = NULL;
17962 int __pyx_clineno = 0;
17963 __Pyx_RefNannySetupContext("atan", 0);
17964 if (__pyx_optional_args) {
17965 if (__pyx_optional_args->__pyx_n > 0) {
17966 __pyx_v_i = __pyx_optional_args->i;
17967 }
17968 }
17969
17970 /* "PyClical.pyx":1827
17971 * {1}
17972 * """
17973 * if not (i is None): # ««««««««
17974 * return clifford().wrap(glucat.atan(toClifford(obj), toClifford(i)))
17975 * else:
17976 */
17977 __pyx_t_1 = (__pyx_v_i != Py_None);
17978 __pyx_t_2 = (__pyx_t_1 != 0);
17979 if (__pyx_t_2) {
17980
17981 /* "PyClical.pyx":1828
17982 * """
17983 * if not (i is None):
17984 * return clifford().wrap(glucat.atan(toClifford(obj), toClifford(i))) # ««««««««
17985 * else:
17986 * try:
17987 */
17988 __Pyx_XDECREF(__pyx_r);
17989 __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1828, __pyx_L1_error)
17990 __Pyx_GOTREF(__pyx_t_3);
17991 try {
17992 __pyx_t_4 = atan(__pyx_f_8PyClical_toClifford(__pyx_v_obj),
__pyx_f_8PyClical_toClifford(__pyx_v_i));
17993 } catch (...) {
17994 __Pyx_CppExn2PyErr();

```

```

17995 __PYX_ERR(0, 1828, __pyx_L1_error)
17996 }
17997 __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1828, __pyx_L1_error)
17998 __Pyx_GOTREF(__pyx_t_5);
17999 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
18000 __pyx_r = __pyx_t_5;
18001 __pyx_t_5 = 0;
18002 goto __pyx_L0;
18003
18004 /* "PyClical.pyx":1827
18005 *
18006 * """
18007 * if not (i is None): # ««««««««
18008 * return clifford().wrap(glucat.atan(toClifford(obj), toClifford(i)))
18009 * else:
18010 */
18011 }
18012
18013 /* "PyClical.pyx":1830
18014 * return clifford().wrap(glucat.atan(toClifford(obj), toClifford(i)))
18015 * else:
18016 * try: # ««««««««
18017 * return math.atan(obj)
18018 * except:
18019 */
18020 /*else*/ {
18021 {
18022 __Pyx_PyThreadState_declare
18023 __Pyx_PyThreadState_assign
18024 __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
18025 __Pyx_XGOTREF(__pyx_t_6);
18026 __Pyx_XGOTREF(__pyx_t_7);
18027 __Pyx_XGOTREF(__pyx_t_8);
18028 /*try:*/ {
18029
18030 /* "PyClical.pyx":1831
18031 * else:
18032 * try:
18033 * return math.atan(obj) # ««««««««
18034 * except:
18035 * return clifford().wrap(glucat.atan(toClifford(obj)))
18036 */
18037 __Pyx_XDECREF(__pyx_r);
18038 __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1831, __pyx_L4_error)
18039 __Pyx_GOTREF(__pyx_t_3);
18040 __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_atan); if
(unlikely(!__pyx_t_9)) __PYX_ERR(0, 1831, __pyx_L4_error)
18041 __Pyx_GOTREF(__pyx_t_9);
18042 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
18043 __pyx_t_3 = NULL;
18044 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
18045 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
18046 if (likely(__pyx_t_3)) {
18047 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
18048 __Pyx_INCREF(__pyx_t_3);
18049 __Pyx_INCREF(function);
18050 __Pyx_DECREF_SET(__pyx_t_9, function);
18051 }
18052 }
18053 __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
18054 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
18055 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1831, __pyx_L4_error)
18056 __Pyx_GOTREF(__pyx_t_5);
18057 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
18058 __pyx_r = __pyx_t_5;
18059 __pyx_t_5 = 0;
18060 goto __pyx_L8_try_return;
18061
18062 /* "PyClical.pyx":1830
18063 * return clifford().wrap(glucat.atan(toClifford(obj), toClifford(i)))
18064 * else:
18065 * try: # ««««««««
18066 * return math.atan(obj)
18067 * except:
18068 */
18069 }
18070 __pyx_L4_error:;
18071 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
18072 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
18073 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
18074
18075 /* "PyClical.pyx":1832
18076 * try:
18077 * return math.atan(obj)

```

```

18078 * except: # ««««««««
18079 * return clifford().wrap(glucat.atan(toClifford(obj)))
18080 *
18081 */
18082
18083 /*except:*/ {
18084 __Pyx_AddTraceback("PyClical.atan", __pyx_clineno, __pyx_lineno,
__pyx_filename);
18085 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
1832, __pyx_L6_except_error)
18086 __Pyx_GOTREF(__pyx_t_5);
18087 __Pyx_GOTREF(__pyx_t_9);
18088 __Pyx_GOTREF(__pyx_t_3);
18089
18090 /* "PyClical.pyx":1833
18091 * return math.atan(obj)
18092 * except:
18093 * return clifford().wrap(glucat.atan(toClifford(obj))) # ««««««««
18094 * cpdef inline tanh(obj):
18095 */
18096 __Pyx_XDECREF(__pyx_r);
18097 __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1833,
__pyx_L6_except_error)
18098 __Pyx_GOTREF(__pyx_t_10);
18099 __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct
__pyx_obj_8PyClical_clifford *)__pyx_t_10), atan(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 1833, __pyx_L6_except_error)
18100 __Pyx_GOTREF(__pyx_t_11);
18101 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
18102 __pyx_r = __pyx_t_11;
18103 __pyx_t_11 = 0;
18104 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
18105 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
18106 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
18107 goto __pyx_L7_except_return;
18108 }
18109 __pyx_L6_except_error;;
18110
18111 /* "PyClical.pyx":1830
18112 * return clifford().wrap(glucat.atan(toClifford(obj), toClifford(i)))
18113 * else:
18114 * try: # ««««««««
18115 * return math.atan(obj)
18116 * except:
18117 */
18118 __Pyx_XGIVEREF(__pyx_t_6);
18119 __Pyx_XGIVEREF(__pyx_t_7);
18120 __Pyx_XGIVEREF(__pyx_t_8);
18121 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
18122 goto __pyx_L1_error;
18123 __pyx_L8_try_return:;
18124 __Pyx_XGIVEREF(__pyx_t_6);
18125 __Pyx_XGIVEREF(__pyx_t_7);
18126 __Pyx_XGIVEREF(__pyx_t_8);
18127 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
18128 goto __pyx_L0;
18129 __pyx_L7_except_return:;
18130 __Pyx_XGIVEREF(__pyx_t_6);
18131 __Pyx_XGIVEREF(__pyx_t_7);
18132 __Pyx_XGIVEREF(__pyx_t_8);
18133 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
18134 goto __pyx_L0;
18135 }
18136 }
18137
18138 /* "PyClical.pyx":1818
18139 * return clifford().wrap(glucat.tan(toClifford(obj)))
18140 *
18141 * cpdef inline atan(obj, i = None): # ««««««««
18142 * """
18143 * Inverse tangent of multivector with optional complexifier.
18144 */
18145
18146 /* function exit code */
18147 __pyx_L1_error:;
18148 __Pyx_XDECREF(__pyx_t_3);
18149 __Pyx_XDECREF(__pyx_t_5);
18150 __Pyx_XDECREF(__pyx_t_9);
18151 __Pyx_XDECREF(__pyx_t_10);
18152 __Pyx_XDECREF(__pyx_t_11);
18153 __Pyx_AddTraceback("PyClical.atan", __pyx_clineno, __pyx_lineno, __pyx_filename);
18154 __pyx_r = 0;
18155 __pyx_L0:;
18156 __Pyx_XGIVEREF(__pyx_r);
18157 __Pyx_RefNannyFinishContext();
18158 return __pyx_r;

```

```

18159 }
18160
18161 /* Python wrapper */
18162 static PyObject * __pyx_pw_8PyClical_75atan(PyObject * __pyx_self, PyObject * __pyx_args,
PyObject * __pyx_kwds); /*proto*/
18163 static char __pyx_doc_8PyClical_74atan[] = "\n Inverse tangent of multivector with
optional complexifier.\n\n >> s=index_set({1,2,3}); x=clifford(\"{1}\"); print(tan(atan(x,s),s))\n
{1}\n >> x=clifford(\"{1}\"); print(tan(atan(x)))\n {1}\n ";
18164 static PyObject * __pyx_pw_8PyClical_75atan(PyObject * __pyx_self, PyObject * __pyx_args,
PyObject * __pyx_kwds) {
18165 PyObject * __pyx_v_obj = 0;
18166 PyObject * __pyx_v_i = 0;
18167 int __pyx_lineno = 0;
18168 const char * __pyx_filename = NULL;
18169 int __pyx_clineno = 0;
18170 PyObject * __pyx_r = 0;
18171 __Pyx_RefNannyDeclarations
18172 __Pyx_RefNannySetupContext("atan (wrapper)", 0);
18173 {
18174 static PyObject ** __pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
18175 PyObject* values[2] = {0,0};
18176 values[1] = ((PyObject *)Py_None);
18177 if (unlikely(__pyx_kwds)) {
18178 Py_ssize_t kw_args;
18179 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
18180 switch (pos_args) {
18181 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
CYTHON_FALLTHROUGH;
18182 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
CYTHON_FALLTHROUGH;
18183 case 0: break;
18184 default: goto __pyx_L5_argtuple_error;
18185 }
18186 kw_args = PyDict_Size(__pyx_kwds);
18187 switch (pos_args) {
18188 case 0:
18189 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
18190
0)) kw_args--;
18192 else goto __pyx_L5_argtuple_error;
18193 CYTHON_FALLTHROUGH;
18194 case 1:
18195 if (kw_args > 0) {
18196 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
18197 if (value) { values[1] = value; kw_args--; }
18198 }
18199 }
18200 if (unlikely(kw_args > 0)) {
18201 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "atan") < 0)) __PYX_ERR(0, 1818, __pyx_L3_error)
18202 }
18203 } else {
18204 switch (PyTuple_GET_SIZE(__pyx_args)) {
18205 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
CYTHON_FALLTHROUGH;
18206 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
18207 break;
18208 default: goto __pyx_L5_argtuple_error;
18209 }
18210 }
18211 __pyx_v_obj = values[0];
18212 __pyx_v_i = values[1];
18213 }
18214 goto __pyx_L4_argument_unpacking_done;
18215 __pyx_L5_argtuple_error:;
18216 __Pyx_RaiseArgtupleInvalid("atan", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
18217 __PYX_ERR(0, 1818, __pyx_L3_error)
18218 __pyx_L3_error:;
18219 __Pyx_AddTraceback("PyClical.atan", __pyx_clineno, __pyx_lineno, __pyx_filename);
18220 __Pyx_RefNannyFinishContext();
18221 return NULL;
18222 __pyx_L4_argument_unpacking_done:;
18223 __pyx_r = __pyx_pf_8PyClical_74atan(__pyx_self, __pyx_v_obj, __pyx_v_i);
18224
18225 /* function exit code */
18226 __Pyx_RefNannyFinishContext();
18227 return __pyx_r;
18228 }
18229
18230 static PyObject * __pyx_pf_8PyClical_74atan(CYTHON_UNUSED PyObject * __pyx_self,
PyObject * __pyx_v_obj, PyObject * __pyx_v_i) {
18231 PyObject * __pyx_r = NULL;
18232 __Pyx_RefNannyDeclarations
18233 PyObject * __pyx_t_1 = NULL;
18234 struct __pyx_opt_args_8PyClical_atan __pyx_t_2;
18235 int __pyx_lineno = 0;
18236 const char * __pyx_filename = NULL;
18237 int __pyx_clineno = 0;

```

```

18238 __Pyx_RefNannySetupContext("atan", 0);
18239 __Pyx_XDECREF(__pyx_r);
18240 __pyx_t_2.__pyx_n = 1;
18241 __pyx_t_2.i = __pyx_v_i;
18242 __pyx_t_1 = __pyx_f_8PyClical_atan(__pyx_v_obj, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1818, __pyx_L1_error)
18243 __Pyx_GOTREF(__pyx_t_1);
18244 __pyx_r = __pyx_t_1;
18245 __pyx_t_1 = 0;
18246 goto __pyx_L0;
18247
18248 /* function exit code */
18249 __pyx_L1_error;;
18250 __Pyx_XDECREF(__pyx_t_1);
18251 __Pyx_AddTraceback("PyClical.atan", __pyx_clineno, __pyx_lineno, __pyx_filename);
18252 __pyx_r = NULL;
18253 __pyx_L0;;
18254 __Pyx_XGIVEREF(__pyx_r);
18255 __Pyx_RefNannyFinishContext();
18256 return __pyx_r;
18257 }
18258
18259 /* "PyClical.pyx":1835
18260 *
18261 * return clifford().wrap(glucat.atan(toClifford(obj)))
18262 * cpdef inline tanh(obj):
18263 * """
18264 * Hyperbolic tangent of multivector.
18265 */
18266
18267 static PyObject * __pyx_pw_8PyClical_77tanh(PyObject * __pyx_self, PyObject
*__pyx_v_obj); /*proto*/
18268 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_tanh(PyObject * __pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
18269 PyObject * __pyx_r = NULL;
18270 __Pyx_RefNannyDeclarations
18271 PyObject * __pyx_t_1 = NULL;
18272 PyObject * __pyx_t_2 = NULL;
18273 PyObject * __pyx_t_3 = NULL;
18274 PyObject * __pyx_t_4 = NULL;
18275 PyObject * __pyx_t_5 = NULL;
18276 PyObject * __pyx_t_6 = NULL;
18277 PyObject * __pyx_t_7 = NULL;
18278 PyObject * __pyx_t_8 = NULL;
18279 int __pyx_lineno = 0;
18280 const char * __pyx_filename = NULL;
18281 int __pyx_clineno = 0;
18282 __Pyx_RefNannySetupContext("tanh", 0);
18283
18284 /* "PyClical.pyx":1842
18285 * {1,2}
18286 * """
18287 * try:
18288 * return math.tanh(obj)
18289 * except:
18290 */
18291 {
18292 __Pyx_PyThreadState_declare
18293 __Pyx_PyThreadState_assign
18294 __Pyx_ExceptionSave(&__pyx_t_1, &__pyx_t_2, &__pyx_t_3);
18295 __Pyx_XGOTREF(__pyx_t_1);
18296 __Pyx_XGOTREF(__pyx_t_2);
18297 __Pyx_XGOTREF(__pyx_t_3);
18298 /*try:*/ {
18299
18300 /* "PyClical.pyx":1843
18301 * """
18302 * try:
18303 * return math.tanh(obj)
18304 * except:
18305 * return clifford().wrap(glucat.tanh(toClifford(obj)))
18306 */
18307 __Pyx_XDECREF(__pyx_r);
18308 __Pyx_GetModuleGlobalName(__pyx_t_5, __pyx_n_s_math); if (unlikely(!__pyx_t_5))
__PYX_ERR(0, 1843, __pyx_L3_error)
18309 __Pyx_GOTREF(__pyx_t_5);
18310 __pyx_t_6 = __Pyx_PyObject_GetAttrStr(__pyx_t_5, __pyx_n_s_tanh); if
(unlikely(!__pyx_t_6)) __PYX_ERR(0, 1843, __pyx_L3_error)
18311 __Pyx_GOTREF(__pyx_t_6);
18312 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
18313 __pyx_t_5 = NULL;
18314 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_6))) {
18315 __pyx_t_5 = PyMethod_GET_SELF(__pyx_t_6);
18316 if (likely(__pyx_t_5)) {
18317 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_6);
18318 __Pyx_INCREF(__pyx_t_5);
18319 __Pyx_INCREF(function);

```

```

18320 __Pyx_DECREF_SET(__pyx_t_6, function);
18321 }
18322 }
18323 __pyx_t_4 = (__pyx_t_5) ? __Pyx_PyObject_Call2Args(__pyx_t_6, __pyx_t_5,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_6, __pyx_v_obj);
18324 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
18325 if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1843, __pyx_L3_error)
18326 __Pyx_GOTREF(__pyx_t_4);
18327 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
18328 __pyx_r = __pyx_t_4;
18329 __pyx_t_4 = 0;
18330 goto __pyx_L7_try_return;
18331
18332 /* "PyClical.pyx":1842
18333 *
18334 * {1,2}
18335 * """
18336 * try:
18337 * return math.tanh(obj)
18338 * except:
18339 */
18340 __pyx_L3_error;;
18341 __Pyx_XDECREF(__pyx_t_4); __pyx_t_4 = 0;
18342 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
18343 __Pyx_XDECREF(__pyx_t_6); __pyx_t_6 = 0;
18344
18345 /* "PyClical.pyx":1844
18346 * try:
18347 * return math.tanh(obj)
18348 * except:
18349 * return clifford().wrap(glucat.tanh(toClifford(obj)))
18350 *
18351 */
18352 /*except:*/ {
18353 __Pyx_AddTraceback("PyClical.tanh", __pyx_clineno, __pyx_lineno,
__pyx_filename);
18354 if (__Pyx_GetException(&__pyx_t_4, &__pyx_t_6, &__pyx_t_5) < 0) __PYX_ERR(0,
1844, __pyx_L5_except_error)
18355 __Pyx_GOTREF(__pyx_t_4);
18356 __Pyx_GOTREF(__pyx_t_6);
18357 __Pyx_GOTREF(__pyx_t_5);
18358
18359 /* "PyClical.pyx":1845
18360 * return math.tanh(obj)
18361 * except:
18362 * return clifford().wrap(glucat.tanh(toClifford(obj)))
18363 *
18364 * cpdef inline atanh(obj,i = None):
18365 */
18366 __Pyx_XDECREF(__pyx_r);
18367 __pyx_t_7 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 1845,
__pyx_L5_except_error)
18368 __Pyx_GOTREF(__pyx_t_7);
18369 __pyx_t_8 = __pyx_f_8PyClical_8clifford_wrap(((struct
__pyx_obj_8PyClical_clifford *)__pyx_t_7), tanh(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_8)) __PYX_ERR(0, 1845, __pyx_L5_except_error)
18370 __Pyx_GOTREF(__pyx_t_8);
18371 __Pyx_DECREF(__pyx_t_7); __pyx_t_7 = 0;
18372 __pyx_r = __pyx_t_8;
18373 __pyx_t_8 = 0;
18374 __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
18375 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
18376 __Pyx_DECREF(__pyx_t_6); __pyx_t_6 = 0;
18377 goto __pyx_L6_except_return;
18378 }
18379 __pyx_L5_except_error;;
18380
18381 /* "PyClical.pyx":1842
18382 *
18383 * {1,2}
18384 * """
18385 * try:
18386 * return math.tanh(obj)
18387 * except:
18388 */
18389 __Pyx_XGIVEREF(__pyx_t_1);
18390 __Pyx_XGIVEREF(__pyx_t_2);
18391 __Pyx_XGIVEREF(__pyx_t_3);
18392 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
18393 goto __pyx_L1_error;
18394 __pyx_L7_try_return;
18395 __Pyx_XGIVEREF(__pyx_t_1);
18396 __Pyx_XGIVEREF(__pyx_t_2);
18397 __Pyx_XGIVEREF(__pyx_t_3);
18398 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
18399 goto __pyx_L0;
18400 __pyx_L6_except_return;

```



```

18400 __Pyx_XGIVEREF(__pyx_t_1);
18401 __Pyx_XGIVEREF(__pyx_t_2);
18402 __Pyx_XGIVEREF(__pyx_t_3);
18403 __Pyx_ExceptionReset(__pyx_t_1, __pyx_t_2, __pyx_t_3);
18404 goto __pyx_L0;
18405 }
18406
18407 /* "PyClical.pyx":1835
18408 * return clifford().wrap(glucat.atan(toClifford(obj)))
18409 *
18410 * cpdef inline tanh(obj):
18411 * """
18412 * Hyperbolic tangent of multivector.
18413 */
18414
18415 /* function exit code */
18416 __pyx_L1_error:;
18417 __Pyx_XDECREF(__pyx_t_4);
18418 __Pyx_XDECREF(__pyx_t_5);
18419 __Pyx_XDECREF(__pyx_t_6);
18420 __Pyx_XDECREF(__pyx_t_7);
18421 __Pyx_XDECREF(__pyx_t_8);
18422 __Pyx_AddTraceback("PyClical.tanh", __pyx_clineno, __pyx_lineno, __pyx_filename);
18423 __pyx_r = 0;
18424 __pyx_L0:;
18425 __Pyx_XGIVEREF(__pyx_r);
18426 __Pyx_RefNannyFinishContext();
18427 return __pyx_r;
18428 }
18429
18430 /* Python wrapper */
18431 static PyObject * __pyx_pw_8PyClical_77tanh(PyObject * __pyx_self, PyObject
*__pyx_v_obj); /*proto*/
18432 static char __pyx_doc_8PyClical_76tanh[] = "\n Hyperbolic tangent of
multivector.\n\n >> x=clifford(\"{1,2}\") * pi/4; print(tanh(x))\n {1,2}\n ";
18433 static PyObject * __pyx_pf_8PyClical_77tanh(PyObject * __pyx_self, PyObject
*__pyx_v_obj) {
18434 PyObject * __pyx_r = 0;
18435 __Pyx_RefNannyDeclarations
18436 __Pyx_RefNannySetupContext("tanh (wrapper)", 0);
18437 __pyx_r = __pyx_pf_8PyClical_76tanh(__pyx_self, ((PyObject *) __pyx_v_obj));
18438
18439 /* function exit code */
18440 __Pyx_RefNannyFinishContext();
18441 return __pyx_r;
18442 }
18443
18444 static PyObject * __pyx_pf_8PyClical_76tanh(CYTHON_UNUSED PyObject * __pyx_self,
PyObject * __pyx_v_obj) {
18445 PyObject * __pyx_r = NULL;
18446 __Pyx_RefNannyDeclarations
18447 PyObject * __pyx_t_1 = NULL;
18448 int __pyx_lineno = 0;
18449 const char * __pyx_filename = NULL;
18450 int __pyx_clineno = 0;
18451 __Pyx_RefNannySetupContext("tanh", 0);
18452 __Pyx_XDECREF(__pyx_r);
18453 __pyx_t_1 = __pyx_f_8PyClical_tanh(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1835, __pyx_L1_error)
18454 __Pyx_GOTREF(__pyx_t_1);
18455 __pyx_r = __pyx_t_1;
18456 __pyx_t_1 = 0;
18457 goto __pyx_L0;
18458
18459 /* function exit code */
18460 __pyx_L1_error:;
18461 __Pyx_XDECREF(__pyx_t_1);
18462 __Pyx_AddTraceback("PyClical.tanh", __pyx_clineno, __pyx_lineno, __pyx_filename);
18463 __pyx_r = NULL;
18464 __pyx_L0:;
18465 __Pyx_XGIVEREF(__pyx_r);
18466 __Pyx_RefNannyFinishContext();
18467 return __pyx_r;
18468 }
18469
18470 /* "PyClical.pyx":1847
18471 * return clifford().wrap(glucat.tanh(toClifford(obj)))
18472 *
18473 * cpdef inline atanh(obj,i = None):
18474 * """
18475 * Inverse hyperbolic tangent of multivector with optional complexifier.
18476 */
18477
18478 static PyObject * __pyx_pw_8PyClical_79atanh(PyObject * __pyx_self, PyObject
*__pyx_args, PyObject * __pyx_kws); /*proto*/
18479 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_atanh(PyObject * __pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch, struct __pyx_opt_args_8PyClical_atanh * __pyx_optional_args) {

```

```

18480 PyObject *__pyx_v_i = ((PyObject *)Py_None);
18481 PyObject *__pyx_r = NULL;
18482 __Pyx_RefNannyDeclarations
18483 int __pyx_t_1;
18484 int __pyx_t_2;
18485 PyObject *__pyx_t_3 = NULL;
18486 Clifford __pyx_t_4;
18487 PyObject *__pyx_t_5 = NULL;
18488 PyObject *__pyx_t_6 = NULL;
18489 PyObject *__pyx_t_7 = NULL;
18490 PyObject *__pyx_t_8 = NULL;
18491 PyObject *__pyx_t_9 = NULL;
18492 PyObject *__pyx_t_10 = NULL;
18493 PyObject *__pyx_t_11 = NULL;
18494 int __pyx_lineno = 0;
18495 const char *__pyx_filename = NULL;
18496 int __pyx_clineno = 0;
18497 __Pyx_RefNannySetupContext("atanh", 0);
18498 if (__pyx_optional_args) {
18499 if (__pyx_optional_args->__pyx_n > 0) {
18500 __pyx_v_i = __pyx_optional_args->i;
18501 }
18502 }
18503
18504 /* "PyClical.pyx":1856
18505 * {1,2}
18506 * """
18507 * if not (i is None): # ««««««««
18508 * return clifford().wrap(glucat.atanh(toClifford(obj), toClifford(i)))
18509 * else:
18510 */
18511 __pyx_t_1 = (__pyx_v_i != Py_None);
18512 __pyx_t_2 = (__pyx_t_1 != 0);
18513 if (__pyx_t_2) {
18514 /* "PyClical.pyx":1857
18515 * """
18516 * if not (i is None):
18517 * return clifford().wrap(glucat.atanh(toClifford(obj), toClifford(i)))
18518 * else:
18519 * try:
18520 * # ««««««««
18521 */
18522 __Pyx_XDECREF(__pyx_r);
18523 __pyx_t_3 = __Pyx_PyObject_CallNoArg(((PyObject
18524 *)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1857, __pyx_L1_error)
18525 __Pyx_GOTREF(__pyx_t_3);
18526 __pyx_t_4 = atanh(__pyx_f_8PyClical_toClifford(__pyx_v_obj),
18527 __pyx_f_8PyClical_toClifford(__pyx_v_i));
18528 } catch (...) {
18529 __Pyx_CppExn2PyErr();
18530 __PYX_ERR(0, 1857, __pyx_L1_error)
18531 }
18532 __pyx_t_5 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
18533 *)__pyx_t_3), __pyx_t_4); if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1857, __pyx_L1_error)
18534 __Pyx_GOTREF(__pyx_t_5);
18535 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
18536 __pyx_r = __pyx_t_5;
18537 __pyx_t_5 = 0;
18538 goto __pyx_L0;
18539 }
18540 /* "PyClical.pyx":1856
18541 * {1,2}
18542 * """
18543 * if not (i is None): # ««««««««
18544 * return clifford().wrap(glucat.atanh(toClifford(obj), toClifford(i)))
18545 * else:
18546 */
18547 }
18548 /* "PyClical.pyx":1859
18549 * return clifford().wrap(glucat.atanh(toClifford(obj), toClifford(i)))
18550 * else:
18551 * try:
18552 * # ««««««««
18553 * return math.atanh(obj)
18554 * except:
18555 */
18556 /*else*/ {
18557 {
18558 __Pyx_PyThreadState_declare
18559 __Pyx_PyThreadState_assign
18560 __Pyx_ExceptionSave(&__pyx_t_6, &__pyx_t_7, &__pyx_t_8);
18561 __Pyx_XGOTREF(__pyx_t_6);
18562 __Pyx_XGOTREF(__pyx_t_7);
18563 __Pyx_XGOTREF(__pyx_t_8);
18564 }
18565 /*try:*/ {

```

```

18563
18564 /* "PyClical.pyx":1860
18565 * else:
18566 * try:
18567 * return math.atanh(obj) # ««««««««
18568 * except:
18569 * return clifford().wrap(glucat.atanh(toClifford(obj)))
18570 */
18571 __Pyx_XDECREF(__pyx_r);
18572 __Pyx_GetModuleGlobalName(__pyx_t_3, __pyx_n_s_math); if
(unlikely(!__pyx_t_3)) __PYX_ERR(0, 1860, __pyx_L4_error)
18573 __Pyx_GOTREF(__pyx_t_3);
18574 __pyx_t_9 = __Pyx_PyObject_GetAttrStr(__pyx_t_3, __pyx_n_s_atanh); if
(unlikely(!__pyx_t_9)) __PYX_ERR(0, 1860, __pyx_L4_error)
18575 __Pyx_GOTREF(__pyx_t_9);
18576 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
18577 __pyx_t_3 = NULL;
18578 if (CYTHON_UNPACK_METHODS && unlikely(PyMethod_Check(__pyx_t_9))) {
18579 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_9);
18580 if (likely(__pyx_t_3)) {
18581 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_9);
18582 __Pyx_INCREF(__pyx_t_3);
18583 __Pyx_INCREF(function);
18584 __Pyx_DECREF_SET(__pyx_t_9, function);
18585 }
18586 }
18587 __pyx_t_5 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_9, __pyx_t_3,
__pyx_v_obj) : __Pyx_PyObject_CallOneArg(__pyx_t_9, __pyx_v_obj);
18588 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
18589 if (unlikely(!__pyx_t_5)) __PYX_ERR(0, 1860, __pyx_L4_error)
18590 __Pyx_GOTREF(__pyx_t_5);
18591 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;
18592 __pyx_r = __pyx_t_5;
18593 __pyx_t_5 = 0;
18594 goto __pyx_L8_try_return;
18595
18596 /* "PyClical.pyx":1859
18597 * return clifford().wrap(glucat.atanh(toClifford(obj), toClifford(i)))
18598 * else:
18599 * try:
18600 * return math.atanh(obj)
18601 * except:
18602 */
18603 }
18604 __pyx_L4_error:;
18605 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
18606 __Pyx_XDECREF(__pyx_t_5); __pyx_t_5 = 0;
18607 __Pyx_XDECREF(__pyx_t_9); __pyx_t_9 = 0;
18608
18609 /* "PyClical.pyx":1861
18610 * try:
18611 * return math.atanh(obj)
18612 * except:
18613 * return clifford().wrap(glucat.atanh(toClifford(obj)))
18614 *
18615 */
18616 /*except:*/ {
18617 __Pyx_AddTraceback("PyClical.atanh", __pyx_clineno, __pyx_lineno,
__pyx_filename);
18618 if (__Pyx_GetException(&__pyx_t_5, &__pyx_t_9, &__pyx_t_3) < 0) __PYX_ERR(0,
1861, __pyx_L6_except_error)
18619 __Pyx_GOTREF(__pyx_t_5);
18620 __Pyx_GOTREF(__pyx_t_9);
18621 __Pyx_GOTREF(__pyx_t_3);
18622
18623 /* "PyClical.pyx":1862
18624 * return math.atanh(obj)
18625 * except:
18626 * return clifford().wrap(glucat.atanh(toClifford(obj)))
18627 *
18628 * cpdef inline random_clifford(index_set ixt, fill = 1.0):
18629 */
18630 __Pyx_XDECREF(__pyx_r);
18631 __pyx_t_10 = __Pyx_PyObject_CallNoArg(((PyObject
*)__pyx_ptype_8PyClical_clifford)); if (unlikely(!__pyx_t_10)) __PYX_ERR(0, 1862,
__pyx_L6_except_error)
18632 __Pyx_GOTREF(__pyx_t_10);
18633 __pyx_t_11 = __pyx_f_8PyClical_8clifford_wrap(((struct
__pyx_obj_8PyClical_clifford *)__pyx_t_10), atanh(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if
(unlikely(!__pyx_t_11)) __PYX_ERR(0, 1862, __pyx_L6_except_error)
18634 __Pyx_GOTREF(__pyx_t_11);
18635 __Pyx_DECREF(__pyx_t_10); __pyx_t_10 = 0;
18636 __pyx_r = __pyx_t_11;
18637 __pyx_t_11 = 0;
18638 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
18639 __Pyx_DECREF(__pyx_t_5); __pyx_t_5 = 0;
18640 __Pyx_DECREF(__pyx_t_9); __pyx_t_9 = 0;

```

```

18641 goto __pyx_L7_except_return;
18642 }
18643 __pyx_L6_except_error;;
18644
18645 /* "PyClical.pyx":1859
18646 * return clifford().wrap(glucat.atanh(toClifford(obj), toClifford(i)))
18647 * else:
18648 * try:
18649 * return math.atanh(obj)
18650 * except:
18651 */
18652 __Pyx_XGIVEREF(__pyx_t_6);
18653 __Pyx_XGIVEREF(__pyx_t_7);
18654 __Pyx_XGIVEREF(__pyx_t_8);
18655 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
18656 goto __pyx_L1_error;
18657 __pyx_L8_try_return;;
18658 __Pyx_XGIVEREF(__pyx_t_6);
18659 __Pyx_XGIVEREF(__pyx_t_7);
18660 __Pyx_XGIVEREF(__pyx_t_8);
18661 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
18662 goto __pyx_L0;
18663 __pyx_L7_except_return;;
18664 __Pyx_XGIVEREF(__pyx_t_6);
18665 __Pyx_XGIVEREF(__pyx_t_7);
18666 __Pyx_XGIVEREF(__pyx_t_8);
18667 __Pyx_ExceptionReset(__pyx_t_6, __pyx_t_7, __pyx_t_8);
18668 goto __pyx_L0;
18669 }
18670
18671
18672 /* "PyClical.pyx":1847
18673 * return clifford().wrap(glucat.tanh(toClifford(obj)))
18674 *
18675 * cpdef inline atanh(obj,i = None):
18676 * """
18677 * Inverse hyperbolic tangent of multivector with optional complexifier.
18678 */
18679
18680 /* function exit code */
18681 __pyx_L1_error;;
18682 __Pyx_XDECREF(__pyx_t_3);
18683 __Pyx_XDECREF(__pyx_t_5);
18684 __Pyx_XDECREF(__pyx_t_9);
18685 __Pyx_XDECREF(__pyx_t_10);
18686 __Pyx_XDECREF(__pyx_t_11);
18687 __Pyx_AddTraceback("PyClical.atanh", __pyx_clineno, __pyx_lineno, __pyx_filename);
18688 __pyx_r = 0;
18689 __pyx_L0;;
18690 __Pyx_XGIVEREF(__pyx_r);
18691 __Pyx_RefNannyFinishContext();
18692 return __pyx_r;
18693 }
18694
18695 /* Python wrapper */
18696 static PyObject * __pyx_pw_8PyClical_79atanh(PyObject * __pyx_self, PyObject
18697 * __pyx_args, PyObject * __pyx_kwds); /*proto*/
18698 static char __pyx_doc_8PyClical_78atanh[] = "\n Inverse hyperbolic tangent of
multivector with optional complexifier.\n\n >> s=index_set({1,2,3}); x=clifford(\"{1,2}\");
print(tanh(atanh(x,s)))\n {1,2}\n >> x=clifford(\"{1,2}\"); print(tanh(atanh(x)))\n {1,2}\n
";
18699 static PyObject * __pyx_pw_8PyClical_79atanh(PyObject * __pyx_self, PyObject
18700 * __pyx_args, PyObject * __pyx_kwds) {
18701 PyObject * __pyx_v_obj = 0;
18702 PyObject * __pyx_v_i = 0;
18703 int __pyx_lineno = 0;
18704 const char * __pyx_filename = NULL;
18705 int __pyx_clineno = 0;
18706 PyObject * __pyx_r = 0;
18707 __Pyx_RefNannyDeclarations
18708 __Pyx_RefNannySetupContext("atanh (wrapper)", 0);
18709 {
18710 static PyObject ** __pyx_pyargnames[] = {&__pyx_n_s_obj,&__pyx_n_s_i,0};
18711 PyObject* values[2] = {0,0};
18712 values[1] = ((PyObject *)Py_None);
18713 if (unlikely(__pyx_kwds)) {
18714 Py_ssize_t kw_args;
18715 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
18716 switch (pos_args) {
18717 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
CYTHON_FALLTHROUGH;
18718 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
CYTHON_FALLTHROUGH;
18719 case 0: break;
18720 default: goto __pyx_L5_argtuple_error;
18721 }
18722 kw_args = PyDict_Size(__pyx_kwds);

```

```

18723 switch (pos_args) {
18724 case 0:
18725 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_obj)) !=
0)) kw_args--;
18726 else goto __pyx_L5_argtuple_error;
18727 CYTHON_FALLTHROUGH;
18728 case 1:
18729 if (kw_args > 0) {
18730 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_i);
18731 if (value) { values[1] = value; kw_args--; }
18732 }
18733 }
18734 if (unlikely(kw_args > 0)) {
18735 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "atanh") < 0)) __PYX_ERR(0, 1847, __pyx_L3_error)
18736 }
18737 } else {
18738 switch (PyTuple_GET_SIZE(__pyx_args)) {
18739 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
18740 CYTHON_FALLTHROUGH;
18741 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
18742 break;
18743 default: goto __pyx_L5_argtuple_error;
18744 }
18745 }
18746 __pyx_v_obj = values[0];
18747 __pyx_v_i = values[1];
18748 }
18749 goto __pyx_L4_argument_unpacking_done;
18750 __pyx_L5_argtuple_error:;
18751 __Pyx_RaiseArgtupleInvalid("atanh", 0, 1, 2, PyTuple_GET_SIZE(__pyx_args));
__PYX_ERR(0, 1847, __pyx_L3_error)
18752 __pyx_L3_error:;
18753 __Pyx_AddTraceback("PyClical.atanh", __pyx_clineno, __pyx_lineno, __pyx_filename);
18754 __Pyx_RefNannyFinishContext();
18755 return NULL;
18756 __pyx_L4_argument_unpacking_done:;
18757 __pyx_r = __pyx_pf_8PyClical_78atanh(__pyx_self, __pyx_v_obj, __pyx_v_i);
18758
18759 /* function exit code */
18760 __Pyx_RefNannyFinishContext();
18761 return __pyx_r;
18762 }
18763
18764 static PyObject * __pyx_pf_8PyClical_78atanh(CYTHON_UNUSED PyObject * __pyx_self,
PyObject * __pyx_v_obj, PyObject * __pyx_v_i) {
18765 PyObject * __pyx_r = NULL;
18766 __Pyx_RefNannyDeclarations
18767 PyObject * __pyx_t_1 = NULL;
18768 struct __pyx_opt_args_8PyClical_atanh __pyx_t_2;
18769 int __pyx_lineno = 0;
18770 const char * __pyx_filename = NULL;
18771 int __pyx_clineno = 0;
18772 __Pyx_RefNannySetupContext("atanh", 0);
18773 __Pyx_XDECREF(__pyx_r);
18774 __pyx_t_2.__pyx_n = 1;
18775 __pyx_t_2.i = __pyx_v_i;
18776 __pyx_t_1 = __pyx_f_8PyClical_atanh(__pyx_v_obj, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1847, __pyx_L1_error)
18777 __Pyx_GOTREF(__pyx_t_1);
18778 __pyx_r = __pyx_t_1;
18779 __pyx_t_1 = 0;
18780 goto __pyx_L0;
18781
18782 /* function exit code */
18783 __pyx_L1_error:;
18784 __Pyx_XDECREF(__pyx_t_1);
18785 __Pyx_AddTraceback("PyClical.atanh", __pyx_clineno, __pyx_lineno, __pyx_filename);
18786 __pyx_r = NULL;
18787 __pyx_L0:;
18788 __Pyx_XGIVEREF(__pyx_r);
18789 __Pyx_RefNannyFinishContext();
18790 return __pyx_r;
18791 }
18792
18793 /* "PyClical.pyx":1864
18794 *
18795 * return clifford().wrap(glucat.atanh(toClifford(obj)))
18796 * cpdef inline random_clifford(index_set ixt, fill = 1.0):
18797 * """
18798 * Random multivector within a frame.
18799 */
18800
18801 static PyObject * __pyx_pw_8PyClical_81random_clifford(PyObject * __pyx_self, PyObject
* __pyx_args, PyObject * __pyx_kwds); /*proto*/
18802 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_random_clifford(struct
__pyx_obj_8PyClical_index_set * __pyx_v_ixt, CYTHON_UNUSED int __pyx_skip_dispatch, struct

```

```

__pyx_opt_args_8PyClicl_random_clifford *__pyx_optional_args) {
18803 PyObject *__pyx_v_fill = ((PyObject *)__pyx_float_1_0);
18804 PyObject *__pyx_r = NULL;
18805 __Pyx_RefNannyDeclarations
18806 PyObject *__pyx_t_1 = NULL;
18807 PyObject *__pyx_t_2 = NULL;
18808 scalar_t __pyx_t_3;
18809 PyObject *__pyx_t_4 = NULL;
18810 int __pyx_lineno = 0;
18811 const char *__pyx_filename = NULL;
18812 int __pyx_clineno = 0;
18813 __Pyx_RefNannySetupContext("random_clifford", 0);
18814 if (__pyx_optional_args) {
18815 if (__pyx_optional_args->__pyx_n > 0) {
18816 __pyx_v_fill = __pyx_optional_args->fill;
18817 }
18818 }
18819
18820 /* "PyClicl.pyx":1871
18821 * {-3,-1,2}
18822 * """
18823 * return clifford().wrap(clifford().instance.random(ixt.unwrap(), <scalar_t>fill))
18824 * # ««««««««
18825 * cpdef inline cga3(obj):
18826 */
18827 __Pyx_XDECREF(__pyx_r);
18828 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClicl_clifford));
18829 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1871, __pyx_L1_error)
18830 __Pyx_GOTREF(__pyx_t_1);
18831 __pyx_t_2 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClicl_clifford));
18832 if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1871, __pyx_L1_error)
18833 __Pyx_GOTREF(__pyx_t_2);
18834 __pyx_t_3 = __pyx_PyFloat_AsDouble(__pyx_v_fill); if (unlikely((__pyx_t_3 ==
18835 ((scalar_t)-1)) && PyErr_Occurred())) __PYX_ERR(0, 1871, __pyx_L1_error)
18836 __pyx_t_4 = __pyx_f_8PyClicl_8clifford_wrap(((struct __pyx_obj_8PyClicl_clifford
18837 *)__pyx_t_1), ((struct __pyx_obj_8PyClicl_clifford
18838 *)__pyx_t_2)->instance->random(__pyx_f_8PyClicl_9index_set_unwrap(__pyx_v_ixt),
18839 ((scalar_t)__pyx_t_3))); if (unlikely(!__pyx_t_4)) __PYX_ERR(0, 1871, __pyx_L1_error)
18840 __Pyx_GOTREF(__pyx_t_4);
18841 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
18842 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
18843 __pyx_r = __pyx_t_4;
18844 __pyx_t_4 = 0;
18845 goto __pyx_L0;
18846
18847 /* "PyClicl.pyx":1864
18848 * return clifford().wrap(glucat.atanh(toClifford(obj)))
18849 * # ««««««««
18850 * cpdef inline random_clifford(index_set ixt, fill = 1.0):
18851 * """
18852 * Random multivector within a frame.
18853 */
18854
18855 /* function exit code */
18856 __pyx_L1_error:;
18857 __Pyx_XDECREF(__pyx_t_1);
18858 __Pyx_XDECREF(__pyx_t_2);
18859 __Pyx_XDECREF(__pyx_t_4);
18860 __Pyx_AddTraceback("PyClicl.random_clifford", __pyx_clineno, __pyx_lineno,
18861 __pyx_filename);
18862 __pyx_r = 0;
18863 __pyx_L0:;
18864 __Pyx_XGIVEREF(__pyx_r);
18865 __Pyx_RefNannyFinishContext();
18866 return __pyx_r;
18867 }
18868
18869 /* Python wrapper */
18870 static PyObject *__pyx_pw_8PyClicl_81random_clifford(PyObject *__pyx_self, PyObject
18871 *__pyx_args, PyObject *__pyx_kwds); /*proto*/
18872 static char __pyx_doc_8PyClicl_80random_clifford[] = "\n Random multivector within
18873 a frame.\n\n >> print(random_clifford(index_set({-3,-1,2})).frame())\n {-3,-1,2}\n ";
18874 static PyObject *__pyx_pw_8PyClicl_81random_clifford(PyObject *__pyx_self, PyObject
18875 *__pyx_args, PyObject *__pyx_kwds) {
18876 struct __pyx_obj_8PyClicl_index_set *__pyx_v_ixt = 0;
18877 PyObject *__pyx_v_fill = 0;
18878 int __pyx_lineno = 0;
18879 const char *__pyx_filename = NULL;
18880 int __pyx_clineno = 0;
18881 PyObject *__pyx_r = 0;
18882 __Pyx_RefNannyDeclarations
18883 __Pyx_RefNannySetupContext("random_clifford (wrapper)", 0);
18884 {
18885 static PyObject **__pyx_pyargnames[] = {&__pyx_n_s_ixt,&__pyx_n_s_fill,0};
18886 PyObject* values[2] = {0,0};
18887 values[1] = ((PyObject *)__pyx_float_1_0);

```

```

18878 if (unlikely(__pyx_kwds)) {
18879 Py_ssize_t kw_args;
18880 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
18881 switch (pos_args) {
18882 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
18883 CYTHON_FALLTHROUGH;
18884 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
18885 CYTHON_FALLTHROUGH;
18886 case 0: break;
18887 default: goto __pyx_L5_argtuple_error;
18888 }
18889 kw_args = PyDict_Size(__pyx_kwds);
18890 switch (pos_args) {
18891 case 0:
18892 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_ixt)) !=
0)) kw_args--;
18893 else goto __pyx_L5_argtuple_error;
18894 CYTHON_FALLTHROUGH;
18895 case 1:
18896 if (kw_args > 0) {
18897 PyObject* value = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_fill);
18898 if (value) { values[1] = value; kw_args--; }
18899 }
18900 }
18901 if (unlikely(kw_args > 0)) {
18902 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
values, pos_args, "random_clifford") < 0)) __PYX_ERR(0, 1864, __pyx_L3_error)
18903 }
18904 } else {
18905 switch (PyTuple_GET_SIZE(__pyx_args)) {
18906 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
18907 CYTHON_FALLTHROUGH;
18908 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
18909 break;
18910 default: goto __pyx_L5_argtuple_error;
18911 }
18912 }
18913 __pyx_v_ixt = ((struct __pyx_obj_8PyClical_index_set *)values[0]);
18914 __pyx_v_fill = values[1];
18915 }
18916 goto __pyx_L4_argument_unpacking_done;
18917 __pyx_L5_argtuple_error:;
18918 __Pyx_RaiseArgtupleInvalid("random_clifford", 0, 1, 2,
PyTuple_GET_SIZE(__pyx_args)); __PYX_ERR(0, 1864, __pyx_L3_error)
18919 __pyx_L3_error:;
18920 __Pyx_AddTraceback("PyClical.random_clifford", __pyx_clineno, __pyx_lineno,
__pyx_filename);
18921 __Pyx_RefNannyFinishContext();
18922 return NULL;
18923 __pyx_L4_argument_unpacking_done:;
18924 if (unlikely(!__Pyx_ArgTypeTest(((PyObject *)__pyx_v_ixt),
__pyx_ptype_8PyClical_index_set, 1, "ixt", 0))) __PYX_ERR(0, 1864, __pyx_L1_error)
18925 __pyx_r = __pyx_pf_8PyClical_80random_clifford(__pyx_self, __pyx_v_ixt,
__pyx_v_fill);
18926
18927 /* function exit code */
18928 goto __pyx_L0;
18929 __pyx_L1_error:;
18930 __pyx_r = NULL;
18931 __pyx_L0:;
18932 __Pyx_RefNannyFinishContext();
18933 return __pyx_r;
18934 }
18935
18936 static PyObject * __pyx_pf_8PyClical_80random_clifford(CYTHON_UNUSED PyObject
*__pyx_self, struct __pyx_obj_8PyClical_index_set *__pyx_v_ixt, PyObject *__pyx_v_fill) {
18937 PyObject *__pyx_r = NULL;
18938 __Pyx_RefNannyDeclarations
18939 PyObject *__pyx_t_1 = NULL;
18940 struct __pyx_opt_args_8PyClical_random_clifford __pyx_t_2;
18941 int __pyx_lineno = 0;
18942 const char *__pyx_filename = NULL;
18943 int __pyx_clineno = 0;
18944 __Pyx_RefNannySetupContext("random_clifford", 0);
18945 __Pyx_XDECREF(__pyx_r);
18946 __pyx_t_2.__pyx_n = 1;
18947 __pyx_t_2.fill = __pyx_v_fill;
18948 __pyx_t_1 = __pyx_f_8PyClical_random_clifford(__pyx_v_ixt, 0, &__pyx_t_2); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1864, __pyx_L1_error)
18949 __Pyx_GOTREF(__pyx_t_1);
18950 __pyx_r = __pyx_t_1;
18951 __pyx_t_1 = 0;
18952 goto __pyx_L0;
18953
18954 /* function exit code */
18955 __pyx_L1_error:;
18956 __Pyx_XDECREF(__pyx_t_1);

```

```

18957 __Pyx_AddTraceback("PyClical.random_clifford", __pyx_clineno, __pyx_lineno,
__pyx_filename);
18958 __pyx_r = NULL;
18959 __pyx_L0:;
18960 __Pyx_XGIVEREF(__pyx_r);
18961 __Pyx_RefNannyFinishContext();
18962 return __pyx_r;
18963 }
18964
18965 /* "PyClical.pyx":1873
18966 * return clifford().wrap(clifford().instance.random(ixt.unwrap(), <scalar_t>fill))
18967 *
18968 * cpdef inline cga3(obj):
18969 * """
18970 * Convert Euclidean 3D multivector to Conformal Geometric Algebra using Doran and Lasenby
definition.
18971 */
18972
18973 static PyObject * __pyx_pw_8PyClical_83cga3(PyObject * __pyx_self, PyObject
* __pyx_v_obj); /*proto*/
18974 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_cga3(PyObject * __pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
18975 PyObject * __pyx_r = NULL;
18976 __Pyx_RefNannyDeclarations
18977 PyObject * __pyx_t_1 = NULL;
18978 PyObject * __pyx_t_2 = NULL;
18979 int __pyx_lineno = 0;
18980 const char * __pyx_filename = NULL;
18981 int __pyx_clineno = 0;
18982 __Pyx_RefNannySetupContext("cga3", 0);
18983
18984 /* "PyClical.pyx":1880
18985 * 87{-1}+4{1}+18{2}+2{3}+85{4}
18986 * """
18987 * return clifford().wrap(glucat.cga3(toClifford(obj)))
18988 *
18989 * cpdef inline cga3std(obj):
18990 */
18991 __Pyx_XDECREF(__pyx_r);
18992 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *) __pyx_ptype_8PyClical_clifford));
18993 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1880, __pyx_L1_error)
18994 __Pyx_GOTREF(__pyx_t_1);
18995 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*) __pyx_t_1), cga3::cga3(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_2))
__PYX_ERR(0, 1880, __pyx_L1_error)
18996 __Pyx_GOTREF(__pyx_t_2);
18997 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
18998 __pyx_r = __pyx_t_2;
18999 __pyx_t_2 = 0;
19000 goto __pyx_L0;
19001
19002 /* "PyClical.pyx":1873
19003 * return clifford().wrap(clifford().instance.random(ixt.unwrap(), <scalar_t>fill))
19004 *
19005 * cpdef inline cga3(obj):
19006 * """
19007 * Convert Euclidean 3D multivector to Conformal Geometric Algebra using Doran and Lasenby
definition.
19008 */
19009
19010 /* function exit code */
19011 __pyx_L1_error:;
19012 __Pyx_XDECREF(__pyx_t_1);
19013 __Pyx_XDECREF(__pyx_t_2);
19014 __Pyx_AddTraceback("PyClical.cga3", __pyx_clineno, __pyx_lineno, __pyx_filename);
19015 __pyx_r = 0;
19016 __pyx_L0:;
19017 __Pyx_XGIVEREF(__pyx_r);
19018 __Pyx_RefNannyFinishContext();
19019 return __pyx_r;
19020 }
19021
19022 /* Python wrapper */
19023 static PyObject * __pyx_pw_8PyClical_83cga3(PyObject * __pyx_self, PyObject
* __pyx_v_obj); /*proto*/
19024 static char __pyx_doc_8PyClical_82cga3[] = "\n Convert Euclidean 3D multivector to
Conformal Geometric Algebra using Doran and Lasenby definition.\n\n >>
x=clifford(\"2{1}+9{2}+{3}\"); print(cga3(x))\n 87{-1}+4{1}+18{2}+2{3}+85{4}\n ";
19025 static PyObject * __pyx_pw_8PyClical_83cga3(PyObject * __pyx_self, PyObject
* __pyx_v_obj) {
19026 PyObject * __pyx_r = 0;
19027 __Pyx_RefNannyDeclarations
19028 __Pyx_RefNannySetupContext("cga3 (wrapper)", 0);
19029 __pyx_r = __pyx_f_8PyClical_82cga3(__pyx_self, ((PyObject *) __pyx_v_obj));
19030
19031 /* function exit code */
__Pyx_RefNannyFinishContext();

```



```

19032 return __pyx_r;
19033 }
19034
19035 static PyObject *__pyx_pf_8PyClical_82cga3(CYTHON_UNUSED PyObject *__pyx_self,
PyObject *__pyx_v_obj) {
19036 PyObject *__pyx_r = NULL;
19037 __Pyx_RefNannyDeclarations
19038 PyObject *__pyx_t_1 = NULL;
19039 int __pyx_lineno = 0;
19040 const char *__pyx_filename = NULL;
19041 int __pyx_clineno = 0;
19042 __Pyx_RefNannySetupContext("cga3", 0);
19043 __Pyx_XDECREF(__pyx_r);
19044 __pyx_t_1 = __pyx_f_8PyClical_cga3(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1873, __pyx_L1_error)
19045 __Pyx_GOTREF(__pyx_t_1);
19046 __pyx_r = __pyx_t_1;
19047 __pyx_t_1 = 0;
19048 goto __pyx_L0;
19049
19050 /* function exit code */
19051 __pyx_L1_error++;
19052 __Pyx_XDECREF(__pyx_t_1);
19053 __Pyx_AddTraceback("PyClical.cga3", __pyx_clineno, __pyx_lineno, __pyx_filename);
19054 __pyx_r = NULL;
19055 __pyx_L0++;
19056 __Pyx_XGIVEREF(__pyx_r);
19057 __Pyx_RefNannyFinishContext();
19058 return __pyx_r;
19059 }
19060
19061 /* "PyClical.pyx":1882
19062 * return clifford().wrap(glucat.cga3(toClifford(obj)))
19063 *
19064 * cpdef inline cga3std(obj):
19065 * """
19066 * Convert CGA3 null vector to standard conformal null vector using Doran and Lasenby definition.
19067 */
19068
19069 static PyObject *__pyx_pw_8PyClical_85cga3std(PyObject *__pyx_self, PyObject
*__pyx_v_obj); /*proto*/
19070 static CYTHON_INLINE PyObject *__pyx_f_8PyClical_cga3std(PyObject *__pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
19071 PyObject *__pyx_r = NULL;
19072 __Pyx_RefNannyDeclarations
19073 PyObject *__pyx_t_1 = NULL;
19074 PyObject *__pyx_t_2 = NULL;
19075 int __pyx_lineno = 0;
19076 const char *__pyx_filename = NULL;
19077 int __pyx_clineno = 0;
19078 __Pyx_RefNannySetupContext("cga3std", 0);
19079
19080 /* "PyClical.pyx":1891
19081 * 0
19082 * """
19083 * return clifford().wrap(glucat.cga3std(toClifford(obj)))
19084 *
19085 * cpdef inline agc3(obj):
19086 */
19087 __Pyx_XDECREF(__pyx_r);
19088 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *)__pyx_ptype_8PyClical_clifford));
19089 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1891, __pyx_L1_error)
19090 __Pyx_GOTREF(__pyx_t_1);
19091 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*)__pyx_t_1), cga3::cga3std(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_2))
__PYX_ERR(0, 1891, __pyx_L1_error)
19092 __Pyx_GOTREF(__pyx_t_2);
19093 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
19094 __pyx_r = __pyx_t_2;
19095 __pyx_t_2 = 0;
19096 goto __pyx_L0;
19097
19098 /* "PyClical.pyx":1882
19099 * return clifford().wrap(glucat.cga3(toClifford(obj)))
19100 *
19101 * cpdef inline cga3std(obj):
19102 * """
19103 * Convert CGA3 null vector to standard conformal null vector using Doran and Lasenby definition.
19104 */
19105
19106 /* function exit code */
19107 __pyx_L1_error++;
19108 __Pyx_XDECREF(__pyx_t_1);
19109 __Pyx_XDECREF(__pyx_t_2);
19110 __Pyx_AddTraceback("PyClical.cga3std", __pyx_clineno, __pyx_lineno, __pyx_filename);
19111 __pyx_r = 0;
19112 __pyx_L0++;

```

```

19112 __Pyx_XGIVEREF(__pyx_r);
19113 __Pyx_RefNannyFinishContext();
19114 return __pyx_r;
19115 }
19116
19117 /* Python wrapper */
19118 static PyObject * __pyx_pw_8PyClical_85cga3std(PyObject * __pyx_self, PyObject
*__pyx_v_obj); /*proto*/
19119 static char __pyx_doc_8PyClical_84cga3std[] = "\n Convert CGA3 null vector to
standard conformal null vector using Doran and Lasenby definition.\n\n >>
x=clifford(\"2{1}+9{2}+{3}\"); print(cga3std(cga3(x)))\n 87{-1}+4{1}+18{2}+2{3}+85{4}\n >>
x=clifford(\"2{1}+9{2}+{3}\"); print(cga3std(cga3(x))-cga3(x))\n 0\n ";
19120 static PyObject * __pyx_pf_8PyClical_85cga3std(PyObject * __pyx_self, PyObject
*__pyx_v_obj) {
19121 PyObject * __pyx_r = 0;
19122 __Pyx_RefNannyDeclarations
19123 __Pyx_RefNannySetupContext("cga3std (wrapper)", 0);
19124 __pyx_r = __pyx_pf_8PyClical_84cga3std(__pyx_self, ((PyObject *) __pyx_v_obj));
19125
19126 /* function exit code */
19127 __Pyx_RefNannyFinishContext();
19128 return __pyx_r;
19129 }
19130
19131 static PyObject * __pyx_pf_8PyClical_84cga3std(CYTHON_UNUSED PyObject * __pyx_self,
PyObject * __pyx_v_obj) {
19132 PyObject * __pyx_r = NULL;
19133 __Pyx_RefNannyDeclarations
19134 PyObject * __pyx_t_1 = NULL;
19135 int __pyx_lineno = 0;
19136 const char * __pyx_filename = NULL;
19137 int __pyx_clineno = 0;
19138 __Pyx_RefNannySetupContext("cga3std", 0);
19139 __Pyx_XDECREf(__pyx_r);
19140 __pyx_t_1 = __pyx_f_8PyClical_cga3std(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1882, __pyx_L1_error)
19141 __Pyx_GOTREF(__pyx_t_1);
19142 __pyx_r = __pyx_t_1;
19143 __pyx_t_1 = 0;
19144 goto __pyx_L0;
19145
19146 /* function exit code */
19147 __pyx_L1_error:;
19148 __Pyx_XDECREf(__pyx_t_1);
19149 __Pyx_AddTraceback("PyClical.cga3std", __pyx_clineno, __pyx_lineno, __pyx_filename);
19150 __pyx_r = NULL;
19151 __pyx_L0:;
19152 __Pyx_XGIVEREF(__pyx_r);
19153 __Pyx_RefNannyFinishContext();
19154 return __pyx_r;
19155 }
19156
19157 /* "PyClical.pyx":1893
19158 * return clifford().wrap(glucat.cga3std(toClifford(obj)))
19159 *
19160 * cpdef inline agc3(obj): # ««««««««
19161 * """
19162 * Convert CGA3 null vector to Euclidean 3D vector using Doran and Lasenby definition.
19163 */
19164
19165 static PyObject * __pyx_pw_8PyClical_87agc3(PyObject * __pyx_self, PyObject
*__pyx_v_obj); /*proto*/
19166 static CYTHON_INLINE PyObject * __pyx_f_8PyClical_agc3(PyObject * __pyx_v_obj,
CYTHON_UNUSED int __pyx_skip_dispatch) {
19167 PyObject * __pyx_r = NULL;
19168 __Pyx_RefNannyDeclarations
19169 PyObject * __pyx_t_1 = NULL;
19170 PyObject * __pyx_t_2 = NULL;
19171 int __pyx_lineno = 0;
19172 const char * __pyx_filename = NULL;
19173 int __pyx_clineno = 0;
19174 __Pyx_RefNannySetupContext("agc3", 0);
19175
19176 /* "PyClical.pyx":1902
19177 * 0
19178 * """
19179 * return clifford().wrap(glucat.agc3(toClifford(obj))) # ««««««««
19180 *
19181 * # Some abbreviations.
19182 */
19183 __Pyx_XDECREf(__pyx_r);
19184 __pyx_t_1 = __Pyx_PyObject_CallNoArg(((PyObject *) __pyx_ptype_8PyClical_clifford));
19185 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1902, __pyx_L1_error)
19186 __Pyx_GOTREF(__pyx_t_1);
19187 __pyx_t_2 = __pyx_f_8PyClical_8clifford_wrap(((struct __pyx_obj_8PyClical_clifford
*) __pyx_t_1), cga3::agc3(__pyx_f_8PyClical_toClifford(__pyx_v_obj))); if (unlikely(!__pyx_t_2))
__PYX_ERR(0, 1902, __pyx_L1_error)

```

```

19187 __Pyx_GOTREF(__pyx_t_2);
19188 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
19189 __pyx_r = __pyx_t_2;
19190 __pyx_t_2 = 0;
19191 goto __pyx_L0;
19192
19193 /* "PyClical.pyx":1893
19194 * return clifford().wrap(glucat.cga3std(toClifford(obj)))
19195 *
19196 * cpdef inline agc3(obj): # ««««««««
19197 * """
19198 * Convert CGA3 null vector to Euclidean 3D vector using Doran and Lasenby definition.
19199 */
19200
19201 /* function exit code */
19202 __pyx_L1_error++;
19203 __Pyx_XDECREF(__pyx_t_1);
19204 __Pyx_XDECREF(__pyx_t_2);
19205 __Pyx_AddTraceback("PyClical.agc3", __pyx_clineno, __pyx_lineno, __pyx_filename);
19206 __pyx_r = 0;
19207 __pyx_L0:;
19208 __Pyx_XGIVEREF(__pyx_r);
19209 __Pyx_RefNannyFinishContext();
19210 return __pyx_r;
19211 }
19212
19213 /* Python wrapper */
19214 static PyObject * __pyx_pw_8PyClical_87agc3(PyObject * __pyx_self, PyObject
19215 * __pyx_v_obj); /*proto*/
19216 static char __pyx_doc_8PyClical_86agc3[] = "\n Convert CGA3 null vector to
Euclidean 3D vector using Doran and Lasenby definition.\n\n >> x=clifford(\"2{1}+9{2}+{3}\");
print(agc3(cga3(x)))\n 2{1}+9{2}+{3}\n >> x=clifford(\"2{1}+9{2}+{3}\");
print(agc3(cga3(x))-x)\n 0\n ";
19217 static PyObject * __pyx_pw_8PyClical_87agc3(PyObject * __pyx_self, PyObject
19218 * __pyx_v_obj) {
19219 PyObject * __pyx_r = 0;
19220 __Pyx_RefNannyDeclarations
19221 __Pyx_RefNannySetupContext("agc3 (wrapper)", 0);
19222 __pyx_r = __pyx_pf_8PyClical_86agc3(__pyx_self, ((PyObject *) __pyx_v_obj));
19223
19224 /* function exit code */
19225 __Pyx_RefNannyFinishContext();
19226 return __pyx_r;
19227 }
19228 static PyObject * __pyx_pf_8PyClical_86agc3(CYTHON_UNUSED PyObject * __pyx_self,
PyObject * __pyx_v_obj) {
19229 PyObject * __pyx_r = NULL;
19230 __Pyx_RefNannyDeclarations
19231 PyObject * __pyx_t_1 = NULL;
19232 int __pyx_lineno = 0;
19233 const char * __pyx_filename = NULL;
19234 int __pyx_clineno = 0;
19235 __Pyx_RefNannySetupContext("agc3", 0);
19236 __Pyx_XDECREF(__pyx_r);
19237 __pyx_t_1 = __pyx_f_8PyClical_agc3(__pyx_v_obj, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1893, __pyx_L1_error)
19238 __Pyx_GOTREF(__pyx_t_1);
19239 __pyx_r = __pyx_t_1;
19240 __pyx_t_1 = 0;
19241 goto __pyx_L0;
19242
19243 /* function exit code */
19244 __pyx_L1_error++;
19245 __Pyx_XDECREF(__pyx_t_1);
19246 __Pyx_AddTraceback("PyClical.agc3", __pyx_clineno, __pyx_lineno, __pyx_filename);
19247 __pyx_r = NULL;
19248 __pyx_L0:;
19249 __Pyx_XGIVEREF(__pyx_r);
19250 __Pyx_RefNannyFinishContext();
19251 return __pyx_r;
19252 }
19253
19254 /* "PyClical.pyx":1936
19255 * """
19256 * def e(obj): # ««««««««
19257 * """
19258 * Abbreviation for clifford(index_set(obj)).
19259 */
19260
19261 /* Python wrapper */
19262 static PyObject * __pyx_pw_8PyClical_89e(PyObject * __pyx_self, PyObject * __pyx_v_obj);
19263 /*proto*/
19264 static char __pyx_doc_8PyClical_88e[] = "\n Abbreviation for
clifford(index_set(obj)).\n\n >> print(e(1))\n {1}\n >> print(e(-1))\n {-1}\n >>
print(e(0))\n 1\n ";

```

```

19264 static PyMethodDef __pyx_mdef_8PyClical_89e = {"e",
(PyCFunction)__pyx_pw_8PyClical_89e, METH_O, __pyx_doc_8PyClical_88e};
19265 static PyObject *__pyx_pw_8PyClical_89e(PyObject *__pyx_self, PyObject *__pyx_v_obj) {
19266 PyObject *__pyx_r = 0;
19267 __Pyx_RefNannyDeclarations
19268 __Pyx_RefNannySetupContext("e (wrapper)", 0);
19269 __pyx_r = __pyx_pf_8PyClical_88e(__pyx_self, ((PyObject *)__pyx_v_obj));
19270
19271 /* function exit code */
19272 __Pyx_RefNannyFinishContext();
19273 return __pyx_r;
19274 }
19275
19276 static PyObject *__pyx_pf_8PyClical_88e(CYTHON_UNUSED PyObject *__pyx_self, PyObject
*__pyx_v_obj) {
19277 PyObject *__pyx_r = NULL;
19278 __Pyx_RefNannyDeclarations
19279 PyObject *__pyx_t_1 = NULL;
19280 PyObject *__pyx_t_2 = NULL;
19281 int __pyx_lineno = 0;
19282 const char *__pyx_filename = NULL;
19283 int __pyx_clineno = 0;
19284 __Pyx_RefNannySetupContext("e", 0);
19285
19286 /* "PyClical.pyx":1947
19287 * 1
19288 * """
19289 * return clifford(index_set(obj)) # ««««««««
19290 *
19291 * def istpq(p, q):
19292 */
19293 __Pyx_XDECREF(__pyx_r);
19294 __pyx_t_1 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_index_set),
__pyx_v_obj); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1947, __pyx_L1_error)
19295 __Pyx_GOTREF(__pyx_t_1);
19296 __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_clifford),
__pyx_t_1); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1947, __pyx_L1_error)
19297 __Pyx_GOTREF(__pyx_t_2);
19298 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
19299 __pyx_r = __pyx_t_2;
19300 __pyx_t_2 = 0;
19301 goto __pyx_L0;
19302
19303 /* "PyClical.pyx":1936
19304 * """
19305 *
19306 * def e(obj): # ««««««««
19307 * """
19308 * Abbreviation for clifford(index_set(obj)).
19309 */
19310
19311 /* function exit code */
19312 __pyx_L1_error;;
19313 __Pyx_XDECREF(__pyx_t_1);
19314 __Pyx_XDECREF(__pyx_t_2);
19315 __Pyx_AddTraceback("PyClical.e", __pyx_clineno, __pyx_lineno, __pyx_filename);
19316 __pyx_r = NULL;
19317 __pyx_L0;
19318 __Pyx_XGIVEREF(__pyx_r);
19319 __Pyx_RefNannyFinishContext();
19320 return __pyx_r;
19321 }
19322
19323 /* "PyClical.pyx":1949
19324 * return clifford(index_set(obj))
19325 *
19326 * def istpq(p, q): # ««««««««
19327 * """
19328 * Abbreviation for index_set({-q,...p}).
19329 */
19330
19331 /* Python wrapper */
19332 static PyObject *__pyx_pw_8PyClical_91istpq(PyObject *__pyx_self, PyObject
*__pyx_args, PyObject *__pyx_kwds); /*proto*/
19333 static char __pyx_doc_8PyClical_90istpq[] = "\n Abbreviation for
index_set({-q,...p}).\n\n >> print(istpq(2,3))\n {-3,-2,-1,1,2}\n ";
19334 static PyMethodDef __pyx_mdef_8PyClical_91istpq = {"istpq",
(PyCFunction)(void*)__pyx_pw_8PyClical_91istpq, METH_VARARGS|METH_KEYWORDS,
__pyx_doc_8PyClical_90istpq};
19335 static PyObject *__pyx_pw_8PyClical_91istpq(PyObject *__pyx_self, PyObject
*__pyx_args, PyObject *__pyx_kwds) {
19336 PyObject *__pyx_v_p = 0;
19337 PyObject *__pyx_v_q = 0;
19338 int __pyx_lineno = 0;
19339 const char *__pyx_filename = NULL;
19340 int __pyx_clineno = 0;
19341 PyObject *__pyx_r = 0;

```

```

19342 __Pyx_RefNannyDeclarations
19343 __Pyx_RefNannySetupContext("istpq (wrapper)", 0);
19344 {
19345 static PyObject * __pyx_pyargnames[] = {&__pyx_n_s_p, &__pyx_n_s_q, 0};
19346 PyObject* values[2] = {0, 0};
19347 if (unlikely(__pyx_kwds)) {
19348 Py_ssize_t kw_args;
19349 const Py_ssize_t pos_args = PyTuple_GET_SIZE(__pyx_args);
19350 switch (pos_args) {
19351 case 2: values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
19352 CYTHON_FALLTHROUGH;
19353 case 1: values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
19354 CYTHON_FALLTHROUGH;
19355 case 0: break;
19356 default: goto __pyx_L5_argtuple_error;
19357 }
19358 kw_args = PyDict_Size(__pyx_kwds);
19359 switch (pos_args) {
19360 case 0:
19361 if (likely((values[0] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_p)) !=
19362 0)) kw_args--;
19363 else goto __pyx_L5_argtuple_error;
19364 CYTHON_FALLTHROUGH;
19365 case 1:
19366 if (likely((values[1] = __Pyx_PyDict_GetItemStr(__pyx_kwds, __pyx_n_s_q)) !=
19367 0)) kw_args--;
19368 else {
19369 __Pyx_RaiseArgtupleInvalid("istpq", 1, 2, 2, 1); __PYX_ERR(0, 1949,
19370 __pyx_L3_error)
19371 }
19372 if (unlikely(kw_args > 0)) {
19373 if (unlikely(__Pyx_ParseOptionalKeywords(__pyx_kwds, __pyx_pyargnames, 0,
19374 values, pos_args, "istpq") < 0)) __PYX_ERR(0, 1949, __pyx_L3_error)
19375 } else if (PyTuple_GET_SIZE(__pyx_args) != 2) {
19376 goto __pyx_L5_argtuple_error;
19377 } else {
19378 values[0] = PyTuple_GET_ITEM(__pyx_args, 0);
19379 values[1] = PyTuple_GET_ITEM(__pyx_args, 1);
19380 }
19381 __pyx_v_p = values[0];
19382 __pyx_v_q = values[1];
19383 }
19384 goto __pyx_L4_argument_unpacking_done;
19385 __pyx_L5_argtuple_error:;
19386 __Pyx_RaiseArgtupleInvalid("istpq", 1, 2, 2, PyTuple_GET_SIZE(__pyx_args));
19387 __PYX_ERR(0, 1949, __pyx_L3_error)
19388 __pyx_L3_error:;
19389 __Pyx_AddTraceback("PyClical.istpq", __pyx_clineno, __pyx_lineno, __pyx_filename);
19390 __Pyx_RefNannyFinishContext();
19391 return NULL;
19392 __pyx_L4_argument_unpacking_done:;
19393 __pyx_r = __pyx_pf_8PyClical_90istpq(__pyx_self, __pyx_v_p, __pyx_v_q);
19394
19395 /* function exit code */
19396 __Pyx_RefNannyFinishContext();
19397 return __pyx_r;
19398 }
19399 }
19400 static PyObject * __pyx_pf_8PyClical_90istpq(CYTHON_UNUSED PyObject * __pyx_self,
19401 PyObject * __pyx_v_p, PyObject * __pyx_v_q) {
19402 PyObject * __pyx_r = NULL;
19403 __Pyx_RefNannyDeclarations
19404 PyObject * __pyx_t_1 = NULL;
19405 PyObject * __pyx_t_2 = NULL;
19406 PyObject * __pyx_t_3 = NULL;
19407 int __pyx_lineno = 0;
19408 const char * __pyx_filename = NULL;
19409 int __pyx_clineno = 0;
19410 __Pyx_RefNannySetupContext("istpq", 0);
19411
19412 /* "PyClical.pyx":1956
19413 * {-3,-2,-1,1,2}
19414 * """
19415 * return index_set(set(range(-q,p+1))) # ««««««««
19416 * ninf3 = e(4) + e(-1) # Null infinity point in 3D Conformal Geometric Algebra [DL].
19417 */
19418 __Pyx_XDECREF(__pyx_r);
19419 __pyx_t_1 = PyNumber_Negative(__pyx_v_q); if (unlikely(!__pyx_t_1)) __PYX_ERR(0,
19420 1956, __pyx_L1_error)
19421 __Pyx_GOTREF(__pyx_t_1);
19422 __pyx_t_2 = __Pyx_PyInt_AddObjC(__pyx_v_p, __pyx_t_1, 1, 0, 0); if
19423 (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1956, __pyx_L1_error)
19424 __Pyx_GOTREF(__pyx_t_2);
19425 __pyx_t_3 = PyTuple_New(2); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1956,

```

```

__pyx_L1_error)
19421 __Pyx_GOTREF(__pyx_t_3);
19422 __Pyx_GIVEREF(__pyx_t_1);
19423 PyTuple_SET_ITEM(__pyx_t_3, 0, __pyx_t_1);
19424 __Pyx_GIVEREF(__pyx_t_2);
19425 PyTuple_SET_ITEM(__pyx_t_3, 1, __pyx_t_2);
19426 __pyx_t_1 = 0;
19427 __pyx_t_2 = 0;
19428 __pyx_t_2 = __Pyx_PyObject_Call(__pyx_builtin_range, __pyx_t_3, NULL); if
(unlikely(!__pyx_t_2)) __PYX_ERR(0, 1956, __pyx_L1_error)
19429 __Pyx_GOTREF(__pyx_t_2);
19430 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
19431 __pyx_t_3 = PySet_New(__pyx_t_2); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 1956,
__pyx_L1_error)
19432 __Pyx_GOTREF(__pyx_t_3);
19433 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
19434 __pyx_t_2 = __Pyx_PyObject_CallOneArg(((PyObject *)__pyx_ptype_8PyClical_index_set),
__pyx_t_3); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1956, __pyx_L1_error)
19435 __Pyx_GOTREF(__pyx_t_2);
19436 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
19437 __pyx_r = __pyx_t_2;
19438 __pyx_t_2 = 0;
19439 goto __pyx_L0;
19440
19441 /* "PyClical.pyx":1949
19442 * return clifford(index_set(obj))
19443 *
19444 * def istpq(p, q): # ««««««««
19445 * """
19446 * Abbreviation for index_set({-q,...p}).
19447 */
19448
19449 /* function exit code */
19450 __pyx_L1_error;;
19451 __Pyx_XDECREF(__pyx_t_1);
19452 __Pyx_XDECREF(__pyx_t_2);
19453 __Pyx_XDECREF(__pyx_t_3);
19454 __Pyx_AddTraceback("PyClical.istpq", __pyx_clineno, __pyx_lineno, __pyx_filename);
19455 __pyx_r = NULL;
19456 __pyx_L0;
19457 __Pyx_XGIVEREF(__pyx_r);
19458 __Pyx_RefNannyFinishContext();
19459 return __pyx_r;
19460 }
19461
19462 /* "PyClical.pyx":1962
19463 *
19464 * # Doctest interface.
19465 * def _test(): # ««««««««
19466 * import PyClical, doctest
19467 * return doctest.testmod(PyClical)
19468 */
19469
19470 /* Python wrapper */
19471 static PyObject * __pyx_pw_8PyClical_93_test(PyObject * __pyx_self, CYTHON_UNUSED
PyObject *unused); /*proto*/
19472 static PyMethodDef __pyx_mdef_8PyClical_93_test = {"_test",
(PyCFunction) __pyx_pw_8PyClical_93_test, METH_NOARGS, 0};
19473 static PyObject * __pyx_pw_8PyClical_93_test(PyObject * __pyx_self, CYTHON_UNUSED
PyObject *unused) {
19474 PyObject * __pyx_r = 0;
19475 __Pyx_RefNannyDeclarations
19476 __Pyx_RefNannySetupContext("_test (wrapper)", 0);
19477 __pyx_r = __pyx_pf_8PyClical_92_test(__pyx_self);
19478
19479 /* function exit code */
19480 __Pyx_RefNannyFinishContext();
19481 return __pyx_r;
19482 }
19483
19484 static PyObject * __pyx_pf_8PyClical_92_test(CYTHON_UNUSED PyObject * __pyx_self) {
19485 PyObject * __pyx_v_PyClical = NULL;
19486 PyObject * __pyx_v_doctest = NULL;
19487 PyObject * __pyx_r = NULL;
19488 __Pyx_RefNannyDeclarations
19489 PyObject * __pyx_t_1 = NULL;
19490 PyObject * __pyx_t_2 = NULL;
19491 PyObject * __pyx_t_3 = NULL;
19492 int __pyx_lineno = 0;
19493 const char * __pyx_filename = NULL;
19494 int __pyx_clineno = 0;
19495 __Pyx_RefNannySetupContext("_test", 0);
19496
19497 /* "PyClical.pyx":1963
19498 * # Doctest interface.
19499 * def _test():
19500 * import PyClical, doctest # ««««««««

```

```

19501 * return doctest.testmod(PyClical)
19502 *
19503 */
19504 __pyx_t_1 = __Pyx_Import(__pyx_n_s_PyClical, 0, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1963, __pyx_L1_error)
19505 __Pyx_GOTREF(__pyx_t_1);
19506 __pyx_v_PyClical = __pyx_t_1;
19507 __pyx_t_1 = 0;
19508 __pyx_t_1 = __Pyx_Import(__pyx_n_s_doctest, 0, 0); if (unlikely(!__pyx_t_1))
__PYX_ERR(0, 1963, __pyx_L1_error)
19509 __Pyx_GOTREF(__pyx_t_1);
19510 __pyx_v_doctest = __pyx_t_1;
19511 __pyx_t_1 = 0;
19512
19513 /* "PyClical.pyx":1964
19514 * def _test():
19515 * import PyClical, doctest
19516 * return doctest.testmod(PyClical) # ««««««««
19517 *
19518 * if __name__ == "__main__":
19519 */
19520 __Pyx_XDECREF(__pyx_r);
19521 __pyx_t_2 = __Pyx_PyObject_GetAttrStr(__pyx_v_doctest, __pyx_n_s_testmod); if
(unlikely(!__pyx_t_2)) __PYX_ERR(0, 1964, __pyx_L1_error)
19522 __Pyx_GOTREF(__pyx_t_2);
19523 __pyx_t_3 = NULL;
19524 if (CYTHON_UNPACK_METHODS && likely(PyMethod_Check(__pyx_t_2))) {
19525 __pyx_t_3 = PyMethod_GET_SELF(__pyx_t_2);
19526 if (likely(__pyx_t_3)) {
19527 PyObject* function = PyMethod_GET_FUNCTION(__pyx_t_2);
19528 __Pyx_INCREF(__pyx_t_3);
19529 __Pyx_INCREF(function);
19530 __Pyx_DECREF_SET(__pyx_t_2, function);
19531 }
19532 }
19533 __pyx_t_1 = (__pyx_t_3) ? __Pyx_PyObject_Call2Args(__pyx_t_2, __pyx_t_3,
__pyx_v_PyClical) : __Pyx_PyObject_CallOneArg(__pyx_t_2, __pyx_v_PyClical);
19534 __Pyx_XDECREF(__pyx_t_3); __pyx_t_3 = 0;
19535 if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1964, __pyx_L1_error)
19536 __Pyx_GOTREF(__pyx_t_1);
19537 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
19538 __pyx_r = __pyx_t_1;
19539 __pyx_t_1 = 0;
19540 goto __pyx_L0;
19541
19542 /* "PyClical.pyx":1962
19543 *
19544 * # Doctest interface.
19545 * def _test(): # ««««««««
19546 * import PyClical, doctest
19547 * return doctest.testmod(PyClical)
19548 */
19549
19550 /* function exit code */
19551 __pyx_L1_error;
19552 __Pyx_XDECREF(__pyx_t_1);
19553 __Pyx_XDECREF(__pyx_t_2);
19554 __Pyx_XDECREF(__pyx_t_3);
19555 __Pyx_AddTraceback("PyClical._test", __pyx_clineno, __pyx_lineno, __pyx_filename);
19556 __pyx_r = NULL;
19557 __pyx_L0;
19558 __Pyx_XDECREF(__pyx_v_PyClical);
19559 __Pyx_XDECREF(__pyx_v_doctest);
19560 __Pyx_XGIVEREF(__pyx_r);
19561 __Pyx_RefNannyFinishContext();
19562 return __pyx_r;
19563 }
19564
19565 /* "string.to_py":31
19566 *
19567 * @cname("__pyx_convert_PyObject_string_to_py_std_in_string")
19568 * cdef inline object __pyx_convert_PyObject_string_to_py_std_in_string(const string& s):
19569 * return __Pyx_PyObject_FromStringAndSize(s.data(), s.size())
19570 * cdef extern from *:
19571 */
19572
19573 static CYTHON_INLINE PyObject
*__pyx_convert_PyObject_string_to_py_std_in_string(std::string const &__pyx_v_s) {
19574 PyObject *__pyx_r = NULL;
19575 __Pyx_RefNannyDeclarations
19576 PyObject *__pyx_t_1 = NULL;
19577 int __pyx_lineno = 0;
19578 const char *__pyx_filename = NULL;
19579 int __pyx_clineno = 0;
19580 __Pyx_RefNannySetupContext("__pyx_convert_PyObject_string_to_py_std_in_string", 0);
19581

```

```

19582 /* "string.to_py":32
19583 * @cname("__pyx_convert_PyObject_string_to_py_std__in_string")
19584 * cdef inline object __pyx_convert_PyObject_string_to_py_std__in_string(const string& s):
19585 * return __Pyx_PyObject_FromStringAndSize(s.data(), s.size()) # ««««««««
19586 * cdef extern from *:
19587 * cdef object __Pyx_PyUnicode_FromStringAndSize(const char*, size_t)
19588 */
19589 __Pyx_XDECREF(__pyx_r);
19590 __pyx_t_1 = __Pyx_PyObject_FromStringAndSize(__pyx_v_s.data(), __pyx_v_s.size()); if
(unlikely(!__pyx_t_1)) __PYX_ERR(1, 32, __pyx_L1_error)
19591 __Pyx_GOTREF(__pyx_t_1);
19592 __pyx_r = __pyx_t_1;
19593 __pyx_t_1 = 0;
19594 goto __pyx_L0;
19595
19596 /* "string.to_py":31
19597 *
19598 * @cname("__pyx_convert_PyObject_string_to_py_std__in_string")
19599 * cdef inline object __pyx_convert_PyObject_string_to_py_std__in_string(const string& s):
19600 * return __Pyx_PyObject_FromStringAndSize(s.data(), s.size())
19601 * cdef extern from *:
19602 */
19603
19604 /* function exit code */
19605 __pyx_L1_error;;
19606 __Pyx_XDECREF(__pyx_t_1);
19607
19608 __Pyx_AddTraceback("string.to_py.__pyx_convert_PyObject_string_to_py_std__in_string", __pyx_clineno,
__pyx_lineno, __pyx_filename);
19609 __pyx_r = 0;
19610 __pyx_L0;;
19611 __Pyx_XGIVEREF(__pyx_r);
19612 __Pyx_RefNannyFinishContext();
19613 return __pyx_r;
19614 }
19615
19616 /* "string.to_py":37
19617 *
19618 * @cname("__pyx_convert_PyUnicode_string_to_py_std__in_string")
19619 * cdef inline object __pyx_convert_PyUnicode_string_to_py_std__in_string(const string& s):
19620 * return __Pyx_PyUnicode_FromStringAndSize(s.data(), s.size())
19621 * cdef extern from *:
19622 */
19623
19624 static CYTHON_INLINE PyObject
__pyx_convert_PyUnicode_string_to_py_std__in_string(std::string const &__pyx_v_s) {
19625 PyObject *__pyx_r = NULL;
19626 __Pyx_RefNannyDeclarations
19627 PyObject *__pyx_t_1 = NULL;
19628 int __pyx_lineno = 0;
19629 const char *__pyx_filename = NULL;
19630 int __pyx_clineno = 0;
19631 __Pyx_RefNannySetupContext("__pyx_convert_PyUnicode_string_to_py_std__in_string",
0);
19632
19633 /* "string.to_py":38
19634 * @cname("__pyx_convert_PyUnicode_string_to_py_std__in_string")
19635 * cdef inline object __pyx_convert_PyUnicode_string_to_py_std__in_string(const string& s):
19636 * return __Pyx_PyUnicode_FromStringAndSize(s.data(), s.size()) # ««««««««
19637 * cdef extern from *:
19638 * cdef object __Pyx_PyStr_FromStringAndSize(const char*, size_t)
19639 */
19640 __Pyx_XDECREF(__pyx_r);
19641 __pyx_t_1 = __Pyx_PyUnicode_FromStringAndSize(__pyx_v_s.data(), __pyx_v_s.size()); if
(unlikely(!__pyx_t_1)) __PYX_ERR(1, 38, __pyx_L1_error)
19642 __Pyx_GOTREF(__pyx_t_1);
19643 __pyx_r = __pyx_t_1;
19644 __pyx_t_1 = 0;
19645 goto __pyx_L0;
19646
19647 /* "string.to_py":37
19648 *
19649 * @cname("__pyx_convert_PyUnicode_string_to_py_std__in_string")
19650 * cdef inline object __pyx_convert_PyUnicode_string_to_py_std__in_string(const string& s):
19651 * return __Pyx_PyUnicode_FromStringAndSize(s.data(), s.size())
19652 * cdef extern from *:
19653 */
19654
19655 /* function exit code */
19656 __pyx_L1_error;;
19657 __Pyx_XDECREF(__pyx_t_1);
19658
19659 __Pyx_AddTraceback("string.to_py.__pyx_convert_PyUnicode_string_to_py_std__in_string", __pyx_clineno,
__pyx_lineno, __pyx_filename);

```



```

19658 __pyx_r = 0;
19659 __pyx_L0;
19660 __Pyx_XGIVEREF(__pyx_r);
19661 __Pyx_RefNannyFinishContext();
19662 return __pyx_r;
19663 }
19664
19665 /* "string.to_py":43
19666 *
19667 * @cname("__pyx_convert_PyStr_string_to_py_std__in_string")
19668 * cdef inline object __pyx_convert_PyStr_string_to_py_std__in_string(const string& s): #
19669 * ««««««
19670 * return __Pyx_PyStr_FromStringAndSize(s.data(), s.size())
19671 * cdef extern from *:
19672 */
19673
19674 static CYTHON_INLINE PyObject
19675 *__pyx_convert_PyStr_string_to_py_std__in_string(std::string const &__pyx_v_s) {
19676 PyObject *__pyx_r = NULL;
19677 __Pyx_RefNannyDeclarations
19678 PyObject *__pyx_t_1 = NULL;
19679 int __pyx_lineno = 0;
19680 const char *__pyx_filename = NULL;
19681 int __pyx_clineno = 0;
19682 __Pyx_RefNannySetupContext("__pyx_convert_PyStr_string_to_py_std__in_string", 0);
19683
19684 /* "string.to_py":44
19685 * @cname("__pyx_convert_PyStr_string_to_py_std__in_string")
19686 * cdef inline object __pyx_convert_PyStr_string_to_py_std__in_string(const string& s): # ««««««
19687 * return __Pyx_PyStr_FromStringAndSize(s.data(), s.size())
19688 * cdef extern from *:
19689 * cdef object __Pyx_PyBytes_FromStringAndSize(const char*, size_t)
19690 */
19691 __Pyx_XDECREF(__pyx_r);
19692 __pyx_t_1 = __Pyx_PyStr_FromStringAndSize(__pyx_v_s.data(), __pyx_v_s.size()); if
19693 (unlikely(!__pyx_t_1)) __PYX_ERR(1, 44, __pyx_L1_error)
19694 __Pyx_GOTREF(__pyx_t_1);
19695 __pyx_r = __pyx_t_1;
19696 __pyx_t_1 = 0;
19697 goto __pyx_L0;
19698
19699 /* "string.to_py":43
19700 *
19701 * @cname("__pyx_convert_PyStr_string_to_py_std__in_string")
19702 * cdef inline object __pyx_convert_PyStr_string_to_py_std__in_string(const string& s): #
19703 * ««««««
19704 * return __Pyx_PyStr_FromStringAndSize(s.data(), s.size())
19705 * cdef extern from *:
19706 */
19707
19708 /* function exit code */
19709 __pyx_L1_error;
19710 __Pyx_XDECREF(__pyx_t_1);
19711 __Pyx_AddTraceback("string.to_py.__pyx_convert_PyStr_string_to_py_std__in_string",
19712 __pyx_clineno, __pyx_lineno, __pyx_filename);
19713 __pyx_r = 0;
19714 __pyx_L0;
19715 __Pyx_XGIVEREF(__pyx_r);
19716 __Pyx_RefNannyFinishContext();
19717 return __pyx_r;
19718 }
19719
19720 /* "string.to_py":49
19721 *
19722 * @cname("__pyx_convert_PyBytes_string_to_py_std__in_string")
19723 * cdef inline object __pyx_convert_PyBytes_string_to_py_std__in_string(const string& s): # ««««««
19724 * ««««««
19725 * return __Pyx_PyBytes_FromStringAndSize(s.data(), s.size())
19726 * cdef extern from *:
19727 */
19728
19729 static CYTHON_INLINE PyObject
19730 *__pyx_convert_PyBytes_string_to_py_std__in_string(std::string const &__pyx_v_s) {
19731 PyObject *__pyx_r = NULL;
19732 __Pyx_RefNannyDeclarations
19733 PyObject *__pyx_t_1 = NULL;
19734 int __pyx_lineno = 0;
19735 const char *__pyx_filename = NULL;
19736 int __pyx_clineno = 0;
19737 __Pyx_RefNannySetupContext("__pyx_convert_PyBytes_string_to_py_std__in_string", 0);
19738
19739 /* "string.to_py":50
19740 * @cname("__pyx_convert_PyBytes_string_to_py_std__in_string")
19741 * cdef inline object __pyx_convert_PyBytes_string_to_py_std__in_string(const string& s): # ««««««
19742 * return __Pyx_PyBytes_FromStringAndSize(s.data(), s.size())
19743 * cdef extern from *:
19744 * cdef object __Pyx_PyByteArray_FromStringAndSize(const char*, size_t)

```

```

19738 */
19739 __Pyx_XDECREF(__pyx_r);
19740 __pyx_t_1 = __Pyx_PyBytes_FromStringAndSize(__pyx_v_s.data(), __pyx_v_s.size()); if
(unlikely(!__pyx_t_1)) __PYX_ERR(1, 50, __pyx_L1_error)
19741 __Pyx_GOTREF(__pyx_t_1);
19742 __pyx_r = __pyx_t_1;
19743 __pyx_t_1 = 0;
19744 goto __pyx_L0;
19745
19746 /* "string.to_py":49
19747
19748 * @cname("__pyx_convert_PyBytes_string_to_py_std__in_string")
19749 * cdef inline object __pyx_convert_PyBytes_string_to_py_std__in_string(const string& s):
19750 * return __Pyx_PyBytes_FromStringAndSize(s.data(), s.size())
19751 * cdef extern from *:
19752 */
19753
19754 /* function exit code */
19755 __pyx_L1_error;
19756 __Pyx_XDECREF(__pyx_t_1);
19757 __Pyx_AddTraceback("string.to_py.__pyx_convert_PyBytes_string_to_py_std__in_string",
__pyx_clineno, __pyx_lineno, __pyx_filename);
19758 __pyx_r = 0;
19759 __pyx_L0;
19760 __Pyx_XGIVEREF(__pyx_r);
19761 __Pyx_RefNannyFinishContext();
19762 return __pyx_r;
19763 }
19764
19765 /* "string.to_py":55
19766
19767 * @cname("__pyx_convert_PyByteArray_string_to_py_std__in_string")
19768 * cdef inline object __pyx_convert_PyByteArray_string_to_py_std__in_string(const string& s):
19769 * return __Pyx_PyByteArray_FromStringAndSize(s.data(), s.size())
19770 *
19771 */
19772
19773 static CYTHON_INLINE PyObject
*__pyx_convert_PyByteArray_string_to_py_std__in_string(std::string const &__pyx_v_s) {
19774 PyObject *__pyx_r = NULL;
19775 __Pyx_RefNannyDeclarations
19776 PyObject *__pyx_t_1 = NULL;
19777 int __pyx_lineno = 0;
19778 const char *__pyx_filename = NULL;
19779 int __pyx_clineno = 0;
19780 __Pyx_RefNannySetupContext("__pyx_convert_PyByteArray_string_to_py_std__in_string",
0);
19781
19782 /* "string.to_py":56
19783 * @cname("__pyx_convert_PyByteArray_string_to_py_std__in_string")
19784 * cdef inline object __pyx_convert_PyByteArray_string_to_py_std__in_string(const string& s):
19785 * return __Pyx_PyByteArray_FromStringAndSize(s.data(), s.size()) # ««««««««
19786 *
19787 */
19788 __Pyx_XDECREF(__pyx_r);
19789 __pyx_t_1 = __Pyx_PyByteArray_FromStringAndSize(__pyx_v_s.data(), __pyx_v_s.size());
19790 if (unlikely(!__pyx_t_1)) __PYX_ERR(1, 56, __pyx_L1_error)
19791 __Pyx_GOTREF(__pyx_t_1);
19792 __pyx_r = __pyx_t_1;
19793 __pyx_t_1 = 0;
19794 goto __pyx_L0;
19795
19796 /* "string.to_py":55
19797
19798 * @cname("__pyx_convert_PyByteArray_string_to_py_std__in_string")
19799 * cdef inline object __pyx_convert_PyByteArray_string_to_py_std__in_string(const string& s):
19800 * return __Pyx_PyByteArray_FromStringAndSize(s.data(), s.size())
19801 *
19802 */
19803
19804 /* function exit code */
19805 __pyx_L1_error;
19806 __Pyx_XDECREF(__pyx_t_1);
19807 __Pyx_AddTraceback("string.to_py.__pyx_convert_PyByteArray_string_to_py_std__in_string",
__pyx_clineno, __pyx_lineno, __pyx_filename);
19808 __pyx_r = 0;
19809 __pyx_L0;
19810 __Pyx_XGIVEREF(__pyx_r);
19811 __Pyx_RefNannyFinishContext();
19812 return __pyx_r;
19813 }
19814 static struct __pyx_vtabstruct_8PyClical_index_set __pyx_vtable_8PyClical_index_set;

```

```

19815 static PyObject *__pyx_tp_new_8PyClical_index_set(PyTypeObject *t, PyObject *a,
19816 PyObject *k) {
19817 struct __pyx_obj_8PyClical_index_set *p;
19818 PyObject *o;
19819 if (likely((t->tp_flags & Py_TPFLAGS_IS_ABSTRACT) == 0)) {
19820 o = (*t->tp_alloc)(t, 0);
19821 } else {
19822 o = (PyObject *) PyBaseObject_Type.tp_new(t, __pyx_empty_tuple, 0);
19823 }
19824 if (unlikely(!o)) return 0;
19825 p = ((struct __pyx_obj_8PyClical_index_set *)o);
19826 p->__pyx_vtab = __pyx_vtabptr_8PyClical_index_set;
19827 if (unlikely(__pyx_pw_8PyClical_9index_set_3__cinit__(o, a, k) < 0)) goto bad;
19828 return o;
19829 bad:
19830 Py_DECREF(o); o = 0;
19831 return NULL;
19832 }
19833
19834 static void __pyx_tp_dealloc_8PyClical_index_set(PyObject *o) {
19835 #if CYTHON_USE_TP_FINALIZE
19836 if (unlikely(PyType_HasFeature(Py_TYPE(o), Py_TPFLAGS_HAVE_FINALIZE) &&
19837 (!PyType_IS_GC(Py_TYPE(o)) || !PyGC_FINALIZED(o))) {
19838 if (PyObject_CallFinalizerFromDealloc(o)) return;
19839 }
19840 #endif
19841 {
19842 PyObject *etype, *eval, *etb;
19843 PyErr_Fetch(&etype, &eval, &etb);
19844 __Pyx_SET_REFCNT(o, Py_REFCNT(o) + 1);
19845 __pyx_pw_8PyClical_9index_set_5__dealloc__(o);
19846 __Pyx_SET_REFCNT(o, Py_REFCNT(o) - 1);
19847 PyErr_Restore(etype, eval, etb);
19848 }
19849 (*Py_TYPE(o)->tp_free)(o);
19850 }
19851 static PyObject *__pyx_sq_item_8PyClical_index_set(PyObject *o, Py_ssize_t i) {
19852 PyObject *r;
19853 PyObject *x = PyInt_FromSsize_t(i); if(!x) return 0;
19854 r = Py_TYPE(o)->tp_as_mapping->mp_subscript(o, x);
19855 Py_DECREF(x);
19856 return r;
19857 }
19858 static int __pyx_mp_ass_subscript_8PyClical_index_set(PyObject *o, PyObject *i,
19859 PyObject *v) {
19860 if (v) {
19861 return __pyx_pw_8PyClical_9index_set_9__setitem__(o, i, v);
19862 }
19863 else {
19864 PyErr_Format(PyExc_NotImplementedError,
19865 "Subscript deletion not supported by %.200s", Py_TYPE(o)->tp_name);
19866 return -1;
19867 }
19868 }
19869 static PyMethodDef __pyx_methods_8PyClical_index_set[] = {
19870 {"copy", (PyCFunction) __pyx_pw_8PyClical_9index_set_1copy, METH_NOARGS,
19871 __pyx_doc_8PyClical_9index_set_copy},
19872 {"count", (PyCFunction) __pyx_pw_8PyClical_9index_set_32count, METH_NOARGS,
19873 __pyx_doc_8PyClical_9index_set_31count},
19874 {"count_neg", (PyCFunction) __pyx_pw_8PyClical_9index_set_34count_neg, METH_NOARGS,
19875 __pyx_doc_8PyClical_9index_set_33count_neg},
19876 {"count_pos", (PyCFunction) __pyx_pw_8PyClical_9index_set_36count_pos, METH_NOARGS,
19877 __pyx_doc_8PyClical_9index_set_35count_pos},
19878 {"min", (PyCFunction) __pyx_pw_8PyClical_9index_set_38min, METH_NOARGS,
19879 __pyx_doc_8PyClical_9index_set_37min},
19880 {"max", (PyCFunction) __pyx_pw_8PyClical_9index_set_40max, METH_NOARGS,
19881 __pyx_doc_8PyClical_9index_set_39max},
19882 {"hash_fn", (PyCFunction) __pyx_pw_8PyClical_9index_set_42hash_fn, METH_NOARGS,
19883 __pyx_doc_8PyClical_9index_set_41hash_fn},
19884 {"sign_of_mult", (PyCFunction) __pyx_pw_8PyClical_9index_set_44sign_of_mult, METH_O,
19885 __pyx_doc_8PyClical_9index_set_43sign_of_mult},
19886 {"sign_of_square", (PyCFunction) __pyx_pw_8PyClical_9index_set_46sign_of_square,
19887 METH_NOARGS, __pyx_doc_8PyClical_9index_set_45sign_of_square},
19888 {"__reduce_cython__",
19889 (PyCFunction) __pyx_pw_8PyClical_9index_set_52__reduce_cython__, METH_NOARGS, 0},
19890 {"__setstate_cython__",
19891 (PyCFunction) __pyx_pw_8PyClical_9index_set_54__setstate_cython__, METH_O, 0},
19892 {0, 0, 0, 0}
19893 };
19894
19895 static PyNumberMethods __pyx_tp_as_number_index_set = {
19896 0, /*nb_add*/
19897 0, /*nb_subtract*/
19898 0, /*nb_multiply*/
19899 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX <

```

```

0x03050000)
19888 0, /*nb_divide*/
19889 #endif
19890 0, /*nb_remainder*/
19891 0, /*nb_divmod*/
19892 0, /*nb_power*/
19893 0, /*nb_negative*/
19894 0, /*nb_positive*/
19895 0, /*nb_absolute*/
19896 0, /*nb_nonzero*/
19897 __pyx_pw_8PyClical_9index_set_18__invert__, /*nb_invert*/
19898 0, /*nb_lshift*/
19899 0, /*nb_rshift*/
19900 __pyx_pw_8PyClical_9index_set_24__and__, /*nb_and*/
19901 __pyx_pw_8PyClical_9index_set_20__xor__, /*nb_xor*/
19902 __pyx_pw_8PyClical_9index_set_28__or__, /*nb_or*/
19903 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX <
0x03050000)
19904 0, /*nb_coerce*/
19905 #endif
19906 0, /*nb_int*/
19907 #if PY_MAJOR_VERSION < 3
19908 0, /*nb_long*/
19909 #else
19910 0, /*reserved*/
19911 #endif
19912 0, /*nb_float*/
19913 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX <
0x03050000)
19914 0, /*nb_oct*/
19915 #endif
19916 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX <
0x03050000)
19917 0, /*nb_hex*/
19918 #endif
19919 0, /*nb_inplace_add*/
19920 0, /*nb_inplace_subtract*/
19921 0, /*nb_inplace_multiply*/
19922 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX <
0x03050000)
19923 0, /*nb_inplace_divide*/
19924 #endif
19925 0, /*nb_inplace_remainder*/
19926 0, /*nb_inplace_power*/
19927 0, /*nb_inplace_lshift*/
19928 0, /*nb_inplace_rshift*/
19929 __pyx_pw_8PyClical_9index_set_26__iand__, /*nb_inplace_and*/
19930 __pyx_pw_8PyClical_9index_set_22__ixor__, /*nb_inplace_xor*/
19931 __pyx_pw_8PyClical_9index_set_30__ior__, /*nb_inplace_or*/
19932 0, /*nb_floor_divide*/
19933 0, /*nb_true_divide*/
19934 0, /*nb_inplace_floor_divide*/
19935 0, /*nb_inplace_true_divide*/
19936 0, /*nb_index*/
19937 #if PY_VERSION_HEX >= 0x03050000
19938 0, /*nb_matrix_multiply*/
19939 #endif
19940 #if PY_VERSION_HEX >= 0x03050000
19941 0, /*nb_inplace_matrix_multiply*/
19942 #endif
19943 };
19944
19945 static PySequenceMethods __pyx_tp_as_sequence_index_set = {
19946 0, /*sq_length*/
19947 0, /*sq_concat*/
19948 0, /*sq_repeat*/
19949 __pyx_sq_item_8PyClical_index_set, /*sq_item*/
19950 0, /*sq_slice*/
19951 0, /*sq_ass_item*/
19952 0, /*sq_ass_slice*/
19953 __pyx_pw_8PyClical_9index_set_13__contains__, /*sq_contains*/
19954 0, /*sq_inplace_concat*/
19955 0, /*sq_inplace_repeat*/
19956 };
19957
19958 static PyMappingMethods __pyx_tp_as_mapping_index_set = {
19959 0, /*mp_length*/
19960 __pyx_pw_8PyClical_9index_set_11__getitem__, /*mp_subscript*/
19961 __pyx_mp_ass_subscript_8PyClical_index_set, /*mp_ass_subscript*/
19962 };
19963
19964 static PyTypeObject __pyx_type_8PyClical_index_set = {
19965 PyVarObject_HEAD_INIT(0, 0)
19966 "PyClical.index_set", /*tp_name*/
19967 sizeof(struct __pyx_obj_8PyClical_index_set), /*tp_basicsize*/
19968 0, /*tp_itemsize*/
19969 __pyx_tp_dealloc_8PyClical_index_set, /*tp_dealloc*/

```

```

19970 #if PY_VERSION_HEX < 0x030800b4
19971 0, /*tp_print*/
19972 #endif
19973 #if PY_VERSION_HEX >= 0x030800b4
19974 0, /*tp_vectorcall_offset*/
19975 #endif
19976 0, /*tp_getattr*/
19977 0, /*tp_setattr*/
19978 #if PY_MAJOR_VERSION < 3
19979 0, /*tp_compare*/
19980 #endif
19981 #if PY_MAJOR_VERSION >= 3
19982 0, /*tp_as_async*/
19983 #endif
19984 __pyx_pw_8PyClical_9index_set_48__repr__, /*tp_repr*/
19985 &__pyx_tp_as_number_index_set, /*tp_as_number*/
19986 &__pyx_tp_as_sequence_index_set, /*tp_as_sequence*/
19987 &__pyx_tp_as_mapping_index_set, /*tp_as_mapping*/
19988 0, /*tp_hash*/
19989 0, /*tp_call*/
19990 __pyx_pw_8PyClical_9index_set_50__str__, /*tp_str*/
19991 0, /*tp_getattro*/
19992 0, /*tp_setattro*/
19993 0, /*tp_as_buffer*/
19994
19995 Py_TPFLAGS_DEFAULT|Py_TPFLAGS_HAVE_VERSION_TAG|Py_TPFLAGS_CHECKTYPES|Py_TPFLAGS_HAVE_NEWBUFFER|Py_TPFLAGS_BASETYPE,
19996 /*tp_flags*/
19997
19998 "\n Python class index_set wraps C++ class IndexSet.\n ", /*tp_doc*/
19999 0, /*tp_traverse*/
20000 0, /*tp_clear*/
20001 __pyx_pw_8PyClical_9index_set_7__richcmp__, /*tp_richcompare*/
20002 0, /*tp_weaklistoffset*/
20003 __pyx_pw_8PyClical_9index_set_15__iter__, /*tp_iter*/
20004 0, /*tp_iternext*/
20005 __pyx_methods_8PyClical_index_set, /*tp_methods*/
20006 0, /*tp_members*/
20007 0, /*tp_getset*/
20008 0, /*tp_base*/
20009 0, /*tp_dict*/
20010 0, /*tp_descr_get*/
20011 0, /*tp_descr_set*/
20012 0, /*tp_dictoffset*/
20013 0, /*tp_init*/
20014 0, /*tp_alloc*/
20015 __pyx_tp_new_8PyClical_index_set, /*tp_new*/
20016 0, /*tp_free*/
20017 0, /*tp_is_gc*/
20018 0, /*tp_bases*/
20019 0, /*tp_mro*/
20020 0, /*tp_cache*/
20021 0, /*tp_subclasses*/
20022 0, /*tp_weaklist*/
20023 0, /*tp_del*/
20024 0, /*tp_version_tag*/
20025 #if PY_VERSION_HEX >= 0x030400a1
20026 0, /*tp_finalize*/
20027 #endif
20028 #if PY_VERSION_HEX >= 0x030800b1 && (!CYTHON_COMPILING_IN_PYPY || PYPY_VERSION_NUM
20029 >= 0x07030800)
20030 0, /*tp_vectorcall*/
20031 #endif
20032 #if PY_VERSION_HEX >= 0x030800b4 && PY_VERSION_HEX < 0x03090000
20033 0, /*tp_print*/
20034 #endif
20035 #if CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX >= 0x03090000
20036 0, /*tp_pypy_flags*/
20037 #endif
20038 };
20039 static struct __pyx_vtabstruct_8PyClical_clifford __pyx_vtable_8PyClical_clifford;
20040
20041 static PyObject * __pyx_tp_new_8PyClical_clifford(PyTypeObject *t, PyObject *a,
20042 PyObject *k) {
20043 struct __pyx_obj_8PyClical_clifford *p;
20044 PyObject *o;
20045 if (likely((t->tp_flags & Py_TPFLAGS_IS_ABSTRACT) == 0)) {
20046 o = (*t->tp_alloc)(t, 0);
20047 } else {
20048 o = (PyObject *) PyBaseObject_Type.tp_new(t, __pyx_empty_tuple, 0);
20049 }
20050 if (unlikely(!o)) return 0;
20051 p = ((struct __pyx_obj_8PyClical_clifford *)o);
20052 p->__pyx_vtab = __pyx_vtabptr_8PyClical_clifford;
20053 if (unlikely(__pyx_pw_8PyClical_8clifford_3__cinit__(o, a, k) < 0)) goto bad;
20054 return o;
20055 bad:
20056 Py_DECREF(o); o = 0;
20057 return NULL;

```

```

20053 }
20054
20055 static void __pyx_tp_dealloc_8PyClical_clifford(PyObject *o) {
20056 #if CYTHON_USE_TP_FINALIZE
20057 if (unlikely(PyType_HasFeature(Py_TYPE(o), Py_TPFLAGS_HAVE_FINALIZE) &&
Py_TYPE(o)->tp_finalize) && (!PyType_IS_GC(Py_TYPE(o)) || !_PyGC_FINALIZED(o))) {
20058 if (PyObject_CallFinalizerFromDealloc(o)) return;
20059 }
20060 #endif
20061 {
20062 PyObject *etype, *eval, *etb;
20063 PyErr_Fetch(&etype, &eval, &etb);
20064 __Pyx_SET_REFCNT(o, Py_REFCNT(o) + 1);
20065 __pyx_pw_8PyClical_8clifford_5__dealloc__(o);
20066 __Pyx_SET_REFCNT(o, Py_REFCNT(o) - 1);
20067 PyErr_Restore(etype, eval, etb);
20068 }
20069 (*Py_TYPE(o)->tp_free)(o);
20070 }
20071
20072 static PyObject * __pyx_sq_item_8PyClical_clifford(PyObject *o, Py_ssize_t i) {
20073 PyObject *r;
20074 PyObject *x = PyInt_FromSsize_t(i); if(!x) return 0;
20075 r = Py_TYPE(o)->tp_as_mapping->mp_subscript(o, x);
20076 Py_DECREF(x);
20077 return r;
20078 }
20079
20080 static PyMethodDef __pyx_methods_8PyClical_clifford[] = {
20081 {"copy", (PyCFunction) __pyx_pw_8PyClical_8clifford_1copy, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_copy},
20082 {"reframe", (PyCFunction) __pyx_pw_8PyClical_8clifford_11reframe, METH_O,
__pyx_doc_8PyClical_8clifford_10reframe},
20083 {"inv", (PyCFunction) __pyx_pw_8PyClical_8clifford_49inv, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_48inv},
20084 {"pow", (PyCFunction) __pyx_pw_8PyClical_8clifford_57pow, METH_O,
__pyx_doc_8PyClical_8clifford_56pow},
20085 {"outer_pow", (PyCFunction) __pyx_pw_8PyClical_8clifford_59outer_pow, METH_O,
__pyx_doc_8PyClical_8clifford_58outer_pow},
20086 {"scalar", (PyCFunction) __pyx_pw_8PyClical_8clifford_63scalar, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_62scalar},
20087 {"pure", (PyCFunction) __pyx_pw_8PyClical_8clifford_65pure, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_64pure},
20088 {"even", (PyCFunction) __pyx_pw_8PyClical_8clifford_67even, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_66even},
20089 {"odd", (PyCFunction) __pyx_pw_8PyClical_8clifford_69odd, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_68odd},
20090 {"vector_part",
(PyCFunction) (void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_8clifford_71vector_part,
METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_8clifford_70vector_part},
20091 {"involute", (PyCFunction) __pyx_pw_8PyClical_8clifford_73involute, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_72involute},
20092 {"reverse", (PyCFunction) __pyx_pw_8PyClical_8clifford_75reverse, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_74reverse},
20093 {"conj", (PyCFunction) __pyx_pw_8PyClical_8clifford_77conj, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_76conj},
20094 {"quad", (PyCFunction) __pyx_pw_8PyClical_8clifford_79quad, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_78quad},
20095 {"norm", (PyCFunction) __pyx_pw_8PyClical_8clifford_81norm, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_80norm},
20096 {"abs", (PyCFunction) __pyx_pw_8PyClical_8clifford_83abs, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_82abs},
20097 {"max_abs", (PyCFunction) __pyx_pw_8PyClical_8clifford_85max_abs, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_84max_abs},
20098 {"truncated", (PyCFunction) __pyx_pw_8PyClical_8clifford_87truncated, METH_O,
__pyx_doc_8PyClical_8clifford_86truncated},
20099 {"isinf", (PyCFunction) __pyx_pw_8PyClical_8clifford_89isinf, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_88isinf},
20100 {"isnan", (PyCFunction) __pyx_pw_8PyClical_8clifford_91isnan, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_90isnan},
20101 {"frame", (PyCFunction) __pyx_pw_8PyClical_8clifford_93frame, METH_NOARGS,
__pyx_doc_8PyClical_8clifford_92frame},
20102 {"__reduce_cython__", (PyCFunction) __pyx_pw_8PyClical_8clifford_99__reduce_cython__,
METH_NOARGS, 0},
20103 {"__setstate_cython__",
(PyCFunction) __pyx_pw_8PyClical_8clifford_101__setstate_cython__, METH_O, 0},
20104 {0, 0, 0, 0};
20105
20106 static PyNumberMethods __pyx_tp_as_number_clifford = {
20107 __pyx_pw_8PyClical_8clifford_21__add__, /*nb_add*/
20108 __pyx_pw_8PyClical_8clifford_25__sub__, /*nb_subtract*/
20109 __pyx_pw_8PyClical_8clifford_29__mul__, /*nb_multiply*/
20110 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX <
0x03050000)
20111 0, /*nb_divide*/
20112 #endif
20113 __pyx_pw_8PyClical_8clifford_33__mod__, /*nb_remainder*/

```

```

20114 0, /*nb_divmod*/
20115 __pyx_pw_8PyClical_8clifford_55_pow__, /*nb_power*/
20116 __pyx_pw_8PyClical_8clifford_17_neg__, /*nb_negative*/
20117 __pyx_pw_8PyClical_8clifford_19_pos__, /*nb_positive*/
20118 0, /*nb_absolute*/
20119 0, /*nb_nonzero*/
20120 0, /*nb_invert*/
20121 0, /*nb_lshift*/
20122 0, /*nb_rshift*/
20123 __pyx_pw_8PyClical_8clifford_37_and__, /*nb_and*/
20124 __pyx_pw_8PyClical_8clifford_41_xor__, /*nb_xor*/
20125 __pyx_pw_8PyClical_8clifford_51_or__, /*nb_or*/
20126 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX <
0x03050000)
20127 0, /*nb_coerce*/
20128 #endif
20129 0, /*nb_int*/
20130 #if PY_MAJOR_VERSION < 3
20131 0, /*nb_long*/
20132 #else
20133 0, /*reserved*/
20134 #endif
20135 0, /*nb_float*/
20136 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX <
0x03050000)
20137 0, /*nb_oct*/
20138 #endif
20139 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX <
0x03050000)
20140 0, /*nb_hex*/
20141 #endif
20142 __pyx_pw_8PyClical_8clifford_23_iadd__, /*nb_inplace_add*/
20143 __pyx_pw_8PyClical_8clifford_27_isub__, /*nb_inplace_subtract*/
20144 __pyx_pw_8PyClical_8clifford_31_imul__, /*nb_inplace_multiply*/
20145 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX <
0x03050000)
20146 __pyx_pw_8PyClical_8clifford_47_idiv__, /*nb_inplace_divide*/
20147 #endif
20148 __pyx_pw_8PyClical_8clifford_35_imod__, /*nb_inplace_remainder*/
20149 0, /*nb_inplace_power*/
20150 0, /*nb_inplace_lshift*/
20151 0, /*nb_inplace_rshift*/
20152 __pyx_pw_8PyClical_8clifford_39_iand__, /*nb_inplace_and*/
20153 __pyx_pw_8PyClical_8clifford_43_ixor__, /*nb_inplace_xor*/
20154 __pyx_pw_8PyClical_8clifford_53_ior__, /*nb_inplace_or*/
20155 0, /*nb_floor_divide*/
20156 __pyx_pw_8PyClical_8clifford_45_truediv__, /*nb_true_divide*/
20157 0, /*nb_inplace_floor_divide*/
20158 0, /*nb_inplace_true_divide*/
20159 0, /*nb_index*/
20160 #if PY_VERSION_HEX >= 0x03050000
20161 0, /*nb_matrix_multiply*/
20162 #endif
20163 #if PY_VERSION_HEX >= 0x03050000
20164 0, /*nb_inplace_matrix_multiply*/
20165 #endif
20166 };
20167
20168 static PySequenceMethods __pyx_tp_as_sequence_clifford = {
20169 0, /*sq_length*/
20170 0, /*sq_concat*/
20171 0, /*sq_repeat*/
20172 __pyx_sq_item_8PyClical_clifford, /*sq_item*/
20173 0, /*sq_slice*/
20174 0, /*sq_ass_item*/
20175 0, /*sq_ass_slice*/
20176 __pyx_pw_8PyClical_8clifford_7_contains__, /*sq_contains*/
20177 0, /*sq_inplace_concat*/
20178 0, /*sq_inplace_repeat*/
20179 };
20180
20181 static PyMappingMethods __pyx_tp_as_mapping_clifford = {
20182 0, /*mp_length*/
20183 __pyx_pw_8PyClical_8clifford_15_getitem__, /*mp_subscript*/
20184 0, /*mp_ass_subscript*/
20185 };
20186
20187 static PyTypeObject __pyx_type_8PyClical_clifford = {
20188 PyVarObject_HEAD_INIT(0, 0)
20189 "PyClical.clifford", /*tp_name*/
20190 sizeof(struct __pyx_obj_8PyClical_clifford), /*tp_basicsize*/
20191 0, /*tp_itemsize*/
20192 __pyx_tp_dealloc_8PyClical_clifford, /*tp_dealloc*/
20193 #if PY_VERSION_HEX < 0x030800b4
20194 0, /*tp_print*/
20195 #endif
20196 #if PY_VERSION_HEX >= 0x030800b4

```

```

20197 0, /*tp_vectorcall_offset*/
20198 #endif
20199 0, /*tp_getattr*/
20200 0, /*tp_setattr*/
20201 #if PY_MAJOR_VERSION < 3
20202 0, /*tp_compare*/
20203 #endif
20204 #if PY_MAJOR_VERSION >= 3
20205 0, /*tp_as_async*/
20206 #endif
20207 __pyx_pw_8PyClical_8clifford_95__repr__, /*tp_repr*/
20208 &__pyx_tp_as_number_clifford, /*tp_as_number*/
20209 &__pyx_tp_as_sequence_clifford, /*tp_as_sequence*/
20210 &__pyx_tp_as_mapping_clifford, /*tp_as_mapping*/
20211 0, /*tp_hash*/
20212 __pyx_pw_8PyClical_8clifford_61__call__, /*tp_call*/
20213 __pyx_pw_8PyClical_8clifford_97__str__, /*tp_str*/
20214 0, /*tp_getattro*/
20215 0, /*tp_setattro*/
20216 0, /*tp_as_buffer*/
20217
Py_TPFLAGS_DEFAULT|Py_TPFLAGS_HAVE_VERSION_TAG|Py_TPFLAGS_CHECKTYPES|Py_TPFLAGS_HAVE_NEWBUFFER|Py_TPFLAGS_BASETYPE,
/*tp_flags*/
20218 "\n Python class clifford wraps C++ class Clifford.\n ", /*tp_doc*/
20219 0, /*tp_traverse*/
20220 0, /*tp_clear*/
20221 __pyx_pw_8PyClical_8clifford_13__richcmp__, /*tp_richcompare*/
20222 0, /*tp_weaklistoffset*/
20223 __pyx_pw_8PyClical_8clifford_9__iter__, /*tp_iter*/
20224 0, /*tp_iternext*/
20225 __pyx_methods_8PyClical_clifford, /*tp_methods*/
20226 0, /*tp_members*/
20227 0, /*tp_getset*/
20228 0, /*tp_base*/
20229 0, /*tp_dict*/
20230 0, /*tp_descr_get*/
20231 0, /*tp_descr_set*/
20232 0, /*tp_dictoffset*/
20233 0, /*tp_init*/
20234 0, /*tp_alloc*/
20235 __pyx_tp_new_8PyClical_clifford, /*tp_new*/
20236 0, /*tp_free*/
20237 0, /*tp_is_gc*/
20238 0, /*tp_bases*/
20239 0, /*tp_mro*/
20240 0, /*tp_cache*/
20241 0, /*tp_subclasses*/
20242 0, /*tp_weaklist*/
20243 0, /*tp_del*/
20244 0, /*tp_version_tag*/
20245 #if PY_VERSION_HEX >= 0x030400a1
20246 0, /*tp_finalize*/
20247 #endif
20248 #if PY_VERSION_HEX >= 0x030800b1 && (!CYTHON_COMPILING_IN_PYPY || PYPY_VERSION_NUM
>= 0x07030800)
20249 0, /*tp_vectorcall*/
20250 #endif
20251 #if PY_VERSION_HEX >= 0x030800b4 && PY_VERSION_HEX < 0x03090000
20252 0, /*tp_print*/
20253 #endif
20254 #if CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX >= 0x03090000
20255 0, /*tp_pypy_flags*/
20256 #endif
20257 };
20258
20259 static struct __pyx_obj_8PyClical__pyx_scope_struct____iter__
*_pyx_freelist_8PyClical__pyx_scope_struct____iter__[8];
20260 static int __pyx_freecount_8PyClical__pyx_scope_struct____iter__ = 0;
20261
20262 static PyObject * __pyx_tp_new_8PyClical__pyx_scope_struct____iter__(PyObject *t,
CYTHON_UNUSED PyObject *a, CYTHON_UNUSED PyObject *k) {
20263 PyObject *o;
20264 if (CYTHON_COMPILING_IN_CPYTHON &&
likely((__pyx_freecount_8PyClical__pyx_scope_struct____iter__ > 0) & (t->tp_basicsize ==
sizeof(struct __pyx_obj_8PyClical__pyx_scope_struct____iter__))) {
20265 o =
(PyObject*)__pyx_freelist_8PyClical__pyx_scope_struct____iter__[--__pyx_freecount_8PyClical__pyx_scope_struct____iter];
20266 memset(o, 0, sizeof(struct __pyx_obj_8PyClical__pyx_scope_struct____iter__));
20267 (void) PyObject_INIT(o, t);
20268 PyObject_GC_Track(o);
20269 } else {
20270 o = (*t->tp_alloc)(t, 0);
20271 if (unlikely(!o)) return 0;
20272 }
20273 return o;
20274 }
20275

```



```

20276 static void __pyx_tp_dealloc_8PyClical__pyx_scope_struct____iter__(PyObject *o) {
20277 struct __pyx_obj_8PyClical__pyx_scope_struct____iter__ *p = (struct
__pyx_obj_8PyClical__pyx_scope_struct____iter__ *)o;
20278 PyObject_GC_UnTrack(o);
20279 Py_CLEAR(p->__pyx_v_idx);
20280 Py_CLEAR(p->__pyx_v_self);
20281 Py_CLEAR(p->__pyx_t_0);
20282 if (CYTHON_COMPILING_IN_CPYTHON &&
((__pyx_freecount_8PyClical__pyx_scope_struct____iter__ < 8) & (Py_TYPE(o)->tp_basicsize ==
sizeof(struct __pyx_obj_8PyClical__pyx_scope_struct____iter__))) {
20283 __pyx_freelist_8PyClical__pyx_scope_struct____iter__[__pyx_freecount_8PyClical__pyx_scope_struct____iter__++]
= ((struct __pyx_obj_8PyClical__pyx_scope_struct____iter__ *)o);
20284 } else {
20285 (*Py_TYPE(o)->tp_free)(o);
20286 }
20287 }
20288
20289 static int __pyx_tp_traverse_8PyClical__pyx_scope_struct____iter__(PyObject *o,
visitproc v, void *a) {
20290 int e;
20291 struct __pyx_obj_8PyClical__pyx_scope_struct____iter__ *p = (struct
__pyx_obj_8PyClical__pyx_scope_struct____iter__ *)o;
20292 if (p->__pyx_v_idx) {
20293 e = (*v)(p->__pyx_v_idx, a); if (e) return e;
20294 }
20295 if (p->__pyx_v_self) {
20296 e = (*v)((PyObject *)p->__pyx_v_self, a); if (e) return e;
20297 }
20298 if (p->__pyx_t_0) {
20299 e = (*v)(p->__pyx_t_0, a); if (e) return e;
20300 }
20301 return 0;
20302 }
20303
20304 static PyTypeObject __pyx_type_8PyClical__pyx_scope_struct____iter__ = {
20305 PyVarObject_HEAD_INIT(0, 0)
20306 "PyClical.__pyx_scope_struct____iter__", /*tp_name*/
20307 sizeof(struct __pyx_obj_8PyClical__pyx_scope_struct____iter__), /*tp_basicsize*/
20308 0, /*tp_itemsize*/
20309 __pyx_tp_dealloc_8PyClical__pyx_scope_struct____iter__, /*tp_dealloc*/
20310 #if PY_VERSION_HEX < 0x030800b4
20311 0, /*tp_print*/
20312 #endif
20313 #if PY_VERSION_HEX >= 0x030800b4
20314 0, /*tp_vectorcall_offset*/
20315 #endif
20316 0, /*tp_getattr*/
20317 0, /*tp_setattr*/
20318 #if PY_MAJOR_VERSION < 3
20319 0, /*tp_compare*/
20320 #endif
20321 #if PY_MAJOR_VERSION >= 3
20322 0, /*tp_as_async*/
20323 #endif
20324 0, /*tp_repr*/
20325 0, /*tp_as_number*/
20326 0, /*tp_as_sequence*/
20327 0, /*tp_as_mapping*/
20328 0, /*tp_hash*/
20329 0, /*tp_call*/
20330 0, /*tp_str*/
20331 0, /*tp_getattro*/
20332 0, /*tp_setattro*/
20333 0, /*tp_as_buffer*/
20334 Py_TPFLAGS_DEFAULT|Py_TPFLAGS_HAVE_VERSION_TAG|Py_TPFLAGS_CHECKTYPES|Py_TPFLAGS_HAVE_NEWBUFFER|Py_TPFLAGS_HAVE_GC,
/*tp_flags*/
20335 0, /*tp_doc*/
20336 __pyx_tp_traverse_8PyClical__pyx_scope_struct____iter__, /*tp_traverse*/
20337 0, /*tp_clear*/
20338 0, /*tp_richcompare*/
20339 0, /*tp_weaklistoffset*/
20340 0, /*tp_iter*/
20341 0, /*tp_iternext*/
20342 0, /*tp_methods*/
20343 0, /*tp_members*/
20344 0, /*tp_getset*/
20345 0, /*tp_base*/
20346 0, /*tp_dict*/
20347 0, /*tp_descr_get*/
20348 0, /*tp_descr_set*/
20349 0, /*tp_dictoffset*/
20350 0, /*tp_init*/
20351 0, /*tp_alloc*/
20352 __pyx_tp_new_8PyClical__pyx_scope_struct____iter__, /*tp_new*/
20353 0, /*tp_free*/

```

```

20354 0, /*tp_is_gc*/
20355 0, /*tp_bases*/
20356 0, /*tp_mro*/
20357 0, /*tp_cache*/
20358 0, /*tp_subclasses*/
20359 0, /*tp_weaklist*/
20360 0, /*tp_del*/
20361 0, /*tp_version_tag*/
20362 #if PY_VERSION_HEX >= 0x030400a1
20363 0, /*tp_finalize*/
20364 #endif
20365 #if PY_VERSION_HEX >= 0x030800b1 && (!CYTHON_COMPILING_IN_PYPY || PYPY_VERSION_NUM
>= 0x07030800)
20366 0, /*tp_vectorcall*/
20367 #endif
20368 #if PY_VERSION_HEX >= 0x030800b4 && PY_VERSION_HEX < 0x03090000
20369 0, /*tp_print*/
20370 #endif
20371 #if CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX >= 0x03090000
20372 0, /*tp_pypy_flags*/
20373 #endif
20374 };
20375
20376 static PyMethodDef __pyx_methods[] = {
20377 {"compare",
20378 (PyCFunction)(void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_3compare, METH_VARARGS|METH_KEYWORDS,
20379 __pyx_doc_8PyClical_2compare},
20380 {"min_neg", (PyCFunction) __pyx_pw_8PyClical_5min_neg, METH_O,
20381 __pyx_doc_8PyClical_4min_neg},
20382 {"max_pos", (PyCFunction) __pyx_pw_8PyClical_7max_pos, METH_O,
20383 __pyx_doc_8PyClical_6max_pos},
20384 {"error_squared_tol", (PyCFunction) __pyx_pw_8PyClical_11error_squared_tol, METH_O,
20385 __pyx_doc_8PyClical_10error_squared_tol},
20386 {"error_squared",
20387 (PyCFunction)(void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_13error_squared,
20388 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_12error_squared},
20389 {"approx_equal",
20390 (PyCFunction)(void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_15approx_equal,
20391 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_14approx_equal},
20392 {"inv", (PyCFunction) __pyx_pw_8PyClical_17inv, METH_O, __pyx_doc_8PyClical_16inv},
20393 {"scalar", (PyCFunction) __pyx_pw_8PyClical_19scalar, METH_O,
20394 __pyx_doc_8PyClical_18scalar},
20395 {"real", (PyCFunction) __pyx_pw_8PyClical_21real, METH_O,
20396 __pyx_doc_8PyClical_20real},
20397 {"imag", (PyCFunction) __pyx_pw_8PyClical_23imag, METH_O,
20398 __pyx_doc_8PyClical_22imag},
20399 {"pure", (PyCFunction) __pyx_pw_8PyClical_25pure, METH_O,
20400 __pyx_doc_8PyClical_24pure},
20401 {"even", (PyCFunction) __pyx_pw_8PyClical_27even, METH_O,
20402 __pyx_doc_8PyClical_26even},
20403 {"odd", (PyCFunction) __pyx_pw_8PyClical_29odd, METH_O, __pyx_doc_8PyClical_28odd},
20404 {"involute", (PyCFunction) __pyx_pw_8PyClical_31involute, METH_O,
20405 __pyx_doc_8PyClical_30involute},
20406 {"reverse", (PyCFunction) __pyx_pw_8PyClical_33reverse, METH_O,
20407 __pyx_doc_8PyClical_32reverse},
20408 {"conj", (PyCFunction) __pyx_pw_8PyClical_35conj, METH_O,
20409 __pyx_doc_8PyClical_34conj},
20410 {"quad", (PyCFunction) __pyx_pw_8PyClical_37quad, METH_O,
20411 __pyx_doc_8PyClical_36quad},
20412 {"norm", (PyCFunction) __pyx_pw_8PyClical_39norm, METH_O,
20413 __pyx_doc_8PyClical_38norm},
20414 {"abs", (PyCFunction) __pyx_pw_8PyClical_41abs, METH_O, __pyx_doc_8PyClical_40abs},
20415 {"max_abs", (PyCFunction) __pyx_pw_8PyClical_43max_abs, METH_O,
20416 __pyx_doc_8PyClical_42max_abs},
20417 {"pow", (PyCFunction)(void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_45pow,
20418 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_44pow},
20419 {"outer_pow",
20420 (PyCFunction)(void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_47outer_pow,
20421 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_46outer_pow},
20422 {"complexifier", (PyCFunction) __pyx_pw_8PyClical_49complexifier, METH_O,
20423 __pyx_doc_8PyClical_48complexifier},
20424 {"sqrt", (PyCFunction)(void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_51sqrt,
20425 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_50sqrt},
20426 {"exp", (PyCFunction) __pyx_pw_8PyClical_53exp, METH_O, __pyx_doc_8PyClical_52exp},
20427 {"log", (PyCFunction)(void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_55log,
20428 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_54log},
20429 {"cos", (PyCFunction)(void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_57cos,
20430 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_56cos},
20431 {"acos", (PyCFunction)(void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_59acos,
20432 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_58acos},
20433 {"cosh", (PyCFunction) __pyx_pw_8PyClical_61cosh, METH_O,
20434 __pyx_doc_8PyClical_60cosh},
20435 {"acosh", (PyCFunction)(void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_63acosh,
20436 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_62acosh},
20437 {"sin", (PyCFunction)(void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_65sin,
20438 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_64sin},
20439 {"asin", (PyCFunction)(void*) (PyCFunctionWithKeywords) __pyx_pw_8PyClical_67asin,

```

```

 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_66asin},
20409 {"sinh", (PyCFunction)__pyx_pw_8PyClical_69sinh, METH_O,
 __pyx_doc_8PyClical_68sinh},
20410 {"asinh", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_71asinh,
 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_70asinh},
20411 {"tan", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_73tan,
 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_72tan},
20412 {"atan", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_75atan,
 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_74atan},
20413 {"tanh", (PyCFunction)__pyx_pw_8PyClical_77tanh, METH_O,
 __pyx_doc_8PyClical_76tanh},
20414 {"atanh", (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_79atanh,
 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_78atanh},
20415 {"random_clifford",
 (PyCFunction)(void*)(PyCFunctionWithKeywords)__pyx_pw_8PyClical_81random_clifford,
 METH_VARARGS|METH_KEYWORDS, __pyx_doc_8PyClical_80random_clifford},
20416 {"cga3", (PyCFunction)__pyx_pw_8PyClical_83cga3, METH_O,
 __pyx_doc_8PyClical_82cga3},
20417 {"cga3std", (PyCFunction)__pyx_pw_8PyClical_85cga3std, METH_O,
 __pyx_doc_8PyClical_84cga3std},
20418 {"agc3", (PyCFunction)__pyx_pw_8PyClical_87agc3, METH_O,
 __pyx_doc_8PyClical_86agc3},
20419 {0, 0, 0, 0}
20420 };
20421
20422 #if PY_MAJOR_VERSION >= 3
20423 #if CYTHON_PEP489_MULTI_PHASE_INIT
20424 static PyObject* __pyx_pymod_create(PyObject *spec, PyModuleDef *def); /*proto*/
20425 static int __pyx_pymod_exec_PyClical(PyObject* module); /*proto*/
20426 static PyModuleDef_Slot __pyx_moduledef_slots[] = {
20427 {Py_mod_create, (void*)__pyx_pymod_create},
20428 {Py_mod_exec, (void*)__pyx_pymod_exec_PyClical},
20429 {0, NULL}
20430 };
20431 #endif
20432
20433 static struct PyModuleDef __pyx_moduledef = {
20434 PyModuleDef_HEAD_INIT,
20435 "PyClical",
20436 0, /* m_doc */
20437 #if CYTHON_PEP489_MULTI_PHASE_INIT
20438 0, /* m_size */
20439 #else
20440 -1, /* m_size */
20441 #endif
20442 __pyx_methods /* m_methods */,
20443 #if CYTHON_PEP489_MULTI_PHASE_INIT
20444 __pyx_moduledef_slots, /* m_slots */
20445 #else
20446 NULL, /* m_reload */
20447 #endif
20448 NULL, /* m_traverse */
20449 NULL, /* m_clear */
20450 NULL /* m_free */
20451 };
20452 #endif
20453 #ifndef CYTHON_SMALL_CODE
20454 #if defined(__clang__)
20455 #define CYTHON_SMALL_CODE
20456 #elif defined(__GNUC__) && (__GNUC__ > 4 || (__GNUC__ == 4 && __GNUC_MINOR__ >= 3))
20457 #define CYTHON_SMALL_CODE __attribute__((cold))
20458 #else
20459 #define CYTHON_SMALL_CODE
20460 #endif
20461 #endif
20462
20463 static __Pyx_StringTabEntry __pyx_string_tab[] = {
20464 {&__pyx_kp_u_, __pyx_k_, sizeof(__pyx_k_), 0, 1, 0, 0},
20465 {&__pyx_kp_u_Abbreviation_for_clifford_index, __pyx_k_Abbreviation_for_clifford_index,
 sizeof(__pyx_k_Abbreviation_for_clifford_index), 0, 1, 0, 0},
20466 {&__pyx_kp_u_Abbreviation_for_index_set_q_p, __pyx_k_Abbreviation_for_index_set_q_p,
 sizeof(__pyx_k_Abbreviation_for_index_set_q_p), 0, 1, 0, 0},
20467 {&__pyx_kp_u_Absolute_value_of_multivector_m, __pyx_k_Absolute_value_of_multivector_m,
 sizeof(__pyx_k_Absolute_value_of_multivector_m), 0, 1, 0, 0},
20468 {&__pyx_kp_u_Absolute_value_square_root_of_n, __pyx_k_Absolute_value_square_root_of_n,
 sizeof(__pyx_k_Absolute_value_square_root_of_n), 0, 1, 0, 0},
20469 {&__pyx_kp_u_Cannot_initialize_clifford_objec, __pyx_k_Cannot_initialize_clifford_objec,
 sizeof(__pyx_k_Cannot_initialize_clifford_objec), 0, 1, 0, 0},
20470 {&__pyx_kp_u_Cannot_initialize_index_set_objec, __pyx_k_Cannot_initialize_index_set_objec,
 sizeof(__pyx_k_Cannot_initialize_index_set_objec), 0, 1, 0, 0},
20471 {&__pyx_kp_u_Cannot_reframe, __pyx_k_Cannot_reframe, sizeof(__pyx_k_Cannot_reframe), 0, 1, 0, 0},
20472 {&__pyx_kp_u_Cannot_take_vector_part_of, __pyx_k_Cannot_take_vector_part_of,
 sizeof(__pyx_k_Cannot_take_vector_part_of), 0, 1, 0, 0},
20473 {&__pyx_kp_u_Cardinality_Number_of_indices_i, __pyx_k_Cardinality_Number_of_indices_i,
 sizeof(__pyx_k_Cardinality_Number_of_indices_i), 0, 1, 0, 0},
20474 {&__pyx_kp_u_Check_if_a_multivector_contains, __pyx_k_Check_if_a_multivector_contains,
 sizeof(__pyx_k_Check_if_a_multivector_contains), 0, 1, 0, 0},

```

```

20475 {&__pyx_kp_u_Check_if_a_multivector_contains_2, __pyx_k_Check_if_a_multivector_contains_2,
sizeof(__pyx_k_Check_if_a_multivector_contains_2), 0, 1, 0, 0},
20476 {&__pyx_kp_u_Conjugation_reverse_o_involute, __pyx_k_Conjugation_reverse_o_involute,
sizeof(__pyx_k_Conjugation_reverse_o_involute), 0, 1, 0, 0},
20477 {&__pyx_kp_u_Conjugation_reverse_o_involute_2, __pyx_k_Conjugation_reverse_o_involute_2,
sizeof(__pyx_k_Conjugation_reverse_o_involute_2), 0, 1, 0, 0},
20478 {&__pyx_kp_u_Contraction_print_clifford_1_cl, __pyx_k_Contraction_print_clifford_1_cl,
sizeof(__pyx_k_Contraction_print_clifford_1_cl), 0, 1, 0, 0},
20479 {&__pyx_kp_u_Contraction_x_clifford_1_x_clif, __pyx_k_Contraction_x_clifford_1_x_clif,
sizeof(__pyx_k_Contraction_x_clifford_1_x_clif), 0, 1, 0, 0},
20480 {&__pyx_kp_u_Convert_CGA3_null_vector_to_Euc, __pyx_k_Convert_CGA3_null_vector_to_Euc,
sizeof(__pyx_k_Convert_CGA3_null_vector_to_Euc), 0, 1, 0, 0},
20481 {&__pyx_kp_u_Convert_CGA3_null_vector_to_sta, __pyx_k_Convert_CGA3_null_vector_to_sta,
sizeof(__pyx_k_Convert_CGA3_null_vector_to_sta), 0, 1, 0, 0},
20482 {&__pyx_kp_u_Convert_Euclidean_3D_multivecto, __pyx_k_Convert_Euclidean_3D_multivecto,
sizeof(__pyx_k_Convert_Euclidean_3D_multivecto), 0, 1, 0, 0},
20483 {&__pyx_kp_u_Copy_this_clifford_object_x_cli, __pyx_k_Copy_this_clifford_object_x_cli,
sizeof(__pyx_k_Copy_this_clifford_object_x_cli), 0, 1, 0, 0},
20484 {&__pyx_kp_u_Copy_this_index_set_object_s_in, __pyx_k_Copy_this_index_set_object_s_in,
sizeof(__pyx_k_Copy_this_index_set_object_s_in), 0, 1, 0, 0},
20485 {&__pyx_kp_u_Cosine_of_multivector_with_opti, __pyx_k_Cosine_of_multivector_with_opti,
sizeof(__pyx_k_Cosine_of_multivector_with_opti), 0, 1, 0, 0},
20486 {&__pyx_kp_u_Even_part_of_multivector_sum_of, __pyx_k_Even_part_of_multivector_sum_of,
sizeof(__pyx_k_Even_part_of_multivector_sum_of), 0, 1, 0, 0},
20487 {&__pyx_kp_u_Even_part_of_multivector_sum_of_2, __pyx_k_Even_part_of_multivector_sum_of_2,
sizeof(__pyx_k_Even_part_of_multivector_sum_of_2), 0, 1, 0, 0},
20488 {&__pyx_kp_u_Exponential_of_multivector_x_cl, __pyx_k_Exponential_of_multivector_x_cl,
sizeof(__pyx_k_Exponential_of_multivector_x_cl), 0, 1, 0, 0},
20489 {&__pyx_kp_u_Geometric_difference_print_clif, __pyx_k_Geometric_difference_print_clif,
sizeof(__pyx_k_Geometric_difference_print_clif), 0, 1, 0, 0},
20490 {&__pyx_kp_u_Geometric_difference_x_clifford, __pyx_k_Geometric_difference_x_clifford,
sizeof(__pyx_k_Geometric_difference_x_clifford), 0, 1, 0, 0},
20491 {&__pyx_kp_u_Geometric_multiplicative_invers, __pyx_k_Geometric_multiplicative_invers,
sizeof(__pyx_k_Geometric_multiplicative_invers), 0, 1, 0, 0},
20492 {&__pyx_kp_u_Geometric_multiplicative_invers_2, __pyx_k_Geometric_multiplicative_invers_2,
sizeof(__pyx_k_Geometric_multiplicative_invers_2), 0, 1, 0, 0},
20493 {&__pyx_kp_u_Geometric_product_print_cliffor, __pyx_k_Geometric_product_print_cliffor,
sizeof(__pyx_k_Geometric_product_print_cliffor), 0, 1, 0, 0},
20494 {&__pyx_kp_u_Geometric_product_x_clifford_2, __pyx_k_Geometric_product_x_clifford_2,
sizeof(__pyx_k_Geometric_product_x_clifford_2), 0, 1, 0, 0},
20495 {&__pyx_kp_u_Geometric_quotient_print_cliffo, __pyx_k_Geometric_quotient_print_cliffo,
sizeof(__pyx_k_Geometric_quotient_print_cliffo), 0, 1, 0, 0},
20496 {&__pyx_kp_u_Geometric_quotient_x_clifford_1, __pyx_k_Geometric_quotient_x_clifford_1,
sizeof(__pyx_k_Geometric_quotient_x_clifford_1), 0, 1, 0, 0},
20497 {&__pyx_kp_u_Geometric_sum_print_clifford_1, __pyx_k_Geometric_sum_print_clifford_1,
sizeof(__pyx_k_Geometric_sum_print_clifford_1), 0, 1, 0, 0},
20498 {&__pyx_kp_u_Geometric_sum_x_clifford_1_x_cl, __pyx_k_Geometric_sum_x_clifford_1_x_cl,
sizeof(__pyx_k_Geometric_sum_x_clifford_1_x_cl), 0, 1, 0, 0},
20499 {&__pyx_kp_u_Get_the_value_of_an_index_set_o, __pyx_k_Get_the_value_of_an_index_set_o,
sizeof(__pyx_k_Get_the_value_of_an_index_set_o), 0, 1, 0, 0},
20500 {&__pyx_kp_u_Hyperbolic_cosine_of_multivecto, __pyx_k_Hyperbolic_cosine_of_multivecto,
sizeof(__pyx_k_Hyperbolic_cosine_of_multivecto), 0, 1, 0, 0},
20501 {&__pyx_kp_u_Hyperbolic_sine_of_multivector, __pyx_k_Hyperbolic_sine_of_multivector,
sizeof(__pyx_k_Hyperbolic_sine_of_multivector), 0, 1, 0, 0},
20502 {&__pyx_kp_u_Hyperbolic_tangent_of_multivect, __pyx_k_Hyperbolic_tangent_of_multivect,
sizeof(__pyx_k_Hyperbolic_tangent_of_multivect), 0, 1, 0, 0},
20503 {&__pyx_kp_u_Imaginary_part_deprecated_alway, __pyx_k_Imaginary_part_deprecated_alway,
sizeof(__pyx_k_Imaginary_part_deprecated_alway), 0, 1, 0, 0},
20504 {&__pyx_n_s_IndexError, __pyx_k_IndexError, sizeof(__pyx_k_IndexError), 0, 0, 1, 1},
20505 {&__pyx_kp_u_Inner_product_print_clifford_1, __pyx_k_Inner_product_print_clifford_1,
sizeof(__pyx_k_Inner_product_print_clifford_1), 0, 1, 0, 0},
20506 {&__pyx_kp_u_Inner_product_x_clifford_1_x_cl, __pyx_k_Inner_product_x_clifford_1_x_cl,
sizeof(__pyx_k_Inner_product_x_clifford_1_x_cl), 0, 1, 0, 0},
20507 {&__pyx_kp_u_Integer_power_of_multivector_ob, __pyx_k_Integer_power_of_multivector_ob,
sizeof(__pyx_k_Integer_power_of_multivector_ob), 0, 1, 0, 0},
20508 {&__pyx_n_s_Integral, __pyx_k_Integral, sizeof(__pyx_k_Integral), 0, 0, 1, 1},
20509 {&__pyx_kp_u_Inverse_cosine_of_multivector_w, __pyx_k_Inverse_cosine_of_multivector_w,
sizeof(__pyx_k_Inverse_cosine_of_multivector_w), 0, 1, 0, 0},
20510 {&__pyx_kp_u_Inverse_hyperbolic_cosine_of_mu, __pyx_k_Inverse_hyperbolic_cosine_of_mu,
sizeof(__pyx_k_Inverse_hyperbolic_cosine_of_mu), 0, 1, 0, 0},
20511 {&__pyx_kp_u_Inverse_hyperbolic_sine_of_mult, __pyx_k_Inverse_hyperbolic_sine_of_mult,
sizeof(__pyx_k_Inverse_hyperbolic_sine_of_mult), 0, 1, 0, 0},
20512 {&__pyx_kp_u_Inverse_hyperbolic_tangent_of_m, __pyx_k_Inverse_hyperbolic_tangent_of_m,
sizeof(__pyx_k_Inverse_hyperbolic_tangent_of_m), 0, 1, 0, 0},
20513 {&__pyx_kp_u_Inverse_sine_of_multivector_wit, __pyx_k_Inverse_sine_of_multivector_wit,
sizeof(__pyx_k_Inverse_sine_of_multivector_wit), 0, 1, 0, 0},
20514 {&__pyx_kp_u_Inverse_tangent_of_multivector, __pyx_k_Inverse_tangent_of_multivector,
sizeof(__pyx_k_Inverse_tangent_of_multivector), 0, 1, 0, 0},
20515 {&__pyx_kp_u_Iterate_over_the_indices_of_an, __pyx_k_Iterate_over_the_indices_of_an,
sizeof(__pyx_k_Iterate_over_the_indices_of_an), 0, 1, 0, 0},
20516 {&__pyx_kp_u_Main_involution_each_i_is_repla, __pyx_k_Main_involution_each_i_is_repla,
sizeof(__pyx_k_Main_involution_each_i_is_repla), 0, 1, 0, 0},
20517 {&__pyx_kp_u_Main_involution_each_i_is_repla_2, __pyx_k_Main_involution_each_i_is_repla_2,
sizeof(__pyx_k_Main_involution_each_i_is_repla_2), 0, 1, 0, 0},
20518 {&__pyx_kp_u_Maximum_absolute_value_of_coord, __pyx_k_Maximum_absolute_value_of_coord,
sizeof(__pyx_k_Maximum_absolute_value_of_coord), 0, 1, 0, 0},
20519 {&__pyx_kp_u_Maximum_member_index_set_1_1_2, __pyx_k_Maximum_member_index_set_1_1_2,

```

```

 sizeof(__pyx_k_Maximum_member_index_set_1_1_2), 0, 1, 0, 0),
20520 {&__pyx_kp_u_Maximum_of_absolute_values_of_c, __pyx_k_Maximum_of_absolute_values_of_c,
 sizeof(__pyx_k_Maximum_of_absolute_values_of_c), 0, 1, 0, 0),
20521 {&__pyx_kp_u_Maximum_positive_index_or_0_if, __pyx_k_Maximum_positive_index_or_0_if,
 sizeof(__pyx_k_Maximum_positive_index_or_0_if), 0, 1, 0, 0),
20522 {&__pyx_kp_u_Minimum_member_index_set_1_1_2, __pyx_k_Minimum_member_index_set_1_1_2,
 sizeof(__pyx_k_Minimum_member_index_set_1_1_2), 0, 1, 0, 0),
20523 {&__pyx_kp_u_Minimum_negative_index_or_0_if, __pyx_k_Minimum_negative_index_or_0_if,
 sizeof(__pyx_k_Minimum_negative_index_or_0_if), 0, 1, 0, 0),
20524 {&__pyx_kp_u_Natural_logarithm_of_multivecto, __pyx_k_Natural_logarithm_of_multivecto,
 sizeof(__pyx_k_Natural_logarithm_of_multivecto), 0, 1, 0, 0),
20525 {&__pyx_kp_u_Norm_sum_of_squares_of_coordina, __pyx_k_Norm_sum_of_squares_of_coordina,
 sizeof(__pyx_k_Norm_sum_of_squares_of_coordina), 0, 1, 0, 0),
20526 {&__pyx_n_s_NotImplemented, __pyx_k_NotImplemented, sizeof(__pyx_k_NotImplemented), 0, 0, 1, 1},
20527 {&__pyx_kp_u_Not_applicable, __pyx_k_Not_applicable, sizeof(__pyx_k_Not_applicable), 0, 1, 0, 0},
20528 {&__pyx_kp_u_Not_applicable_for_a_in_cliffor, __pyx_k_Not_applicable_for_a_in_cliffor,
 sizeof(__pyx_k_Not_applicable_for_a_in_cliffor), 0, 1, 0, 0),
20529 {&__pyx_kp_u_Number_of_negative_indices_incl, __pyx_k_Number_of_negative_indices_incl,
 sizeof(__pyx_k_Number_of_negative_indices_incl), 0, 1, 0, 0),
20530 {&__pyx_kp_u_Number_of_positive_indices_incl, __pyx_k_Number_of_positive_indices_incl,
 sizeof(__pyx_kp_u_Number_of_positive_indices_incl), 0, 1, 0, 0),
20531 {&__pyx_kp_u_Odd_part_of_multivector_sum_of, __pyx_k_Odd_part_of_multivector_sum_of,
 sizeof(__pyx_kp_u_Odd_part_of_multivector_sum_of), 0, 1, 0, 0),
20532 {&__pyx_kp_u_Odd_part_of_multivector_sum_of_2, __pyx_k_Odd_part_of_multivector_sum_of_2,
 sizeof(__pyx_k_Odd_part_of_multivector_sum_of_2), 0, 1, 0, 0),
20533 {&__pyx_kp_u_Outer_product_power_of_multivec, __pyx_k_Outer_product_power_of_multivec,
 sizeof(__pyx_kp_u_Outer_product_power_of_multivec), 0, 1, 0, 0),
20534 {&__pyx_kp_u_Outer_product_power_x_clifford, __pyx_k_Outer_product_power_x_clifford,
 sizeof(__pyx_kp_u_Outer_product_power_x_clifford), 0, 1, 0, 0),
20535 {&__pyx_kp_u_Outer_product_print_clifford_1, __pyx_k_Outer_product_print_clifford_1,
 sizeof(__pyx_kp_u_Outer_product_print_clifford_1), 0, 1, 0, 0),
20536 {&__pyx_kp_u_Outer_product_x_clifford_1_x_cl, __pyx_k_Outer_product_x_clifford_1_x_cl,
 sizeof(__pyx_kp_u_Outer_product_x_clifford_1_x_cl), 0, 1, 0, 0),
20537 {&__pyx_kp_u_Power_self_to_the_m_x_clifford, __pyx_k_Power_self_to_the_m_x_clifford,
 sizeof(__pyx_kp_u_Power_self_to_the_m_x_clifford), 0, 1, 0, 0),
20538 {&__pyx_kp_u_Power_self_to_the_m_x_clifford_2, __pyx_k_Power_self_to_the_m_x_clifford_2,
 sizeof(__pyx_kp_u_Power_self_to_the_m_x_clifford_2), 0, 1, 0, 0),
20539 {&__pyx_kp_u_Pure_grade_vector_part_print_cl, __pyx_k_Pure_grade_vector_part_print_cl,
 sizeof(__pyx_kp_u_Pure_grade_vector_part_print_cl), 0, 1, 0, 0),
20540 {&__pyx_kp_u_Pure_part_print_clifford_1_1_1, __pyx_k_Pure_part_print_clifford_1_1_1,
 sizeof(__pyx_kp_u_Pure_part_print_clifford_1_1_1), 0, 1, 0, 0),
20541 {&__pyx_kp_u_Pure_part_print_pure_clifford_1, __pyx_k_Pure_part_print_pure_clifford_1,
 sizeof(__pyx_kp_u_Pure_part_print_pure_clifford_1), 0, 1, 0, 0),
20542 {&__pyx_kp_u_Put_self_into_a_larger_frame_co, __pyx_k_Put_self_into_a_larger_frame_co,
 sizeof(__pyx_kp_u_Put_self_into_a_larger_frame_co), 0, 1, 0, 0),
20543 {&__pyx_n_s_PyClical, __pyx_k_PyClical, sizeof(__pyx_k_PyClical), 0, 0, 1, 1},
20544 {&__pyx_kp_s_PyClical_pyx, __pyx_k_PyClical_pyx, sizeof(__pyx_k_PyClical_pyx), 0, 0, 1, 0},
20545 {&__pyx_kp_u_Quadratic_form_rev_x_x_0_print, __pyx_k_Quadratic_form_rev_x_x_0_print,
 sizeof(__pyx_kp_u_Quadratic_form_rev_x_x_0_print), 0, 1, 0, 0),
20546 {&__pyx_kp_u_Quadratic_form_rev_x_x_0_print_2, __pyx_k_Quadratic_form_rev_x_x_0_print_2,
 sizeof(__pyx_kp_u_Quadratic_form_rev_x_x_0_print_2), 0, 1, 0, 0),
20547 {&__pyx_kp_u_Quadratic_norm_error_tolerance, __pyx_k_Quadratic_norm_error_tolerance,
 sizeof(__pyx_kp_u_Quadratic_norm_error_tolerance), 0, 1, 0, 0),
20548 {&__pyx_kp_u_Random_multivector_within_a_fra, __pyx_k_Random_multivector_within_a_fra,
 sizeof(__pyx_kp_u_Random_multivector_within_a_fra), 0, 1, 0, 0),
20549 {&__pyx_n_s_Real, __pyx_k_Real, sizeof(__pyx_k_Real), 0, 0, 1, 1},
20550 {&__pyx_kp_u_Real_part_synonym_for_scalar_pa, __pyx_k_Real_part_synonym_for_scalar_pa,
 sizeof(__pyx_kp_u_Real_part_synonym_for_scalar_pa), 0, 1, 0, 0),
20551 {&__pyx_kp_u_Relative_or_absolute_error_usin, __pyx_k_Relative_or_absolute_error_usin,
 sizeof(__pyx_kp_u_Relative_or_absolute_error_usin), 0, 1, 0, 0),
20552 {&__pyx_kp_u_Remove_all_terms_of_self_with_r, __pyx_k_Remove_all_terms_of_self_with_r,
 sizeof(__pyx_kp_u_Remove_all_terms_of_self_with_r), 0, 1, 0, 0),
20553 {&__pyx_kp_u_Reversion_eg_1_2_2_1_print_reve, __pyx_k_Reversion_eg_1_2_2_1_print_reve,
 sizeof(__pyx_kp_u_Reversion_eg_1_2_2_1_print_reve), 0, 1, 0, 0),
20554 {&__pyx_kp_u_Reversion_eg_clifford_1_cliffor, __pyx_k_Reversion_eg_clifford_1_cliffor,
 sizeof(__pyx_kp_u_Reversion_eg_clifford_1_cliffor), 0, 1, 0, 0),
20555 {&__pyx_n_s_RuntimeError, __pyx_k_RuntimeError, sizeof(__pyx_k_RuntimeError), 0, 0, 1, 1},
20556 {&__pyx_kp_u_Scalar_part_clifford_1_1_1_2_sc, __pyx_k_Scalar_part_clifford_1_1_1_2_sc,
 sizeof(__pyx_kp_u_Scalar_part_clifford_1_1_1_2_sc), 0, 1, 0, 0),
20557 {&__pyx_kp_u_Scalar_part_scalar_clifford_1_1, __pyx_k_Scalar_part_scalar_clifford_1_1,
 sizeof(__pyx_kp_u_Scalar_part_scalar_clifford_1_1), 0, 1, 0, 0),
20558 {&__pyx_n_s_Sequence, __pyx_k_Sequence, sizeof(__pyx_k_Sequence), 0, 0, 1, 1},
20559 {&__pyx_kp_u_Set_complement_not_print_index, __pyx_k_Set_complement_not_print_index,
 sizeof(__pyx_kp_u_Set_complement_not_print_index), 0, 1, 0, 0),
20560 {&__pyx_kp_u_Set_intersection_and_print_inde, __pyx_k_Set_intersection_and_print_inde,
 sizeof(__pyx_kp_u_Set_intersection_and_print_inde), 0, 1, 0, 0),
20561 {&__pyx_kp_u_Set_intersection_and_x_index_se, __pyx_k_Set_intersection_and_x_index_se,
 sizeof(__pyx_kp_u_Set_intersection_and_x_index_se), 0, 1, 0, 0),
20562 {&__pyx_kp_u_Set_the_value_of_an_index_set_o, __pyx_k_Set_the_value_of_an_index_set_o,
 sizeof(__pyx_kp_u_Set_the_value_of_an_index_set_o), 0, 1, 0, 0),
20563 {&__pyx_kp_u_Set_union_or_print_index_set_1, __pyx_k_Set_union_or_print_index_set_1,
 sizeof(__pyx_kp_u_Set_union_or_print_index_set_1), 0, 1, 0, 0),
20564 {&__pyx_kp_u_Set_union_or_x_index_set_1_x_in, __pyx_k_Set_union_or_x_index_set_1_x_in,
 sizeof(__pyx_kp_u_Set_union_or_x_index_set_1_x_in), 0, 1, 0, 0),
20565 {&__pyx_kp_u_Sign_of_geometric_product_of_tw, __pyx_k_Sign_of_geometric_product_of_tw,
 sizeof(__pyx_kp_u_Sign_of_geometric_product_of_tw), 0, 1, 0, 0),
20566 {&__pyx_kp_u_Sign_of_geometric_square_of_a_C, __pyx_k_Sign_of_geometric_square_of_a_C,

```

```

 sizeof(_pyx_k_Sign_of_geometric_square_of_a_C), 0, 1, 0, 0),
20567 {&_pyx_kp_u_Sine_of_multivector_with_option, _pyx_k_Sine_of_multivector_with_option,
 sizeof(_pyx_k_Sine_of_multivector_with_option), 0, 1, 0, 0),
20568 {&_pyx_kp_u_Square_root_of_1_which_commutates, _pyx_k_Square_root_of_1_which_commutates,
 sizeof(_pyx_k_Square_root_of_1_which_commutates), 0, 1, 0, 0),
20569 {&_pyx_kp_u_Square_root_of_multivector_with, _pyx_k_Square_root_of_multivector_with,
 sizeof(_pyx_k_Square_root_of_multivector_with), 0, 1, 0, 0),
20570 {&_pyx_kp_u_Subalgebra_generated_by_all_gen, _pyx_k_Subalgebra_generated_by_all_gen,
 sizeof(_pyx_k_Subalgebra_generated_by_all_gen), 0, 1, 0, 0),
20571 {&_pyx_kp_u_Subscripting_map_from_index_set, _pyx_k_Subscripting_map_from_index_set,
 sizeof(_pyx_k_Subscripting_map_from_index_set), 0, 1, 0, 0),
20572 {&_pyx_kp_u_Symmetric_set_difference_exclus, _pyx_k_Symmetric_set_difference_exclus,
 sizeof(_pyx_k_Symmetric_set_difference_exclus), 0, 1, 0, 0),
20573 {&_pyx_kp_u_Symmetric_set_difference_exclus_2, _pyx_k_Symmetric_set_difference_exclus_2,
 sizeof(_pyx_k_Symmetric_set_difference_exclus_2), 0, 1, 0, 0),
20574 {&_pyx_kp_u_Tangent_of_multivector_with_opt, _pyx_k_Tangent_of_multivector_with_opt,
 sizeof(_pyx_k_Tangent_of_multivector_with_opt), 0, 1, 0, 0),
20575 {&_pyx_kp_u_Test_for_approximate_equality_o, _pyx_k_Test_for_approximate_equality_o,
 sizeof(_pyx_k_Test_for_approximate_equality_o), 0, 1, 0, 0),
20576 {&_pyx_kp_u_Tests_for_functions_that_Doctes, _pyx_k_Tests_for_functions_that_Doctes,
 sizeof(_pyx_k_Tests_for_functions_that_Doctes), 0, 1, 0, 0),
20577 {&_pyx_kp_u_Tests_for_functions_that_Doctes_2, _pyx_k_Tests_for_functions_that_Doctes_2,
 sizeof(_pyx_k_Tests_for_functions_that_Doctes_2), 0, 1, 0, 0),
20578 {&_pyx_kp_u_The_informal_string_representat, _pyx_k_The_informal_string_representat,
 sizeof(_pyx_k_The_informal_string_representat), 0, 1, 0, 0),
20579 {&_pyx_kp_u_The_informal_string_representat_2, _pyx_k_The_informal_string_representat_2,
 sizeof(_pyx_k_The_informal_string_representat_2), 0, 1, 0, 0),
20580 {&_pyx_kp_u_The_official_string_representat, _pyx_k_The_official_string_representat,
 sizeof(_pyx_k_The_official_string_representat), 0, 1, 0, 0),
20581 {&_pyx_kp_u_The_official_string_representat_2, _pyx_k_The_official_string_representat_2,
 sizeof(_pyx_k_The_official_string_representat_2), 0, 1, 0, 0),
20582 {&_pyx_kp_u_This_comparison_operator_is_not, _pyx_k_This_comparison_operator_is_not,
 sizeof(_pyx_k_This_comparison_operator_is_not), 0, 1, 0, 0),
20583 {&_pyx_kp_u_Transform_left_hand_side_using, _pyx_k_Transform_left_hand_side_using,
 sizeof(_pyx_k_Transform_left_hand_side_using), 0, 1, 0, 0),
20584 {&_pyx_kp_u_Transform_left_hand_side_using_2, _pyx_k_Transform_left_hand_side_using_2,
 sizeof(_pyx_k_Transform_left_hand_side_using_2), 0, 1, 0, 0),
20585 {&_pyx_n_s_TypeError, _pyx_k_TypeError, sizeof(_pyx_k_TypeError), 0, 0, 1, 1},
20586 {&_pyx_kp_u_UTF_8, _pyx_k_UTF_8, sizeof(_pyx_k_UTF_8), 0, 1, 0, 0),
20587 {&_pyx_kp_u_Unary_print_clifford_1_1, _pyx_k_Unary_print_clifford_1_1,
 sizeof(_pyx_k_Unary_print_clifford_1_1), 0, 1, 0, 0),
20588 {&_pyx_kp_u_Unary_print_clifford_1_1_2, _pyx_k_Unary_print_clifford_1_1_2,
 sizeof(_pyx_k_Unary_print_clifford_1_1_2), 0, 1, 0, 0),
20589 {&_pyx_n_s_ValueError, _pyx_k_ValueError, sizeof(_pyx_k_ValueError), 0, 0, 1, 1},
20590 {&_pyx_kp_u_Vector_part_of_multivector_as_a, _pyx_k_Vector_part_of_multivector_as_a,
 sizeof(_pyx_k_Vector_part_of_multivector_as_a), 0, 1, 0, 0),
20591 {&_pyx_kp_u_2, _pyx_k_2, sizeof(_pyx_k_2), 0, 1, 0, 0),
20592 {&_pyx_kp_u_5, _pyx_k_5, sizeof(_pyx_k_5), 0, 1, 0, 0),
20593 {&_pyx_kp_u_6, _pyx_k_6, sizeof(_pyx_k_6), 0, 1, 0, 0),
20594 {&_pyx_kp_u_7, _pyx_k_7, sizeof(_pyx_k_7), 0, 1, 0, 0),
20595 {&_pyx_kp_u_8, _pyx_k_8, sizeof(_pyx_k_8), 0, 1, 0, 0),
20596 {&_pyx_kp_u_9, _pyx_k_9, sizeof(_pyx_k_9), 0, 1, 0, 0),
20597 {&_pyx_n_s_abc, _pyx_k_abc, sizeof(_pyx_k_abc), 0, 0, 1, 1},
20598 {&_pyx_kp_u_abs_line_1522, _pyx_k_abs_line_1522, sizeof(_pyx_k_abs_line_1522), 0, 1, 0, 0),
20599 {&_pyx_n_s_acos, _pyx_k_acos, sizeof(_pyx_k_acos), 0, 0, 1, 1},
20600 {&_pyx_kp_u_acos_line_1668, _pyx_k_acos_line_1668, sizeof(_pyx_k_acos_line_1668), 0, 1, 0, 0),
20601 {&_pyx_n_s_acosh, _pyx_k_acosh, sizeof(_pyx_k_acosh), 0, 0, 1, 1},
20602 {&_pyx_kp_u_acosh_line_1705, _pyx_k_acosh_line_1705, sizeof(_pyx_k_acosh_line_1705), 0, 1, 0, 0),
20603 {&_pyx_kp_u_agc3_line_1893, _pyx_k_agc3_line_1893, sizeof(_pyx_k_agc3_line_1893), 0, 1, 0, 0),
20604 {&_pyx_kp_u_approx_equal_line_1359, _pyx_k_approx_equal_line_1359,
 sizeof(_pyx_k_approx_equal_line_1359), 0, 1, 0, 0),
20605 {&_pyx_n_s_args, _pyx_k_args, sizeof(_pyx_k_args), 0, 0, 1, 1},
20606 {&_pyx_kp_u_as_frame, _pyx_k_as_frame, sizeof(_pyx_k_as_frame), 0, 1, 0, 0),
20607 {&_pyx_n_s_asin, _pyx_k_asin, sizeof(_pyx_k_asin), 0, 0, 1, 1},
20608 {&_pyx_kp_u_asin_line_1747, _pyx_k_asin_line_1747, sizeof(_pyx_k_asin_line_1747), 0, 1, 0, 0),
20609 {&_pyx_n_s_asinh, _pyx_k_asinh, sizeof(_pyx_k_asinh), 0, 0, 1, 1},
20610 {&_pyx_kp_u_asinh_line_1782, _pyx_k_asinh_line_1782, sizeof(_pyx_k_asinh_line_1782), 0, 1, 0, 0),
20611 {&_pyx_n_s_atan, _pyx_k_atan, sizeof(_pyx_k_atan), 0, 0, 1, 1},
20612 {&_pyx_kp_u_atan_line_1818, _pyx_k_atan_line_1818, sizeof(_pyx_k_atan_line_1818), 0, 1, 0, 0),
20613 {&_pyx_n_s_atanh, _pyx_k_atanh, sizeof(_pyx_k_atanh), 0, 0, 1, 1},
20614 {&_pyx_kp_u_atanh_line_1847, _pyx_k_atanh_line_1847, sizeof(_pyx_k_atanh_line_1847), 0, 1, 0, 0),
20615 {&_pyx_kp_u_cga3_line_1873, _pyx_k_cga3_line_1873, sizeof(_pyx_k_cga3_line_1873), 0, 1, 0, 0),
20616 {&_pyx_kp_u_cga3std_line_1882, _pyx_k_cga3std_line_1882, sizeof(_pyx_k_cga3std_line_1882), 0, 1,
 0, 0),
20617 {&_pyx_n_s_cl, _pyx_k_cl, sizeof(_pyx_k_cl), 0, 0, 1, 1},
20618 {&_pyx_n_s_clifford, _pyx_k_clifford, sizeof(_pyx_k_clifford), 0, 0, 1, 1},
20619 {&_pyx_kp_u_clifford_add_line_740, _pyx_k_clifford_add_line_740,
 sizeof(_pyx_k_clifford_add_line_740), 0, 1, 0, 0),
20620 {&_pyx_kp_u_clifford_and_line_836, _pyx_k_clifford_and_line_836,
 sizeof(_pyx_k_clifford_and_line_836), 0, 1, 0, 0),
20621 {&_pyx_kp_u_clifford_call_line_1020, _pyx_k_clifford_call_line_1020,
 sizeof(_pyx_k_clifford_call_line_1020), 0, 1, 0, 0),
20622 {&_pyx_kp_u_clifford_getitem_line_707, _pyx_k_clifford_getitem_line_707,
 sizeof(_pyx_k_clifford_getitem_line_707), 0, 1, 0, 0),
20623 {&_pyx_kp_u_clifford_iadd_line_751, _pyx_k_clifford_iadd_line_751,
 sizeof(_pyx_k_clifford_iadd_line_751), 0, 1, 0, 0),
20624 {&_pyx_kp_u_clifford_iand_line_851, _pyx_k_clifford_iand_line_851,

```



```

 sizeof(_pyx_k_clifford_iand_line_851), 0, 1, 0, 0),
20625 {&_pyx_kp_u_clifford_idiv_line_911, _pyx_k_clifford_idiv_line_911,
 sizeof(_pyx_k_clifford_idiv_line_911), 0, 1, 0, 0),
20626 {&_pyx_kp_u_clifford_imod_line_821, _pyx_k_clifford_imod_line_821,
 sizeof(_pyx_k_clifford_imod_line_821), 0, 1, 0, 0),
20627 {&_pyx_kp_u_clifford_imul_line_793, _pyx_k_clifford_imul_line_793,
 sizeof(_pyx_k_clifford_imul_line_793), 0, 1, 0, 0),
20628 {&_pyx_kp_u_clifford_ior_line_950, _pyx_k_clifford_ior_line_950,
 sizeof(_pyx_k_clifford_ior_line_950), 0, 1, 0, 0),
20629 {&_pyx_kp_u_clifford_isub_line_771, _pyx_k_clifford_isub_line_771,
 sizeof(_pyx_k_clifford_isub_line_771), 0, 1, 0, 0),
20630 {&_pyx_kp_u_clifford_iter_line_638, _pyx_k_clifford_iter_line_638,
 sizeof(_pyx_k_clifford_iter_line_638), 0, 1, 0, 0),
20631 {&_pyx_kp_u_clifford_ixor_line_881, _pyx_k_clifford_ixor_line_881,
 sizeof(_pyx_k_clifford_ixor_line_881), 0, 1, 0, 0),
20632 {&_pyx_kp_u_clifford_mod_line_806, _pyx_k_clifford_mod_line_806,
 sizeof(_pyx_k_clifford_mod_line_806), 0, 1, 0, 0),
20633 {&_pyx_kp_u_clifford_mul_line_780, _pyx_k_clifford_mul_line_780,
 sizeof(_pyx_k_clifford_mul_line_780), 0, 1, 0, 0),
20634 {&_pyx_kp_u_clifford_neg_line_722, _pyx_k_clifford_neg_line_722,
 sizeof(_pyx_k_clifford_neg_line_722), 0, 1, 0, 0),
20635 {&_pyx_kp_u_clifford_or_line_939, _pyx_k_clifford_or_line_939,
 sizeof(_pyx_k_clifford_or_line_939), 0, 1, 0, 0),
20636 {&_pyx_kp_u_clifford_pos_line_731, _pyx_k_clifford_pos_line_731,
 sizeof(_pyx_k_clifford_pos_line_731), 0, 1, 0, 0),
20637 {&_pyx_kp_u_clifford_pow_line_961, _pyx_k_clifford_pow_line_961,
 sizeof(_pyx_k_clifford_pow_line_961), 0, 1, 0, 0),
20638 {&_pyx_kp_u_clifford_repr_line_1235, _pyx_k_clifford_repr_line_1235,
 sizeof(_pyx_k_clifford_repr_line_1235), 0, 1, 0, 0),
20639 {&_pyx_kp_u_clifford_str_line_1244, _pyx_k_clifford_str_line_1244,
 sizeof(_pyx_k_clifford_str_line_1244), 0, 1, 0, 0),
20640 {&_pyx_kp_u_clifford_sub_line_760, _pyx_k_clifford_sub_line_760,
 sizeof(_pyx_k_clifford_sub_line_760), 0, 1, 0, 0),
20641 {&_pyx_kp_u_clifford_truediv_line_896, _pyx_k_clifford_truediv_line_896,
 sizeof(_pyx_k_clifford_truediv_line_896), 0, 1, 0, 0),
20642 {&_pyx_kp_u_clifford_xor_line_866, _pyx_k_clifford_xor_line_866,
 sizeof(_pyx_k_clifford_xor_line_866), 0, 1, 0, 0),
20643 {&_pyx_kp_u_clifford_abs_line_1175, _pyx_k_clifford_abs_line_1175,
 sizeof(_pyx_k_clifford_abs_line_1175), 0, 1, 0, 0),
20644 {&_pyx_kp_u_clifford_conj_line_1138, _pyx_k_clifford_conj_line_1138,
 sizeof(_pyx_k_clifford_conj_line_1138), 0, 1, 0, 0),
20645 {&_pyx_kp_u_clifford_copy_line_556, _pyx_k_clifford_copy_line_556,
 sizeof(_pyx_k_clifford_copy_line_556), 0, 1, 0, 0),
20646 {&_pyx_kp_u_clifford_even_line_1061, _pyx_k_clifford_even_line_1061,
 sizeof(_pyx_k_clifford_even_line_1061), 0, 1, 0, 0),
20647 {&_pyx_kp_u_clifford_frame_line_1224, _pyx_k_clifford_frame_line_1224,
 sizeof(_pyx_k_clifford_frame_line_1224), 0, 1, 0, 0),
20648 {&_pyx_n_s_clifford_hidden_doctests, _pyx_k_clifford_hidden_doctests,
 sizeof(_pyx_k_clifford_hidden_doctests), 0, 0, 1, 1),
20649 {&_pyx_kp_u_clifford_hidden_doctests_line_12, _pyx_k_clifford_hidden_doctests_line_12,
 sizeof(_pyx_k_clifford_hidden_doctests_line_12), 0, 1, 0, 0),
20650 {&_pyx_kp_u_clifford_inv_line_926, _pyx_k_clifford_inv_line_926,
 sizeof(_pyx_k_clifford_inv_line_926), 0, 1, 0, 0),
20651 {&_pyx_kp_u_clifford_involute_line_1107, _pyx_k_clifford_involute_line_1107,
 sizeof(_pyx_k_clifford_involute_line_1107), 0, 1, 0, 0),
20652 {&_pyx_kp_u_clifford_isinf_line_1206, _pyx_k_clifford_isinf_line_1206,
 sizeof(_pyx_k_clifford_isinf_line_1206), 0, 1, 0, 0),
20653 {&_pyx_kp_u_clifford_isnan_line_1215, _pyx_k_clifford_isnan_line_1215,
 sizeof(_pyx_k_clifford_isnan_line_1215), 0, 1, 0, 0),
20654 {&_pyx_kp_u_clifford_max_abs_line_1184, _pyx_k_clifford_max_abs_line_1184,
 sizeof(_pyx_k_clifford_max_abs_line_1184), 0, 1, 0, 0),
20655 {&_pyx_kp_u_clifford_norm_line_1164, _pyx_k_clifford_norm_line_1164,
 sizeof(_pyx_k_clifford_norm_line_1164), 0, 1, 0, 0),
20656 {&_pyx_kp_u_clifford_odd_line_1070, _pyx_k_clifford_odd_line_1070,
 sizeof(_pyx_k_clifford_odd_line_1070), 0, 1, 0, 0),
20657 {&_pyx_kp_u_clifford_outer_pow_line_1004, _pyx_k_clifford_outer_pow_line_1004,
 sizeof(_pyx_k_clifford_outer_pow_line_1004), 0, 1, 0, 0),
20658 {&_pyx_kp_u_clifford_pow_line_980, _pyx_k_clifford_pow_line_980,
 sizeof(_pyx_k_clifford_pow_line_980), 0, 1, 0, 0),
20659 {&_pyx_kp_u_clifford_pure_line_1050, _pyx_k_clifford_pure_line_1050,
 sizeof(_pyx_k_clifford_pure_line_1050), 0, 1, 0, 0),
20660 {&_pyx_kp_u_clifford_quad_line_1153, _pyx_k_clifford_quad_line_1153,
 sizeof(_pyx_k_clifford_quad_line_1153), 0, 1, 0, 0),
20661 {&_pyx_kp_u_clifford_reframe_line_649, _pyx_k_clifford_reframe_line_649,
 sizeof(_pyx_k_clifford_reframe_line_649), 0, 1, 0, 0),
20662 {&_pyx_kp_u_clifford_reverse_line_1123, _pyx_k_clifford_reverse_line_1123,
 sizeof(_pyx_k_clifford_reverse_line_1123), 0, 1, 0, 0),
20663 {&_pyx_kp_u_clifford_scalar_line_1039, _pyx_k_clifford_scalar_line_1039,
 sizeof(_pyx_k_clifford_scalar_line_1039), 0, 1, 0, 0),
20664 {&_pyx_kp_u_clifford_truncated_line_1195, _pyx_k_clifford_truncated_line_1195,
 sizeof(_pyx_k_clifford_truncated_line_1195), 0, 1, 0, 0),
20665 {&_pyx_kp_u_clifford_vector_part_line_1079, _pyx_k_clifford_vector_part_line_1079,
 sizeof(_pyx_k_clifford_vector_part_line_1079), 0, 1, 0, 0),
20666 {&_pyx_n_s_cline_in_traceback, _pyx_k_cline_in_traceback, sizeof(_pyx_k_cline_in_traceback), 0,
 0, 1, 1},
20667 {&_pyx_n_s_close, _pyx_k_close, sizeof(_pyx_k_close), 0, 0, 1, 1},
20668 {&_pyx_n_s_collections, _pyx_k_collections, sizeof(_pyx_k_collections), 0, 0, 1, 1},

```

```

20669 {&__pyx_kp_u_compare_line_492, __pyx_k_compare_line_492, sizeof(__pyx_k_compare_line_492), 0, 1, 0,
20670 0},
20671 {&__pyx_kp_u_complexifier_line_1576, __pyx_k_complexifier_line_1576,
20672 sizeof(__pyx_k_complexifier_line_1576), 0, 1, 0, 0},
20673 {&__pyx_n_s_conj, __pyx_k_conj, sizeof(__pyx_k_conj), 0, 0, 1, 1},
20674 {&__pyx_kp_u_conj_line_1485, __pyx_k_conj_line_1485, sizeof(__pyx_k_conj_line_1485), 0, 1, 0, 0},
20675 {&__pyx_n_s_copy, __pyx_k_copy, sizeof(__pyx_k_copy), 0, 0, 1, 1},
20676 {&__pyx_n_s_cos, __pyx_k_cos, sizeof(__pyx_k_cos), 0, 0, 1, 1},
20677 {&__pyx_kp_u_cos_line_1651, __pyx_k_cos_line_1651, sizeof(__pyx_k_cos_line_1651), 0, 1, 0, 0},
20678 {&__pyx_n_s_cosh, __pyx_k_cosh, sizeof(__pyx_k_cosh), 0, 0, 1, 1},
20679 {&__pyx_kp_u_cosh_line_1689, __pyx_k_cosh_line_1689, sizeof(__pyx_k_cosh_line_1689), 0, 1, 0, 0},
20680 {&__pyx_n_s_doctest, __pyx_k_doctest, sizeof(__pyx_k_doctest), 0, 0, 1, 1},
20681 {&__pyx_n_s_e, __pyx_k_e, sizeof(__pyx_k_e), 0, 0, 1, 1},
20682 {&__pyx_kp_u_e_line_1936, __pyx_k_e_line_1936, sizeof(__pyx_k_e_line_1936), 0, 1, 0, 0},
20683 {&__pyx_n_s_encode, __pyx_k_encode, sizeof(__pyx_k_encode), 0, 0, 1, 1},
20684 {&__pyx_kp_u_error_squared_line_1346, __pyx_k_error_squared_line_1346,
20685 sizeof(__pyx_k_error_squared_line_1346), 0, 1, 0, 0},
20686 {&__pyx_kp_u_error_squared_tol_line_1337, __pyx_k_error_squared_tol_line_1337,
20687 sizeof(__pyx_k_error_squared_tol_line_1337), 0, 1, 0, 0},
20688 {&__pyx_n_s_even, __pyx_k_even, sizeof(__pyx_k_even), 0, 0, 1, 1},
20689 {&__pyx_kp_u_even_line_1437, __pyx_k_even_line_1437, sizeof(__pyx_k_even_line_1437), 0, 1, 0, 0},
20690 {&__pyx_n_s_exp, __pyx_k_exp, sizeof(__pyx_k_exp), 0, 0, 1, 1},
20691 {&__pyx_kp_u_exp_line_1614, __pyx_k_exp_line_1614, sizeof(__pyx_k_exp_line_1614), 0, 1, 0, 0},
20692 {&__pyx_n_s_fill, __pyx_k_fill, sizeof(__pyx_k_fill), 0, 0, 1, 1},
20693 {&__pyx_n_s_frm, __pyx_k_frm, sizeof(__pyx_k_frm), 0, 0, 1, 1},
20694 {&__pyx_kp_u_from, __pyx_k_from, sizeof(__pyx_k_from), 0, 1, 0, 0},
20695 {&__pyx_n_s_getstate, __pyx_k_getstate, sizeof(__pyx_k_getstate), 0, 0, 1, 1},
20696 {&__pyx_n_s_grade, __pyx_k_grade, sizeof(__pyx_k_grade), 0, 0, 1, 1},
20697 {&__pyx_n_s_i, __pyx_k_i, sizeof(__pyx_k_i), 0, 0, 1, 1},
20698 {&__pyx_kp_u_imag_line_1415, __pyx_k_imag_line_1415, sizeof(__pyx_k_imag_line_1415), 0, 1, 0, 0},
20699 {&__pyx_n_s_import, __pyx_k_import, sizeof(__pyx_k_import), 0, 0, 1, 1},
20700 {&__pyx_n_s_index_set, __pyx_k_index_set, sizeof(__pyx_k_index_set), 0, 0, 1, 1},
20701 {&__pyx_kp_u_index_set_and_line_271, __pyx_k_index_set_and_line_271,
20702 sizeof(__pyx_k_index_set_and_line_271), 0, 1, 0, 0},
20703 {&__pyx_kp_u_index_set_getitem_line_191, __pyx_k_index_set_getitem_line_191,
20704 sizeof(__pyx_k_index_set_getitem_line_191), 0, 1, 0, 0},
20705 {&__pyx_kp_u_index_set_iand_line_282, __pyx_k_index_set_iand_line_282,
20706 sizeof(__pyx_k_index_set_iand_line_282), 0, 1, 0, 0},
20707 {&__pyx_kp_u_index_set_invert_line_240, __pyx_k_index_set_invert_line_240,
20708 sizeof(__pyx_k_index_set_invert_line_240), 0, 1, 0, 0},
20709 {&__pyx_kp_u_index_set_ior_line_304, __pyx_k_index_set_ior_line_304,
20710 sizeof(__pyx_k_index_set_ior_line_304), 0, 1, 0, 0},
20711 {&__pyx_n_s_index_set_iter, __pyx_k_index_set_iter, sizeof(__pyx_k_index_set_iter), 0, 0, 1,
20712 1},
20713 {&__pyx_kp_u_index_set_iter_line_229, __pyx_k_index_set_iter_line_229,
20714 sizeof(__pyx_k_index_set_iter_line_229), 0, 1, 0, 0},
20715 {&__pyx_kp_u_index_set_ixor_line_260, __pyx_k_index_set_ixor_line_260,
20716 sizeof(__pyx_k_index_set_ixor_line_260), 0, 1, 0, 0},
20717 {&__pyx_kp_u_index_set_or_line_293, __pyx_k_index_set_or_line_293,
20718 sizeof(__pyx_k_index_set_or_line_293), 0, 1, 0, 0},
20719 {&__pyx_kp_u_index_set_repr_line_384, __pyx_k_index_set_repr_line_384,
20720 sizeof(__pyx_k_index_set_repr_line_384), 0, 1, 0, 0},
20721 {&__pyx_kp_u_index_set_setitem_line_179, __pyx_k_index_set_setitem_line_179,
20722 sizeof(__pyx_k_index_set_setitem_line_179), 0, 1, 0, 0},
20723 {&__pyx_kp_u_index_set_str_line_395, __pyx_k_index_set_str_line_395,
20724 sizeof(__pyx_k_index_set_str_line_395), 0, 1, 0, 0},
20725 {&__pyx_kp_u_index_set_xor_line_249, __pyx_k_index_set_xor_line_249,
20726 sizeof(__pyx_k_index_set_xor_line_249), 0, 1, 0, 0},
20727 {&__pyx_kp_u_index_set_copy_line_65, __pyx_k_index_set_copy_line_65,
20728 sizeof(__pyx_k_index_set_copy_line_65), 0, 1, 0, 0},
20729 {&__pyx_kp_u_index_set_count_line_315, __pyx_k_index_set_count_line_315,
20730 sizeof(__pyx_k_index_set_count_line_315), 0, 1, 0, 0},
20731 {&__pyx_kp_u_index_set_count_neg_line_324, __pyx_k_index_set_count_neg_line_324,
20732 sizeof(__pyx_k_index_set_count_neg_line_324), 0, 1, 0, 0},
20733 {&__pyx_kp_u_index_set_count_pos_line_333, __pyx_k_index_set_count_pos_line_333,
20734 sizeof(__pyx_k_index_set_count_pos_line_333), 0, 1, 0, 0},
20735 {&__pyx_n_s_index_set_hidden_doctests, __pyx_k_index_set_hidden_doctests,
20736 sizeof(__pyx_k_index_set_hidden_doctests), 0, 0, 1, 1},
20737 {&__pyx_kp_u_index_set_hidden_doctests_line_4, __pyx_k_index_set_hidden_doctests_line_4,
20738 sizeof(__pyx_k_index_set_hidden_doctests_line_4), 0, 1, 0, 0},
20739 {&__pyx_kp_u_index_set_max_line_351, __pyx_k_index_set_max_line_351,
20740 sizeof(__pyx_k_index_set_max_line_351), 0, 1, 0, 0},
20741 {&__pyx_kp_u_index_set_min_line_342, __pyx_k_index_set_min_line_342,
20742 sizeof(__pyx_k_index_set_min_line_342), 0, 1, 0, 0},
20743 {&__pyx_kp_u_index_set_sign_of_mult_line_366, __pyx_k_index_set_sign_of_mult_line_366,
20744 sizeof(__pyx_k_index_set_sign_of_mult_line_366), 0, 1, 0, 0},
20745 {&__pyx_kp_u_index_set_sign_of_square_line_37, __pyx_k_index_set_sign_of_square_line_37,
20746 sizeof(__pyx_k_index_set_sign_of_square_line_37), 0, 1, 0, 0},
20747 {&__pyx_n_s_inv, __pyx_k_inv, sizeof(__pyx_k_inv), 0, 0, 1, 1},
20748 {&__pyx_kp_u_inv_line_1378, __pyx_k_inv_line_1378, sizeof(__pyx_k_inv_line_1378), 0, 1, 0, 0},
20749 {&__pyx_kp_u_invalid, __pyx_k_invalid, sizeof(__pyx_k_invalid), 0, 1, 0, 0},
20750 {&__pyx_kp_u_invalid_string, __pyx_k_invalid_string, sizeof(__pyx_k_invalid_string), 0, 1, 0, 0},
20751 {&__pyx_n_s_involute, __pyx_k_involute, sizeof(__pyx_k_involute), 0, 0, 1, 1},
20752 {&__pyx_kp_u_involute_line_1455, __pyx_k_involute_line_1455, sizeof(__pyx_k_involute_line_1455), 0,
20753 1, 0, 0},
20754 {&__pyx_n_s_ist, __pyx_k_ist, sizeof(__pyx_k_ist), 0, 0, 1, 1},
20755 {&__pyx_n_s_istpq, __pyx_k_istpq, sizeof(__pyx_k_istpq), 0, 0, 1, 1},

```



```

20728 {&_pyx_kp_u_istpq_line_1949, __pyx_k_istpq_line_1949, sizeof(__pyx_k_istpq_line_1949), 0, 1, 0, 0},
20729 {&_pyx_n_s_iter, __pyx_k_iter, sizeof(__pyx_k_iter), 0, 0, 1, 1},
20730 {&_pyx_n_s_ixt, __pyx_k_ixt, sizeof(__pyx_k_ixt), 0, 0, 1, 1},
20731 {&_pyx_kp_u_lexicographic_compare_eg_3_4_5, __pyx_k_lexicographic_compare_eg_3_4_5,
sizeof(__pyx_k_lexicographic_compare_eg_3_4_5), 0, 1, 0, 0},
20732 {&_pyx_n_s_lhs, __pyx_k_lhs, sizeof(__pyx_k_lhs), 0, 0, 1, 1},
20733 {&_pyx_n_s_log, __pyx_k_log, sizeof(__pyx_k_log), 0, 0, 1, 1},
20734 {&_pyx_kp_u_log_line_1628, __pyx_k_log_line_1628, sizeof(__pyx_k_log_line_1628), 0, 1, 0, 0},
20735 {&_pyx_n_s_m, __pyx_k_m, sizeof(__pyx_k_m), 0, 0, 1, 1},
20736 {&_pyx_n_s_main, __pyx_k_main, sizeof(__pyx_k_main), 0, 0, 1, 1},
20737 {&_pyx_n_u_main, __pyx_k_main, sizeof(__pyx_k_main), 0, 1, 0, 1},
20738 {&_pyx_n_s_math, __pyx_k_math, sizeof(__pyx_k_math), 0, 0, 1, 1},
20739 {&_pyx_n_s_max, __pyx_k_max, sizeof(__pyx_k_max), 0, 0, 1, 1},
20740 {&_pyx_kp_u_max_abs_line_1531, __pyx_k_max_abs_line_1531, sizeof(__pyx_k_max_abs_line_1531), 0, 1,
0, 0},
20741 {&_pyx_kp_u_max_pos_line_513, __pyx_k_max_pos_line_513, sizeof(__pyx_k_max_pos_line_513), 0, 1, 0,
0},
20742 {&_pyx_n_s_min, __pyx_k_min, sizeof(__pyx_k_min), 0, 0, 1, 1},
20743 {&_pyx_kp_u_min_neg_line_504, __pyx_k_min_neg_line_504, sizeof(__pyx_k_min_neg_line_504), 0, 1, 0,
0},
20744 {&_pyx_n_s_name, __pyx_k_name, sizeof(__pyx_k_name), 0, 0, 1, 1},
20745 {&_pyx_n_s_nbar3, __pyx_k_nbar3, sizeof(__pyx_k_nbar3), 0, 0, 1, 1},
20746 {&_pyx_n_s_ninf3, __pyx_k_ninf3, sizeof(__pyx_k_ninf3), 0, 0, 1, 1},
20747 {&_pyx_kp_u_no_default_reduce_due_to_non, __pyx_k_no_default_reduce_due_to_non,
sizeof(__pyx_k_no_default_reduce_due_to_non), 0, 0, 1, 0},
20748 {&_pyx_n_s_norm, __pyx_k_norm, sizeof(__pyx_k_norm), 0, 0, 1, 1},
20749 {&_pyx_kp_u_norm_line_1511, __pyx_k_norm_line_1511, sizeof(__pyx_k_norm_line_1511), 0, 1, 0, 0},
20750 {&_pyx_kp_u_norm_sum_of_squares_of_coordina, __pyx_k_norm_sum_of_squares_of_coordina,
sizeof(__pyx_k_norm_sum_of_squares_of_coordina), 0, 1, 0, 0},
20751 {&_pyx_n_s_numbers, __pyx_k_numbers, sizeof(__pyx_k_numbers), 0, 0, 1, 1},
20752 {&_pyx_n_s_obj, __pyx_k_obj, sizeof(__pyx_k_obj), 0, 0, 1, 1},
20753 {&_pyx_n_s_odd, __pyx_k_odd, sizeof(__pyx_k_odd), 0, 0, 1, 1},
20754 {&_pyx_kp_u_odd_line_1446, __pyx_k_odd_line_1446, sizeof(__pyx_k_odd_line_1446), 0, 1, 0, 0},
20755 {&_pyx_n_s_other, __pyx_k_other, sizeof(__pyx_k_other), 0, 0, 1, 1},
20756 {&_pyx_n_s_outer_pow, __pyx_k_outer_pow, sizeof(__pyx_k_outer_pow), 0, 0, 1, 1},
20757 {&_pyx_kp_u_outer_pow_line_1567, __pyx_k_outer_pow_line_1567, sizeof(__pyx_k_outer_pow_line_1567),
0, 1, 0, 0},
20758 {&_pyx_n_s_p, __pyx_k_p, sizeof(__pyx_k_p), 0, 0, 1, 1},
20759 {&_pyx_n_s_pi, __pyx_k_pi, sizeof(__pyx_k_pi), 0, 0, 1, 1},
20760 {&_pyx_n_s_pow, __pyx_k_pow, sizeof(__pyx_k_pow), 0, 0, 1, 1},
20761 {&_pyx_kp_u_pow_line_1543, __pyx_k_pow_line_1543, sizeof(__pyx_k_pow_line_1543), 0, 1, 0, 0},
20762 {&_pyx_n_s_pure, __pyx_k_pure, sizeof(__pyx_k_pure), 0, 0, 1, 1},
20763 {&_pyx_kp_u_pure_line_1426, __pyx_k_pure_line_1426, sizeof(__pyx_k_pure_line_1426), 0, 1, 0, 0},
20764 {&_pyx_n_s_pyx_vtable, __pyx_k_pyx_vtable, sizeof(__pyx_k_pyx_vtable), 0, 0, 1, 1},
20765 {&_pyx_n_s_q, __pyx_k_q, sizeof(__pyx_k_q), 0, 0, 1, 1},
20766 {&_pyx_n_s_quad, __pyx_k_quad, sizeof(__pyx_k_quad), 0, 0, 1, 1},
20767 {&_pyx_kp_u_quad_line_1500, __pyx_k_quad_line_1500, sizeof(__pyx_k_quad_line_1500), 0, 1, 0, 0},
20768 {&_pyx_kp_u_random_clifford_line_1864, __pyx_k_random_clifford_line_1864,
sizeof(__pyx_k_random_clifford_line_1864), 0, 1, 0, 0},
20769 {&_pyx_n_s_range, __pyx_k_range, sizeof(__pyx_k_range), 0, 0, 1, 1},
20770 {&_pyx_kp_u_real_line_1404, __pyx_k_real_line_1404, sizeof(__pyx_k_real_line_1404), 0, 1, 0, 0},
20771 {&_pyx_n_s_reduce, __pyx_k_reduce, sizeof(__pyx_k_reduce), 0, 0, 1, 1},
20772 {&_pyx_n_s_reduce_cython, __pyx_k_reduce_cython, sizeof(__pyx_k_reduce_cython), 0, 0, 1, 1},
20773 {&_pyx_n_s_reduce_ex, __pyx_k_reduce_ex, sizeof(__pyx_k_reduce_ex), 0, 0, 1, 1},
20774 {&_pyx_n_s_reverse, __pyx_k_reverse, sizeof(__pyx_k_reverse), 0, 0, 1, 1},
20775 {&_pyx_kp_u_reverse_line_1470, __pyx_k_reverse_line_1470, sizeof(__pyx_k_reverse_line_1470), 0, 1,
0, 0},
20776 {&_pyx_n_s_rhs, __pyx_k_rhs, sizeof(__pyx_k_rhs), 0, 0, 1, 1},
20777 {&_pyx_n_s_scalar, __pyx_k_scalar, sizeof(__pyx_k_scalar), 0, 0, 1, 1},
20778 {&_pyx_n_s_scalar_epsilon, __pyx_k_scalar_epsilon, sizeof(__pyx_k_scalar_epsilon), 0, 0, 1, 1},
20779 {&_pyx_kp_u_scalar_line_1393, __pyx_k_scalar_line_1393, sizeof(__pyx_k_scalar_line_1393), 0, 1, 0,
0},
20780 {&_pyx_n_s_send, __pyx_k_send, sizeof(__pyx_k_send), 0, 0, 1, 1},
20781 {&_pyx_n_s_setstate, __pyx_k_setstate, sizeof(__pyx_k_setstate), 0, 0, 1, 1},
20782 {&_pyx_n_s_setstate_cython, __pyx_k_setstate_cython, sizeof(__pyx_k_setstate_cython), 0, 0, 1, 1},
20783 {&_pyx_n_s_sin, __pyx_k_sin, sizeof(__pyx_k_sin), 0, 0, 1, 1},
20784 {&_pyx_kp_u_sin_line_1728, __pyx_k_sin_line_1728, sizeof(__pyx_k_sin_line_1728), 0, 1, 0, 0},
20785 {&_pyx_n_s_sinh, __pyx_k_sinh, sizeof(__pyx_k_sinh), 0, 0, 1, 1},
20786 {&_pyx_kp_u_sinh_line_1768, __pyx_k_sinh_line_1768, sizeof(__pyx_k_sinh_line_1768), 0, 1, 0, 0},
20787 {&_pyx_n_s_sqrt, __pyx_k_sqrt, sizeof(__pyx_k_sqrt), 0, 0, 1, 1},
20788 {&_pyx_kp_u_sqrt_line_1591, __pyx_k_sqrt_line_1591, sizeof(__pyx_k_sqrt_line_1591), 0, 1, 0, 0},
20789 {&_pyx_n_s_tan, __pyx_k_tan, sizeof(__pyx_k_tan), 0, 0, 1, 1},
20790 {&_pyx_kp_u_tan_line_1801, __pyx_k_tan_line_1801, sizeof(__pyx_k_tan_line_1801), 0, 1, 0, 0},
20791 {&_pyx_n_s_tanh, __pyx_k_tanh, sizeof(__pyx_k_tanh), 0, 0, 1, 1},
20792 {&_pyx_kp_u_tanh_line_1835, __pyx_k_tanh_line_1835, sizeof(__pyx_k_tanh_line_1835), 0, 1, 0, 0},
20793 {&_pyx_n_s_tau, __pyx_k_tau, sizeof(__pyx_k_tau), 0, 0, 1, 1},
20794 {&_pyx_n_s_test, __pyx_k_test, sizeof(__pyx_k_test), 0, 0, 1, 1},
20795 {&_pyx_n_s_test_2, __pyx_k_test_2, sizeof(__pyx_k_test_2), 0, 0, 1, 1},
20796 {&_pyx_n_s_testmod, __pyx_k_testmod, sizeof(__pyx_k_testmod), 0, 0, 1, 1},
20797 {&_pyx_n_s_threshold, __pyx_k_threshold, sizeof(__pyx_k_threshold), 0, 0, 1, 1},
20798 {&_pyx_n_s_throw, __pyx_k_throw, sizeof(__pyx_k_throw), 0, 0, 1, 1},
20799 {&_pyx_kp_u_to_frame, __pyx_k_to_frame, sizeof(__pyx_k_to_frame), 0, 1, 0, 0},
20800 {&_pyx_n_s_tol, __pyx_k_tol, sizeof(__pyx_k_tol), 0, 0, 1, 1},
20801 {&_pyx_kp_u_using, __pyx_k_using, sizeof(__pyx_k_using), 0, 1, 0, 0},
20802 {&_pyx_kp_u_using_invalid, __pyx_k_using_invalid, sizeof(__pyx_k_using_invalid), 0, 1, 0, 0},
20803 {&_pyx_kp_u_utf_8, __pyx_k_utf_8, sizeof(__pyx_k_utf_8), 0, 1, 0, 0},
20804 {&_pyx_kp_u_value, __pyx_k_value, sizeof(__pyx_k_value), 0, 1, 0, 0},

```

```

20805 {&__pyx_n_s_version, __pyx_k_version, sizeof(__pyx_k_version), 0, 0, 1, 1},
20806 {&__pyx_n_s_xrange, __pyx_k_xrange, sizeof(__pyx_k_xrange), 0, 0, 1, 1},
20807 {0, 0, 0, 0, 0, 0, 0}
20808 };
20809 static CYTHON_SMALL_CODE int __Pyx_InitCachedBuiltins(void) {
20810 __pyx_builtin_IndexError = __Pyx_GetBuiltinName(__pyx_n_s_IndexError); if
20811 (!__pyx_builtin_IndexError) __PYX_ERR(0, 103, __pyx_L1_error)
20812 __pyx_builtin_RuntimeError = __Pyx_GetBuiltinName(__pyx_n_s_RuntimeError); if
20813 (!__pyx_builtin_RuntimeError) __PYX_ERR(0, 105, __pyx_L1_error)
20814 __pyx_builtin_TypeError = __Pyx_GetBuiltinName(__pyx_n_s_TypeError); if (!__pyx_builtin_TypeError)
20815 __PYX_ERR(0, 105, __pyx_L1_error)
20816 __pyx_builtin_ValueError = __Pyx_GetBuiltinName(__pyx_n_s_ValueError); if
20817 (!__pyx_builtin_ValueError) __PYX_ERR(0, 106, __pyx_L1_error)
20818 __pyx_builtin_NotImplemented = __Pyx_GetBuiltinName(__pyx_n_s_NotImplemented); if
20819 (!__pyx_builtin_NotImplemented) __PYX_ERR(0, 159, __pyx_L1_error)
20820 __pyx_builtin_range = __Pyx_GetBuiltinName(__pyx_n_s_range); if (!__pyx_builtin_range) __PYX_ERR(0,
20821 236, __pyx_L1_error)
20822 #if PY_MAJOR_VERSION >= 3
20823 __pyx_builtin_xrange = __Pyx_GetBuiltinName(__pyx_n_s_range); if (!__pyx_builtin_xrange)
20824 __PYX_ERR(0, 1099, __pyx_L1_error)
20825 #else
20826 __pyx_builtin_xrange = __Pyx_GetBuiltinName(__pyx_n_s_xrange); if (!__pyx_builtin_xrange)
20827 __PYX_ERR(0, 1099, __pyx_L1_error)
20828 #endif
20829 return 0;
20830 __pyx_L1_error:;
20831 return -1;
20832 }
20833 static CYTHON_SMALL_CODE int __Pyx_InitCachedConstants(void) {
20834 __Pyx_RefNannyDeclarations
20835 __Pyx_RefNannySetupContext("__Pyx_InitCachedConstants", 0);
20836
20837 /* "(tree fragment)":2
20838 * def __reduce_cython__(self):
20839 * raise TypeError("no default __reduce__ due to non-trivial __cinit__") # ««««««««
20840 * def __setstate_cython__(self, __pyx_state):
20841 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
20842 */
20843 __pyx_tuple__3 = PyTuple_Pack(1, __pyx_kp_s_no_default__reduce__due_to_non); if
20844 (unlikely(!__pyx_tuple__3)) __PYX_ERR(1, 2, __pyx_L1_error)
20845 __Pyx_GOTREF(__pyx_tuple__3);
20846 __Pyx_GIVEREF(__pyx_tuple__3);
20847
20848 /* "(tree fragment)":4
20849 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
20850 * def __setstate_cython__(self, __pyx_state):
20851 * raise TypeError("no default __reduce__ due to non-trivial __cinit__") # ««««««««
20852 */
20853 __pyx_tuple__4 = PyTuple_Pack(1, __pyx_kp_s_no_default__reduce__due_to_non); if
20854 (unlikely(!__pyx_tuple__4)) __PYX_ERR(1, 4, __pyx_L1_error)
20855 __Pyx_GOTREF(__pyx_tuple__4);
20856 __Pyx_GIVEREF(__pyx_tuple__4);
20857
20858 /* "PyClical.pyx":636
20859 * TypeError: Not applicable.
20860 * """
20861 * raise TypeError("Not applicable.") # ««««««««
20862 *
20863 * def __iter__(self):
20864 */
20865 __pyx_tuple__10 = PyTuple_Pack(1, __pyx_kp_u_Not_applicable); if (unlikely(!__pyx_tuple__10))
20866 __PYX_ERR(0, 636, __pyx_L1_error)
20867 __Pyx_GOTREF(__pyx_tuple__10);
20868 __Pyx_GIVEREF(__pyx_tuple__10);
20869
20870 /* "(tree fragment)":2
20871 * def __reduce_cython__(self):
20872 * raise TypeError("no default __reduce__ due to non-trivial __cinit__") # ««««««««
20873 * def __setstate_cython__(self, __pyx_state):
20874 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
20875 */
20876 __pyx_tuple__11 = PyTuple_Pack(1, __pyx_kp_s_no_default__reduce__due_to_non); if
20877 (unlikely(!__pyx_tuple__11)) __PYX_ERR(1, 2, __pyx_L1_error)
20878 __Pyx_GOTREF(__pyx_tuple__11);
20879 __Pyx_GIVEREF(__pyx_tuple__11);
20880
20881 /* "(tree fragment)":4
20882 * raise TypeError("no default __reduce__ due to non-trivial __cinit__")
20883 * def __setstate_cython__(self, __pyx_state):
20884 * raise TypeError("no default __reduce__ due to non-trivial __cinit__") # ««««««««
20885 */
20886 __pyx_tuple__12 = PyTuple_Pack(1, __pyx_kp_s_no_default__reduce__due_to_non); if
20887 (unlikely(!__pyx_tuple__12)) __PYX_ERR(1, 4, __pyx_L1_error)
20888 __Pyx_GOTREF(__pyx_tuple__12);
20889 __Pyx_GIVEREF(__pyx_tuple__12);
20890 }

```

```

20879 /* "PyClical.pyx":406
20880 * return index_set_to_str(self.unwrap()).decode()
20881 *
20882 * def index_set_hidden_doctests(): # ««««««««
20883 * """
20884 * Tests for functions that Doctest cannot see.
20885 */
20886 __pyx_codeobj__13 = (PyObject*)__Pyx_PyCode_New(0, 0, 0, 0, CO_OPTIMIZED|CO_NEWLOCALS,
__pyx_empty_bytes, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_empty_tuple,
__pyx_empty_tuple, __pyx_kp_s_PyClical_pyx, __pyx_n_s_index_set_hidden_doctests, 406,
__pyx_empty_bytes); if (unlikely(!__pyx_codeobj__13)) __PYX_ERR(0, 406, __pyx_L1_error)

20887
20888 /* "PyClical.pyx":1253
20889 * return clifford_to_str(self.unwrap()).decode()
20890 *
20891 * def clifford_hidden_doctests(): # ««««««««
20892 * """
20893 * Tests for functions that Doctest cannot see.
20894 */
20895 __pyx_codeobj__14 = (PyObject*)__Pyx_PyCode_New(0, 0, 0, 0, CO_OPTIMIZED|CO_NEWLOCALS,
__pyx_empty_bytes, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_empty_tuple,
__pyx_empty_tuple, __pyx_kp_s_PyClical_pyx, __pyx_n_s_clifford_hidden_doctests, 1253,
__pyx_empty_bytes); if (unlikely(!__pyx_codeobj__14)) __PYX_ERR(0, 1253, __pyx_L1_error)

20896
20897 /* "PyClical.pyx":1907
20898 * scalar_epsilon = epsilon
20899 *
20900 * pi = atan(clifford(1.0)) * 4.0 # ««««««««
20901 * tau = atan(clifford(1.0)) * 8.0
20902 *
20903 */
20904 __pyx_tuple__15 = PyTuple_Pack(1, __pyx_float_1_0); if (unlikely(!__pyx_tuple__15)) __PYX_ERR(0,
1907, __pyx_L1_error)
20905 __Pyx_GOTREF(__pyx_tuple__15);
20906 __Pyx_GIVEREF(__pyx_tuple__15);
20907
20908 /* "PyClical.pyx":1936
20909 * """
20910 *
20911 * def e(obj): # ««««««««
20912 * """
20913 * Abbreviation for clifford(index_set(obj)).
20914 */
20915 __pyx_tuple__16 = PyTuple_Pack(1, __pyx_n_s_obj); if (unlikely(!__pyx_tuple__16)) __PYX_ERR(0, 1936,
__pyx_L1_error)
20916 __Pyx_GOTREF(__pyx_tuple__16);
20917 __Pyx_GIVEREF(__pyx_tuple__16);
20918 __pyx_codeobj__17 = (PyObject*)__Pyx_PyCode_New(1, 0, 1, 0, CO_OPTIMIZED|CO_NEWLOCALS,
__pyx_empty_bytes, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_tuple__16, __pyx_empty_tuple,
__pyx_empty_tuple, __pyx_kp_s_PyClical_pyx, __pyx_n_s_e, 1936, __pyx_empty_bytes); if
(unlikely(!__pyx_codeobj__17)) __PYX_ERR(0, 1936, __pyx_L1_error)

20919
20920 /* "PyClical.pyx":1949
20921 * return clifford(index_set(obj))
20922 *
20923 * def istpq(p, q): # ««««««««
20924 * """
20925 * Abbreviation for index_set({-q,...p}).
20926 */
20927 __pyx_tuple__18 = PyTuple_Pack(2, __pyx_n_s_p, __pyx_n_s_q); if (unlikely(!__pyx_tuple__18))
__PYX_ERR(0, 1949, __pyx_L1_error)
20928 __Pyx_GOTREF(__pyx_tuple__18);
20929 __Pyx_GIVEREF(__pyx_tuple__18);
20930 __pyx_codeobj__19 = (PyObject*)__Pyx_PyCode_New(2, 0, 2, 0, CO_OPTIMIZED|CO_NEWLOCALS,
__pyx_empty_bytes, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_tuple__18, __pyx_empty_tuple,
__pyx_empty_tuple, __pyx_kp_s_PyClical_pyx, __pyx_n_s_istpq, 1949, __pyx_empty_bytes); if
(unlikely(!__pyx_codeobj__19)) __PYX_ERR(0, 1949, __pyx_L1_error)

20931
20932 /* "PyClical.pyx":1958
20933 * return index_set(set(range(-q,p+1)))
20934 *
20935 * ninf3 = e(4) + e(-1) # Null infinity point in 3D Conformal Geometric Algebra [DL]. #
20936 * ««««««««
20937 * nbar3 = e(4) - e(-1) # Null bar point in 3D Conformal Geometric Algebra [DL].
20938 *
20939 */
20939 __pyx_tuple__20 = PyTuple_Pack(1, __pyx_int_4); if (unlikely(!__pyx_tuple__20)) __PYX_ERR(0, 1958,
__pyx_L1_error)
20940 __Pyx_GOTREF(__pyx_tuple__20);
20941 __Pyx_GIVEREF(__pyx_tuple__20);
20942 __pyx_tuple__21 = PyTuple_Pack(1, __pyx_int_neg_1); if (unlikely(!__pyx_tuple__21)) __PYX_ERR(0,
1958, __pyx_L1_error)
20943 __Pyx_GOTREF(__pyx_tuple__21);
20944 __Pyx_GIVEREF(__pyx_tuple__21);
20945
20946 /* "PyClical.pyx":1962
20947 *

```

```

20948 * # Doctest interface.
20949 * def _test(): # ««««««
20950 * import PyClical, doctest
20951 * return doctest.testmod(PyClical)
20952 */
20953 __pyx_tuple__22 = PyTuple_Pack(2, __pyx_n_s_PyClical, __pyx_n_s_doctest); if
(unlikely(!__pyx_tuple__22)) __PYX_ERR(0, 1962, __pyx_L1_error)
20954 __Pyx_GOTREF(__pyx_tuple__22);
20955 __Pyx_GIVEREF(__pyx_tuple__22);
20956 __pyx_codeobj__23 = (PyObject*)__Pyx_PyCode_New(0, 0, 2, 0, CO_OPTIMIZED|CO_NEWLOCALS,
__pyx_empty_bytes, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_tuple__22, __pyx_empty_tuple,
__pyx_empty_tuple, __pyx_kp_s_PyClical_pyx, __pyx_n_s_test, 1962, __pyx_empty_bytes); if
(unlikely(!__pyx_codeobj__23)) __PYX_ERR(0, 1962, __pyx_L1_error)
20957 __Pyx_RefNannyFinishContext();
20958 return 0;
20959 __pyx_L1_error:;
20960 __Pyx_RefNannyFinishContext();
20961 return -1;
20962 }
20963
20964 static CYTHON_SMALL_CODE int __Pyx_InitGlobals(void) {
20965 if (__Pyx_InitStrings(__pyx_string_tab) < 0) __PYX_ERR(0, 1, __pyx_L1_error);
20966 __pyx_float_0_0 = PyFloat_FromDouble(0.0); if (unlikely(!__pyx_float_0_0)) __PYX_ERR(0, 1,
__pyx_L1_error)
20967 __pyx_float_1_0 = PyFloat_FromDouble(1.0); if (unlikely(!__pyx_float_1_0)) __PYX_ERR(0, 1,
__pyx_L1_error)
20968 __pyx_float_4_0 = PyFloat_FromDouble(4.0); if (unlikely(!__pyx_float_4_0)) __PYX_ERR(0, 1,
__pyx_L1_error)
20969 __pyx_float_8_0 = PyFloat_FromDouble(8.0); if (unlikely(!__pyx_float_8_0)) __PYX_ERR(0, 1,
__pyx_L1_error)
20970 __pyx_int_0 = PyInt_FromLong(0); if (unlikely(!__pyx_int_0)) __PYX_ERR(0, 1, __pyx_L1_error)
20971 __pyx_int_1 = PyInt_FromLong(1); if (unlikely(!__pyx_int_1)) __PYX_ERR(0, 1, __pyx_L1_error)
20972 __pyx_int_4 = PyInt_FromLong(4); if (unlikely(!__pyx_int_4)) __PYX_ERR(0, 1, __pyx_L1_error)
20973 __pyx_int_neg_1 = PyInt_FromLong(-1); if (unlikely(!__pyx_int_neg_1)) __PYX_ERR(0, 1,
__pyx_L1_error)
20974 return 0;
20975 __pyx_L1_error:;
20976 return -1;
20977 }
20978
20979 static CYTHON_SMALL_CODE int __Pyx_modinit_global_init_code(void); /*proto*/
20980 static CYTHON_SMALL_CODE int __Pyx_modinit_variable_export_code(void); /*proto*/
20981 static CYTHON_SMALL_CODE int __Pyx_modinit_function_export_code(void); /*proto*/
20982 static CYTHON_SMALL_CODE int __Pyx_modinit_type_init_code(void); /*proto*/
20983 static CYTHON_SMALL_CODE int __Pyx_modinit_type_import_code(void); /*proto*/
20984 static CYTHON_SMALL_CODE int __Pyx_modinit_variable_import_code(void); /*proto*/
20985 static CYTHON_SMALL_CODE int __Pyx_modinit_function_import_code(void); /*proto*/
20986
20987 static int __Pyx_modinit_global_init_code(void) {
20988 __Pyx_RefNannyDeclarations
20989 __Pyx_RefNannySetupContext("__Pyx_modinit_global_init_code", 0);
20990 /*--- Global init code ---*/
20991 __Pyx_RefNannyFinishContext();
20992 return 0;
20993 }
20994
20995 static int __Pyx_modinit_variable_export_code(void) {
20996 __Pyx_RefNannyDeclarations
20997 __Pyx_RefNannySetupContext("__Pyx_modinit_variable_export_code", 0);
20998 /*--- Variable export code ---*/
20999 __Pyx_RefNannyFinishContext();
21000 return 0;
21001 }
21002
21003 static int __Pyx_modinit_function_export_code(void) {
21004 __Pyx_RefNannyDeclarations
21005 __Pyx_RefNannySetupContext("__Pyx_modinit_function_export_code", 0);
21006 /*--- Function export code ---*/
21007 __Pyx_RefNannyFinishContext();
21008 return 0;
21009 }
21010
21011 static int __Pyx_modinit_type_init_code(void) {
21012 __Pyx_RefNannyDeclarations
21013 int __pyx_lineno = 0;
21014 const char *__pyx_filename = NULL;
21015 int __pyx_clineno = 0;
21016 __Pyx_RefNannySetupContext("__Pyx_modinit_type_init_code", 0);
21017 /*--- Type init code ---*/
21018 __pyx_vtabptr_8PyClical_index_set = &__pyx_vtable_8PyClical_index_set;
21019 __pyx_vtable_8PyClical_index_set.wrap = (PyObject *) (struct __pyx_obj_8PyClical_index_set *,
IndexSet) __pyx_f_8PyClical_9index_set_wrap;
21020 __pyx_vtable_8PyClical_index_set.unwrap = (IndexSet *) (struct __pyx_obj_8PyClical_index_set
*) __pyx_f_8PyClical_9index_set_unwrap;
21021 __pyx_vtable_8PyClical_index_set.copy = (PyObject *) (struct __pyx_obj_8PyClical_index_set *, int
__pyx_skip_dispatch) __pyx_f_8PyClical_9index_set_copy;
21022 if (PyType_Ready(&__pyx_type_8PyClical_index_set) < 0) __PYX_ERR(0, 46, __pyx_L1_error)

```

```

21023 #if PY_VERSION_HEX < 0x030800B1
21024 __pyx_type_8PyClical_index_set.tp_print = 0;
21025 #endif
21026 if ((CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP) &&
likely(!__pyx_type_8PyClical_index_set.tp_dictoffset && __pyx_type_8PyClical_index_set.tp_getattro ==
PyObject_GenericGetAttr)) {
21027 __pyx_type_8PyClical_index_set.tp_getattro = __Pyx_PyObject_GenericGetAttr;
21028 }
21029 #if CYTHON_COMPILING_IN_CPYTHON
21030 {
21031 PyObject *wrapper = PyObject_GetAttrString(PyObject *)&__pyx_type_8PyClical_index_set,
"__setitem__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21032 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21033 __pyx_wrapperbase_8PyClical_9index_set_8__setitem__ = *((PyWrapperDescrObject
*)wrapper)->d_base;
21034 __pyx_wrapperbase_8PyClical_9index_set_8__setitem__.doc =
__pyx_doc_8PyClical_9index_set_8__setitem__;
21035 (PyWrapperDescrObject *)wrapper->d_base =
&__pyx_wrapperbase_8PyClical_9index_set_8__setitem__;
21036 }
21037 }
21038 #endif
21039 #if CYTHON_COMPILING_IN_CPYTHON
21040 {
21041 PyObject *wrapper = PyObject_GetAttrString(PyObject *)&__pyx_type_8PyClical_index_set,
"__getitem__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21042 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21043 __pyx_wrapperbase_8PyClical_9index_set_10__getitem__ = *((PyWrapperDescrObject
*)wrapper)->d_base;
21044 __pyx_wrapperbase_8PyClical_9index_set_10__getitem__.doc =
__pyx_doc_8PyClical_9index_set_10__getitem__;
21045 (PyWrapperDescrObject *)wrapper->d_base =
&__pyx_wrapperbase_8PyClical_9index_set_10__getitem__;
21046 }
21047 }
21048 #endif
21049 #if CYTHON_COMPILING_IN_CPYTHON
21050 {
21051 PyObject *wrapper = PyObject_GetAttrString(PyObject *)&__pyx_type_8PyClical_index_set,
"__contains__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21052 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21053 __pyx_wrapperbase_8PyClical_9index_set_12__contains__ = *((PyWrapperDescrObject
*)wrapper)->d_base;
21054 __pyx_wrapperbase_8PyClical_9index_set_12__contains__.doc =
__pyx_doc_8PyClical_9index_set_12__contains__;
21055 (PyWrapperDescrObject *)wrapper->d_base =
&__pyx_wrapperbase_8PyClical_9index_set_12__contains__;
21056 }
21057 }
21058 #endif
21059 #if CYTHON_COMPILING_IN_CPYTHON
21060 {
21061 PyObject *wrapper = PyObject_GetAttrString(PyObject *)&__pyx_type_8PyClical_index_set,
"__iter__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21062 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21063 __pyx_wrapperbase_8PyClical_9index_set_14__iter__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21064 __pyx_wrapperbase_8PyClical_9index_set_14__iter__.doc =
__pyx_doc_8PyClical_9index_set_14__iter__;
21065 (PyWrapperDescrObject *)wrapper->d_base = &__pyx_wrapperbase_8PyClical_9index_set_14__iter__;
21066 }
21067 }
21068 #endif
21069 #if CYTHON_COMPILING_IN_CPYTHON
21070 {
21071 PyObject *wrapper = PyObject_GetAttrString(PyObject *)&__pyx_type_8PyClical_index_set,
"__invert__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21072 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21073 __pyx_wrapperbase_8PyClical_9index_set_17__invert__ = *((PyWrapperDescrObject
*)wrapper)->d_base;
21074 __pyx_wrapperbase_8PyClical_9index_set_17__invert__.doc =
__pyx_doc_8PyClical_9index_set_17__invert__;
21075 (PyWrapperDescrObject *)wrapper->d_base =
&__pyx_wrapperbase_8PyClical_9index_set_17__invert__;
21076 }
21077 }
21078 #endif
21079 #if CYTHON_COMPILING_IN_CPYTHON
21080 {
21081 PyObject *wrapper = PyObject_GetAttrString(PyObject *)&__pyx_type_8PyClical_index_set,
"__xor__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21082 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21083 __pyx_wrapperbase_8PyClical_9index_set_19__xor__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21084 __pyx_wrapperbase_8PyClical_9index_set_19__xor__.doc = __pyx_doc_8PyClical_9index_set_19__xor__;
21085 (PyWrapperDescrObject *)wrapper->d_base = &__pyx_wrapperbase_8PyClical_9index_set_19__xor__;
21086 }
21087 }
21088 #endif

```

```

21089 #if CYTHON_COMPILING_IN_CPYTHON
21090 {
21091 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set,
21092 "__ixor__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21093 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21094 __pyx_wrapperbase_8PyClical_9index_set_21__ixor__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21095 __pyx_wrapperbase_8PyClical_9index_set_21__ixor__.doc =
21096 __pyx_doc_8PyClical_9index_set_21__ixor__;
21097 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_21__ixor__;
21098 }
21099 }
21100 #endif
21101 #if CYTHON_COMPILING_IN_CPYTHON
21102 {
21103 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set,
21104 "__and__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21105 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21106 __pyx_wrapperbase_8PyClical_9index_set_23__and__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21107 __pyx_wrapperbase_8PyClical_9index_set_23__and__.doc = __pyx_doc_8PyClical_9index_set_23__and__;
21108 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_23__and__;
21109 }
21110 }
21111 #endif
21112 #if CYTHON_COMPILING_IN_CPYTHON
21113 {
21114 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set,
21115 "__iand__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21116 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21117 __pyx_wrapperbase_8PyClical_9index_set_25__iand__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21118 __pyx_wrapperbase_8PyClical_9index_set_25__iand__.doc =
21119 __pyx_doc_8PyClical_9index_set_25__iand__;
21120 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_25__iand__;
21121 }
21122 }
21123 #endif
21124 #if CYTHON_COMPILING_IN_CPYTHON
21125 {
21126 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set, "__ior__");
21127 if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21128 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21129 __pyx_wrapperbase_8PyClical_9index_set_27__ior__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21130 __pyx_wrapperbase_8PyClical_9index_set_27__ior__.doc = __pyx_doc_8PyClical_9index_set_27__ior__;
21131 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_27__ior__;
21132 }
21133 }
21134 #endif
21135 #if CYTHON_COMPILING_IN_CPYTHON
21136 {
21137 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set,
21138 "__ior__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21139 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21140 __pyx_wrapperbase_8PyClical_9index_set_29__ior__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21141 __pyx_wrapperbase_8PyClical_9index_set_29__ior__.doc = __pyx_doc_8PyClical_9index_set_29__ior__;
21142 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_29__ior__;
21143 }
21144 }
21145 #endif
21146 #if CYTHON_COMPILING_IN_CPYTHON
21147 {
21148 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set,
21149 "__repr__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21150 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21151 __pyx_wrapperbase_8PyClical_9index_set_47__repr__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21152 __pyx_wrapperbase_8PyClical_9index_set_47__repr__.doc =
21153 __pyx_doc_8PyClical_9index_set_47__repr__;
21154 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_47__repr__;
21155 }
21156 }
21157 #endif
21158 #if CYTHON_COMPILING_IN_CPYTHON
21159 {
21160 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_index_set,
21161 "__str__"); if (unlikely(!wrapper)) __PYX_ERR(0, 46, __pyx_L1_error)
21162 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21163 __pyx_wrapperbase_8PyClical_9index_set_49__str__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21164 __pyx_wrapperbase_8PyClical_9index_set_49__str__.doc = __pyx_doc_8PyClical_9index_set_49__str__;
21165 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_9index_set_49__str__;
21166 }
21167 }
21168 #endif
21169 if (__Pyx_SetVtable(__pyx_type_8PyClical_index_set.tp_dict, __pyx_vtabptr_8PyClical_index_set) < 0)
21170 __PYX_ERR(0, 46, __pyx_L1_error)
21171 if (PyObject_SetAttr(__pyx_m, __pyx_n_s_index_set, (PyObject *)&__pyx_type_8PyClical_index_set) < 0)
21172 __PYX_ERR(0, 46, __pyx_L1_error)
21173 if (__Pyx_setup_reduce((PyObject *)&__pyx_type_8PyClical_index_set) < 0) __PYX_ERR(0, 46,
21174 __pyx_L1_error)
21175 __pyx_ptype_8PyClical_index_set = &__pyx_type_8PyClical_index_set;

```

```

21163 __pyx_vtabptr_8PyClical_clifford = &__pyx_vtable_8PyClical_clifford;
21164 __pyx_vtable_8PyClical_clifford.wrap = (PyObject (*)(struct __pyx_obj_8PyClical_clifford *,
Clifford))__pyx_f_8PyClical_8clifford_wrap;
21165 __pyx_vtable_8PyClical_clifford.unwrap = (Clifford (*)(struct __pyx_obj_8PyClical_clifford
*))__pyx_f_8PyClical_8clifford_unwrap;
21166 __pyx_vtable_8PyClical_clifford.copy = (PyObject (*)(struct __pyx_obj_8PyClical_clifford *, int
__pyx_skip_dispatch))__pyx_f_8PyClical_8clifford_copy;
21167 if (PyType_Ready(&__pyx_type_8PyClical_clifford) < 0) __PYX_ERR(0, 537, __pyx_L1_error)
21168 #if PY_VERSION_HEX < 0x030800B1
21169 __pyx_type_8PyClical_clifford.tp_print = 0;
21170 #endif
21171 if ((CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP) &&
likely(!__pyx_type_8PyClical_clifford.tp_dictoffset && __pyx_type_8PyClical_clifford.tp_getattro ==
PyObject_GenericGetAttr)) {
21172 __pyx_type_8PyClical_clifford.tp_getattro = __Pyx_PyObject_GenericGetAttr;
21173 }
21174 #if CYTHON_COMPILING_IN_CPYTHON
21175 {
21176 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
"__contains__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21177 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21178 __pyx_wrapperbase_8PyClical_8clifford_6__contains__ = *((PyWrapperDescrObject
*)wrapper)->d_base;
21179 __pyx_wrapperbase_8PyClical_8clifford_6__contains__.doc =
__pyx_doc_8PyClical_8clifford_6__contains__;
21180 ((PyWrapperDescrObject *)wrapper)->d_base =
&__pyx_wrapperbase_8PyClical_8clifford_6__contains__;
21181 }
21182 }
21183 #endif
21184 #if CYTHON_COMPILING_IN_CPYTHON
21185 {
21186 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
"__iter__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21187 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21188 __pyx_wrapperbase_8PyClical_8clifford_8__iter__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21189 __pyx_wrapperbase_8PyClical_8clifford_8__iter__.doc = __pyx_doc_8PyClical_8clifford_8__iter__;
21190 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_8__iter__;
21191 }
21192 }
21193 #endif
21194 #if CYTHON_COMPILING_IN_CPYTHON
21195 {
21196 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
"__getitem__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21197 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21198 __pyx_wrapperbase_8PyClical_8clifford_14__getitem__ = *((PyWrapperDescrObject
*)wrapper)->d_base;
21199 __pyx_wrapperbase_8PyClical_8clifford_14__getitem__.doc =
__pyx_doc_8PyClical_8clifford_14__getitem__;
21200 ((PyWrapperDescrObject *)wrapper)->d_base =
&__pyx_wrapperbase_8PyClical_8clifford_14__getitem__;
21201 }
21202 }
21203 #endif
21204 #if CYTHON_COMPILING_IN_CPYTHON
21205 {
21206 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__neg__");
if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21207 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21208 __pyx_wrapperbase_8PyClical_8clifford_16__neg__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21209 __pyx_wrapperbase_8PyClical_8clifford_16__neg__.doc = __pyx_doc_8PyClical_8clifford_16__neg__;
21210 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_16__neg__;
21211 }
21212 }
21213 #endif
21214 #if CYTHON_COMPILING_IN_CPYTHON
21215 {
21216 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__pos__");
if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21217 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21218 __pyx_wrapperbase_8PyClical_8clifford_18__pos__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21219 __pyx_wrapperbase_8PyClical_8clifford_18__pos__.doc = __pyx_doc_8PyClical_8clifford_18__pos__;
21220 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_18__pos__;
21221 }
21222 }
21223 #endif
21224 #if CYTHON_COMPILING_IN_CPYTHON
21225 {
21226 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__add__");
if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21227 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21228 __pyx_wrapperbase_8PyClical_8clifford_20__add__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21229 __pyx_wrapperbase_8PyClical_8clifford_20__add__.doc = __pyx_doc_8PyClical_8clifford_20__add__;
21230 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_20__add__;
21231 }
21232 }

```



```

21233 #endif
21234 #if CYTHON_COMPILING_IN_CPYTHON
21235 {
21236 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford,
21237 "___iadd___"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21238 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21239 __pyx_wrapperbase_8PyClical_8clifford_22___iadd___ = *((PyWrapperDescrObject *)wrapper)->d_base;
21240 __pyx_wrapperbase_8PyClical_8clifford_22___iadd___ .doc = __pyx_doc_8PyClical_8clifford_22___iadd___;
21241 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_22___iadd___;
21242 }
21243 #endif
21244 #if CYTHON_COMPILING_IN_CPYTHON
21245 {
21246 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford, "___sub___");
21247 if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21248 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21249 __pyx_wrapperbase_8PyClical_8clifford_24___sub___ = *((PyWrapperDescrObject *)wrapper)->d_base;
21250 __pyx_wrapperbase_8PyClical_8clifford_24___sub___ .doc = __pyx_doc_8PyClical_8clifford_24___sub___;
21251 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_24___sub___;
21252 }
21253 #endif
21254 #if CYTHON_COMPILING_IN_CPYTHON
21255 {
21256 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford,
21257 "___isub___"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21258 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21259 __pyx_wrapperbase_8PyClical_8clifford_26___isub___ = *((PyWrapperDescrObject *)wrapper)->d_base;
21260 __pyx_wrapperbase_8PyClical_8clifford_26___isub___ .doc = __pyx_doc_8PyClical_8clifford_26___isub___;
21261 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_26___isub___;
21262 }
21263 #endif
21264 #if CYTHON_COMPILING_IN_CPYTHON
21265 {
21266 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford, "___mul___");
21267 if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21268 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21269 __pyx_wrapperbase_8PyClical_8clifford_28___mul___ = *((PyWrapperDescrObject *)wrapper)->d_base;
21270 __pyx_wrapperbase_8PyClical_8clifford_28___mul___ .doc = __pyx_doc_8PyClical_8clifford_28___mul___;
21271 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_28___mul___;
21272 }
21273 #endif
21274 #if CYTHON_COMPILING_IN_CPYTHON
21275 {
21276 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford,
21277 "___imul___"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21278 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21279 __pyx_wrapperbase_8PyClical_8clifford_30___imul___ = *((PyWrapperDescrObject *)wrapper)->d_base;
21280 __pyx_wrapperbase_8PyClical_8clifford_30___imul___ .doc = __pyx_doc_8PyClical_8clifford_30___imul___;
21281 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_30___imul___;
21282 }
21283 #endif
21284 #if CYTHON_COMPILING_IN_CPYTHON
21285 {
21286 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford, "___mod___");
21287 if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21288 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21289 __pyx_wrapperbase_8PyClical_8clifford_32___mod___ = *((PyWrapperDescrObject *)wrapper)->d_base;
21290 __pyx_wrapperbase_8PyClical_8clifford_32___mod___ .doc = __pyx_doc_8PyClical_8clifford_32___mod___;
21291 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_32___mod___;
21292 }
21293 #endif
21294 #if CYTHON_COMPILING_IN_CPYTHON
21295 {
21296 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford,
21297 "___imod___"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21298 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21299 __pyx_wrapperbase_8PyClical_8clifford_34___imod___ = *((PyWrapperDescrObject *)wrapper)->d_base;
21300 __pyx_wrapperbase_8PyClical_8clifford_34___imod___ .doc = __pyx_doc_8PyClical_8clifford_34___imod___;
21301 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_34___imod___;
21302 }
21303 #endif
21304 #if CYTHON_COMPILING_IN_CPYTHON
21305 {
21306 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford, "___and___");
21307 if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21308 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21309 __pyx_wrapperbase_8PyClical_8clifford_36___and___ = *((PyWrapperDescrObject *)wrapper)->d_base;
21310 __pyx_wrapperbase_8PyClical_8clifford_36___and___ .doc = __pyx_doc_8PyClical_8clifford_36___and___;
21311 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_36___and___;

```



```

21312 }
21313 #endif
21314 #if CYTHON_COMPILING_IN_CPYTHON
21315 {
21316 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford,
21317 "iand"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21318 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21319 __pyx_wrapperbase_8PyClical_8clifford_38_iand__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21320 __pyx_wrapperbase_8PyClical_8clifford_38_iand__.doc = __pyx_doc_8PyClical_8clifford_38_iand__;
21321 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_38_iand__;
21322 }
21323 }
21324 #endif
21325 #if CYTHON_COMPILING_IN_CPYTHON
21326 {
21327 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford, "__xor__");
21328 if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21329 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21330 __pyx_wrapperbase_8PyClical_8clifford_40_xor__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21331 __pyx_wrapperbase_8PyClical_8clifford_40_xor__.doc = __pyx_doc_8PyClical_8clifford_40_xor__;
21332 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_40_xor__;
21333 }
21334 }
21335 #endif
21336 #if CYTHON_COMPILING_IN_CPYTHON
21337 {
21338 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford,
21339 "ixor"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21340 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21341 __pyx_wrapperbase_8PyClical_8clifford_42_ixor__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21342 __pyx_wrapperbase_8PyClical_8clifford_42_ixor__.doc = __pyx_doc_8PyClical_8clifford_42_ixor__;
21343 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_42_ixor__;
21344 }
21345 }
21346 #endif
21347 #if CYTHON_COMPILING_IN_CPYTHON
21348 {
21349 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford,
21350 "truediv"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21351 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21352 __pyx_wrapperbase_8PyClical_8clifford_44_truediv__ = *((PyWrapperDescrObject
21353 *)wrapper)->d_base;
21354 __pyx_wrapperbase_8PyClical_8clifford_44_truediv__.doc =
21355 __pyx_doc_8PyClical_8clifford_44_truediv__;
21356 ((PyWrapperDescrObject *)wrapper)->d_base =
21357 &__pyx_wrapperbase_8PyClical_8clifford_44_truediv__;
21358 }
21359 }
21360 #endif
21361 #if PY_MAJOR_VERSION < 3 || (CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX < 0x03050000)
21362 #if CYTHON_COMPILING_IN_CPYTHON
21363 {
21364 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford,
21365 "idiv"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21366 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21367 __pyx_wrapperbase_8PyClical_8clifford_46_idiv__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21368 __pyx_wrapperbase_8PyClical_8clifford_46_idiv__.doc = __pyx_doc_8PyClical_8clifford_46_idiv__;
21369 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_46_idiv__;
21370 }
21371 }
21372 #endif
21373 #endif
21374 #if CYTHON_COMPILING_IN_CPYTHON
21375 {
21376 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford, "__or__");
21377 if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21378 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21379 __pyx_wrapperbase_8PyClical_8clifford_50_or__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21380 __pyx_wrapperbase_8PyClical_8clifford_50_or__.doc = __pyx_doc_8PyClical_8clifford_50_or__;
21381 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_50_or__;
21382 }
21383 }
21384 #endif
21385 #if CYTHON_COMPILING_IN_CPYTHON
21386 {
21387 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford, "__ior__");
21388 if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_L1_error)
21389 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21390 __pyx_wrapperbase_8PyClical_8clifford_52_ior__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21391 __pyx_wrapperbase_8PyClical_8clifford_52_ior__.doc = __pyx_doc_8PyClical_8clifford_52_ior__;
21392 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_52_ior__;
21393 }
21394 }
21395 #endif
21396 #if CYTHON_COMPILING_IN_CPYTHON
21397 {
21398 PyObject *wrapper = PyObject_GetAttrString((PyObject *) &__pyx_type_8PyClical_clifford, "__pow__");

```

```

 if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_l1_error)
21389 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21390 __pyx_wrapperbase_8PyClical_8clifford_54_pow__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21391 __pyx_wrapperbase_8PyClical_8clifford_54_pow__.doc = __pyx_doc_8PyClical_8clifford_54_pow__;
21392 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_54_pow__;
21393 }
21394 }
21395 #endif
21396 #if CYTHON_COMPILING_IN_CPYTHON
21397 {
21398 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
21399 "__call__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_l1_error)
21399 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21400 __pyx_wrapperbase_8PyClical_8clifford_60_call__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21401 __pyx_wrapperbase_8PyClical_8clifford_60_call__.doc = __pyx_doc_8PyClical_8clifford_60_call__;
21402 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_60_call__;
21403 }
21404 }
21405 #endif
21406 #if CYTHON_COMPILING_IN_CPYTHON
21407 {
21408 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford,
21409 "__repr__"); if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_l1_error)
21409 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21410 __pyx_wrapperbase_8PyClical_8clifford_94_repr__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21411 __pyx_wrapperbase_8PyClical_8clifford_94_repr__.doc = __pyx_doc_8PyClical_8clifford_94_repr__;
21412 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_94_repr__;
21413 }
21414 }
21415 #endif
21416 #if CYTHON_COMPILING_IN_CPYTHON
21417 {
21418 PyObject *wrapper = PyObject_GetAttrString((PyObject *)&__pyx_type_8PyClical_clifford, "__str__");
21419 if (unlikely(!wrapper)) __PYX_ERR(0, 537, __pyx_l1_error)
21420 if (Py_TYPE(wrapper) == &PyWrapperDescr_Type) {
21421 __pyx_wrapperbase_8PyClical_8clifford_96_str__ = *((PyWrapperDescrObject *)wrapper)->d_base;
21422 __pyx_wrapperbase_8PyClical_8clifford_96_str__.doc = __pyx_doc_8PyClical_8clifford_96_str__;
21423 ((PyWrapperDescrObject *)wrapper)->d_base = &__pyx_wrapperbase_8PyClical_8clifford_96_str__;
21424 }
21425 #endif
21426 if (__Pyx_SetVtable(__pyx_type_8PyClical_clifford.tp_dict, __pyx_vtabptr_8PyClical_clifford) < 0)
21427 __PYX_ERR(0, 537, __pyx_l1_error)
21427 if (PyObject_SetAttr(__pyx_m, __pyx_n_s_clifford, (PyObject *)&__pyx_type_8PyClical_clifford) < 0)
21428 __PYX_ERR(0, 537, __pyx_l1_error)
21428 if (__Pyx_setup_reduce((PyObject *)&__pyx_type_8PyClical_clifford) < 0) __PYX_ERR(0, 537,
21429 __pyx_l1_error)
21429 __pyx_ptype_8PyClical_clifford = &__pyx_type_8PyClical_clifford;
21430 if (PyType_Ready(&__pyx_type_8PyClical__pyx_scope_struct__iter__) < 0) __PYX_ERR(0, 229,
21431 __pyx_l1_error)
21431 #if PY_VERSION_HEX < 0x030800B1
21432 __pyx_type_8PyClical__pyx_scope_struct__iter__.tp_print = 0;
21433 #endif
21434 if ((CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP) &&
21435 likely(!__pyx_type_8PyClical__pyx_scope_struct__iter__.tp_dictoffset &&
21436 __pyx_type_8PyClical__pyx_scope_struct__iter__.tp_getattro == PyObject_GenericGetAttr)) {
21435 __pyx_type_8PyClical__pyx_scope_struct__iter__.tp_getattro =
21436 __Pyx_PyObject_GenericGetAttrNoDict;
21437 }
21437 __pyx_ptype_8PyClical__pyx_scope_struct__iter__ =
21438 &__pyx_type_8PyClical__pyx_scope_struct__iter__;
21438 __Pyx_RefNannyFinishContext();
21439 return 0;
21440 __pyx_l1_error:;
21441 __Pyx_RefNannyFinishContext();
21442 return -1;
21443 }
21444
21445 static int __Pyx_modinit_type_import_code(void) {
21446 __Pyx_RefNannyDeclarations
21447 __Pyx_RefNannySetupContext("__Pyx_modinit_type_import_code", 0);
21448 /*--- Type import code ---*/
21449 __Pyx_RefNannyFinishContext();
21450 return 0;
21451 }
21452
21453 static int __Pyx_modinit_variable_import_code(void) {
21454 __Pyx_RefNannyDeclarations
21455 __Pyx_RefNannySetupContext("__Pyx_modinit_variable_import_code", 0);
21456 /*--- Variable import code ---*/
21457 __Pyx_RefNannyFinishContext();
21458 return 0;
21459 }
21460
21461 static int __Pyx_modinit_function_import_code(void) {
21462 __Pyx_RefNannyDeclarations
21463 __Pyx_RefNannySetupContext("__Pyx_modinit_function_import_code", 0);

```

```

21464 /*--- Function import code ---*/
21465 __Pyx_RefNannyFinishContext();
21466 return 0;
21467 }
21468
21469
21470 #ifndef CYTHON_NO_PYINIT_EXPORT
21471 #define __Pyx_PyMODINIT_FUNC PyMODINIT_FUNC
21472 #elif PY_MAJOR_VERSION < 3
21473 #ifdef __cplusplus
21474 #define __Pyx_PyMODINIT_FUNC extern "C" void
21475 #else
21476 #define __Pyx_PyMODINIT_FUNC void
21477 #endif
21478 #else
21479 #ifdef __cplusplus
21480 #define __Pyx_PyMODINIT_FUNC extern "C" PyObject *
21481 #else
21482 #define __Pyx_PyMODINIT_FUNC PyObject *
21483 #endif
21484 #endif
21485
21486
21487 #if PY_MAJOR_VERSION < 3
21488 __Pyx_PyMODINIT_FUNC initPyClical(void) CYTHON_SMALL_CODE; /*proto*/
21489 __Pyx_PyMODINIT_FUNC initPyClical(void)
21490 #else
21491 __Pyx_PyMODINIT_FUNC PyInit_PyClical(void) CYTHON_SMALL_CODE; /*proto*/
21492 __Pyx_PyMODINIT_FUNC PyInit_PyClical(void)
21493 #if CYTHON_PEP489_MULTI_PHASE_INIT
21494 {
21495 return PyModuleDef_Init(&__pyx_moduledef);
21496 }
21497 static CYTHON_SMALL_CODE int __Pyx_check_single_interpreter(void) {
21498 #if PY_VERSION_HEX >= 0x030700A1
21499 static PY_INT64_T main_interpreter_id = -1;
21500 PY_INT64_T current_id = PyInterpreterState_GetID(PyThreadState_Get()->interp);
21501 if (main_interpreter_id == -1) {
21502 main_interpreter_id = current_id;
21503 return (unlikely(current_id == -1)) ? -1 : 0;
21504 } else if (unlikely(main_interpreter_id != current_id))
21505 #else
21506 static PyInterpreterState *main_interpreter = NULL;
21507 PyInterpreterState *current_interpreter = PyThreadState_Get()->interp;
21508 if (!main_interpreter) {
21509 main_interpreter = current_interpreter;
21510 } else if (unlikely(main_interpreter != current_interpreter))
21511 #endif
21512 {
21513 PyErr_SetString(
21514 PyExc_ImportError,
21515 "Interpreter change detected - this module can only be loaded into one interpreter per
process.");
21516 return -1;
21517 }
21518 return 0;
21519 }
21520 static CYTHON_SMALL_CODE int __Pyx_copy_spec_to_module(PyObject *spec, PyObject *moddict, const char*
from_name, const char* to_name, int allow_none) {
21521 PyObject *value = PyObject_GetAttrString(spec, from_name);
21522 int result = 0;
21523 if (likely(value)) {
21524 if (allow_none || value != Py_None) {
21525 result = PyDict_SetItemString(moddict, to_name, value);
21526 }
21527 Py_DECREF(value);
21528 } else if (PyErr_ExceptionMatches(PyExc_AttributeError)) {
21529 PyErr_Clear();
21530 } else {
21531 result = -1;
21532 }
21533 return result;
21534 }
21535 static CYTHON_SMALL_CODE PyObject* __pyx_pymod_create(PyObject *spec, CYTHON_UNUSED PyModuleDef *def)
{
21536 PyObject *module = NULL, *moddict, *modname;
21537 if (__Pyx_check_single_interpreter())
21538 return NULL;
21539 if (__pyx_m)
21540 return __Pyx_NewRef(__pyx_m);
21541 modname = PyObject_GetAttrString(spec, "name");
21542 if (unlikely(!modname)) goto bad;
21543 module = PyModule_NewObject(modname);
21544 Py_DECREF(modname);
21545 if (unlikely(!module)) goto bad;
21546 moddict = PyModule_GetDict(module);
21547 if (unlikely(!moddict)) goto bad;

```

```

21548 if (unlikely(__Pyx_copy_spec_to_module(spec, moddict, "loader", "__loader__", 1) < 0)) goto bad;
21549 if (unlikely(__Pyx_copy_spec_to_module(spec, moddict, "origin", "__file__", 1) < 0)) goto bad;
21550 if (unlikely(__Pyx_copy_spec_to_module(spec, moddict, "parent", "__package__", 1) < 0)) goto bad;
21551 if (unlikely(__Pyx_copy_spec_to_module(spec, moddict, "submodule_search_locations", "__path__", 0)
 < 0)) goto bad;
21552 return module;
21553 bad:
21554 Py_XDECREF(module);
21555 return NULL;
21556 }
21557
21558
21559 static CYTHON_SMALL_CODE int __pyx_pymod_exec_PyClical(PyObject *__pyx_pyinit_module)
21560 #endif
21561 #endif
21562 {
21563 PyObject *__pyx_t_1 = NULL;
21564 PyObject *__pyx_t_2 = NULL;
21565 PyObject *__pyx_t_3 = NULL;
21566 int __pyx_t_4;
21567 int __pyx_lineno = 0;
21568 const char *__pyx_filename = NULL;
21569 int __pyx_clineno = 0;
21570 __Pyx_RefNannyDeclarations
21571 #if CYTHON_PEP489_MULTI_PHASE_INIT
21572 if (__pyx_m) {
21573 if (__pyx_m == __pyx_pyinit_module) return 0;
21574 PyErr_SetString(PyExc_RuntimeError, "Module 'PyClical' has already been imported.
 Re-initialisation is not supported.");
21575 return -1;
21576 }
21577 #elif PY_MAJOR_VERSION >= 3
21578 if (__pyx_m) return __Pyx_NewRef(__pyx_m);
21579 #endif
21580 #if CYTHON_REFNANNY
21581 __Pyx_RefNanny = __Pyx_RefNannyImportAPI("refnanny");
21582 if (!__Pyx_RefNanny) {
21583 PyErr_Clear();
21584 __Pyx_RefNanny = __Pyx_RefNannyImportAPI("Cython.Runtime.refnanny");
21585 if (!__Pyx_RefNanny)
21586 Py_FatalError("failed to import 'refnanny' module");
21587 }
21588 #endif
21589 __Pyx_RefNannySetupContext("__Pyx_PyMODINIT_FUNC PyInit_PyClical(void)", 0);
21590 if (__Pyx_check_binary_version() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21591 #ifdef __Pxy_PyFrame_Initialize_Offsets
21592 __Pxy_PyFrame_Initialize_Offsets();
21593 #endif
21594 __pyx_empty_tuple = PyTuple_New(0); if (unlikely(!__pyx_empty_tuple)) __PYX_ERR(0, 1,
 __pyx_L1_error)
21595 __pyx_empty_bytes = PyBytes_FromStringAndSize("", 0); if (unlikely(!__pyx_empty_bytes)) __PYX_ERR(0,
 1, __pyx_L1_error)
21596 __pyx_empty_unicode = PyUnicode_FromStringAndSize("", 0); if (unlikely(!__pyx_empty_unicode))
 __PYX_ERR(0, 1, __pyx_L1_error)
21597 #ifdef __Pyx_CyFunction_USED
21598 if (__pyx_CyFunction_init() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21599 #endif
21600 #ifdef __Pyx_FusedFunction_USED
21601 if (__pyx_FusedFunction_init() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21602 #endif
21603 #ifdef __Pyx_Coroutine_USED
21604 if (__pyx_Coroutine_init() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21605 #endif
21606 #ifdef __Pyx_Generator_USED
21607 if (__pyx_Generator_init() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21608 #endif
21609 #ifdef __Pyx_AsyncGen_USED
21610 if (__pyx_AsyncGen_init() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21611 #endif
21612 #ifdef __Pyx_StopAsyncIteration_USED
21613 if (__pyx_StopAsyncIteration_init() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21614 #endif
21615 /*--- Library function declarations ---*/
21616 /*--- Threads initialization code ---*/
21617 #if defined(WITH_THREAD) && PY_VERSION_HEX < 0x030700F0 && defined(__PYX_FORCE_INIT_THREADS) &&
 __PYX_FORCE_INIT_THREADS
21618 PyEval_InitThreads();
21619 #endif
21620 /*--- Module creation code ---*/
21621 #if CYTHON_PEP489_MULTI_PHASE_INIT
21622 __pyx_m = __pyx_pyinit_module;
21623 Py_INCREF(__pyx_m);
21624 #else
21625 #if PY_MAJOR_VERSION < 3
21626 __pyx_m = Py_InitModule4("PyClical", __pyx_methods, 0, 0, PYTHON_API_VERSION); Py_XINCREF(__pyx_m);
21627 #else
21628 __pyx_m = PyModule_Create(&__pyx_moduledef);

```

```

21629 #endif
21630 if (unlikely(!__pyx_m)) __PYX_ERR(0, 1, __pyx_L1_error)
21631 #endif
21632 __pyx_d = PyModule_GetDict(__pyx_m); if (unlikely(!__pyx_d)) __PYX_ERR(0, 1, __pyx_L1_error)
21633 Py_INCREF(__pyx_d);
21634 __pyx_b = PyImport_AddModule(__Pyx_BUILTIN_MODULE_NAME); if (unlikely(!__pyx_b)) __PYX_ERR(0, 1,
__pyx_L1_error)
21635 Py_INCREF(__pyx_b);
21636 __pyx_cython_runtime = PyImport_AddModule((char *) "cython_runtime"); if
(unlikely(!__pyx_cython_runtime)) __PYX_ERR(0, 1, __pyx_L1_error)
21637 Py_INCREF(__pyx_cython_runtime);
21638 if (PyObject_SetAttrString(__pyx_m, "__builtins__", __pyx_b) < 0) __PYX_ERR(0, 1, __pyx_L1_error);
21639 /*--- Initialize various global constants etc. ---*/
21640 if (__Pyx_InitGlobals() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21641 #if PY_MAJOR_VERSION < 3 && (__PYX_DEFAULT_STRING_ENCODING_IS_ASCII ||
__PYX_DEFAULT_STRING_ENCODING_IS_DEFAULT)
21642 if (__Pyx_init_sys_getdefaultencoding_params() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21643 #endif
21644 if (__pyx_module_is_main_PyClical) {
21645 if (PyObject_SetAttr(__pyx_m, __pyx_n_s_name, __pyx_n_s_main) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21646 }
21647 #if PY_MAJOR_VERSION >= 3
21648 {
21649 PyObject *modules = PyImport_GetModuleDict(); if (unlikely(!modules)) __PYX_ERR(0, 1,
__pyx_L1_error)
21650 if (!PyDict_GetItemString(modules, "PyClical")) {
21651 if (unlikely(PyDict_SetItemString(modules, "PyClical", __pyx_m) < 0)) __PYX_ERR(0, 1,
__pyx_L1_error)
21652 }
21653 }
21654 #endif
21655 /*--- Builtin init code ---*/
21656 if (__Pyx_InitCachedBuiltins() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21657 /*--- Constants init code ---*/
21658 if (__Pyx_InitCachedConstants() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21659 /*--- Global type/function init code ---*/
21660 (void)__Pyx_modinit_global_init_code();
21661 (void)__Pyx_modinit_variable_export_code();
21662 (void)__Pyx_modinit_function_export_code();
21663 if (unlikely(__Pyx_modinit_type_init_code() < 0)) __PYX_ERR(0, 1, __pyx_L1_error)
21664 (void)__Pyx_modinit_type_import_code();
21665 (void)__Pyx_modinit_variable_import_code();
21666 (void)__Pyx_modinit_function_import_code();
21667 /*--- Execution code ---*/
21668 #if defined(__Pyx_Generator_USED) || defined(__Pyx_Coroutine_USED)
21669 if (__Pyx_patch_abc() < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21670 #endif
21671
21672 /* "PyClical.pyx":29
21673 * # C. Doran and A. Lasenby, "Geometric algebra for physicists", Cambridge, 2003.
21674 *
21675 * import math # ««««««««
21676 * import numbers
21677 * import collections
21678 */
21679 __pyx_t_1 = __Pyx_Import(__pyx_n_s_math, 0, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 29,
__pyx_L1_error)
21680 __Pyx_GOTREF(__pyx_t_1);
21681 if (PyDict_SetItem(__pyx_d, __pyx_n_s_math, __pyx_t_1) < 0) __PYX_ERR(0, 29, __pyx_L1_error)
21682 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21683
21684 /* "PyClical.pyx":30
21685 *
21686 * import math
21687 * import numbers # ««««««««
21688 * import collections
21689 *
21690 */
21691 __pyx_t_1 = __Pyx_Import(__pyx_n_s_numbers, 0, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 30,
__pyx_L1_error)
21692 __Pyx_GOTREF(__pyx_t_1);
21693 if (PyDict_SetItem(__pyx_d, __pyx_n_s_numbers, __pyx_t_1) < 0) __PYX_ERR(0, 30, __pyx_L1_error)
21694 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21695
21696 /* "PyClical.pyx":31
21697 * import math
21698 * import numbers
21699 * import collections # ««««««««
21700 *
21701 * from PyClical cimport *
21702 */
21703 __pyx_t_1 = __Pyx_Import(__pyx_n_s_collections, 0, 0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 31,
__pyx_L1_error)
21704 __Pyx_GOTREF(__pyx_t_1);
21705 if (PyDict_SetItem(__pyx_d, __pyx_n_s_collections, __pyx_t_1) < 0) __PYX_ERR(0, 31, __pyx_L1_error)
21706 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21707

```

```

21708 /* "PyClical.pyx":35
21709 * from PyClical cimport *
21710 *
21711 * __version__ = str(glucat_package_version,'utf-8') # ««««««««
21712 *
21713 * # Forward reference
21714 */
21715 __pyx_t_1 = __pyx_convert_PyBytes_string_to_py_std_in_string(glucat_package_version); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 35, __pyx_L1_error)
21716 __Pyx_GOTREF(__pyx_t_1);
21717 __pyx_t_2 = PyTuple_New(2); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 35, __pyx_L1_error)
21718 __Pyx_GOTREF(__pyx_t_2);
21719 __Pyx_GIVEREF(__pyx_t_1);
21720 PyTuple_SET_ITEM(__pyx_t_2, 0, __pyx_t_1);
21721 __Pyx_INCREF(__pyx_kp_u_utf_8);
21722 __Pyx_GIVEREF(__pyx_kp_u_utf_8);
21723 PyTuple_SET_ITEM(__pyx_t_2, 1, __pyx_kp_u_utf_8);
21724 __pyx_t_1 = 0;
21725 __pyx_t_1 = __Pyx_PyObject_Call(((PyObject *)(&PyUnicode_Type)), __pyx_t_2, NULL); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 35, __pyx_L1_error)
21726 __Pyx_GOTREF(__pyx_t_1);
21727 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
21728 if (PyDict_SetItem(__pyx_d, __pyx_n_s_version, __pyx_t_1) < 0) __PYX_ERR(0, 35, __pyx_L1_error)
21729 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21730
21731 /* "PyClical.pyx":406
21732 * return index_set_to_str(self.unwrap()).decode()
21733 *
21734 * def index_set_hidden_doctests(): # ««««««««
21735 * """
21736 * Tests for functions that Doctest cannot see.
21737 */
21738 __pyx_t_1 = PyCFunction_NewEx(&__pyx_mdef_8PyClical_lindex_set_hidden_doctests, NULL,
__pyx_n_s_PyClical); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 406, __pyx_L1_error)
21739 __Pyx_GOTREF(__pyx_t_1);
21740 if (PyDict_SetItem(__pyx_d, __pyx_n_s_index_set_hidden_doctests, __pyx_t_1) < 0) __PYX_ERR(0, 406,
__pyx_L1_error)
21741 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21742
21743 /* "PyClical.pyx":1253
21744 * return clifford_to_str(self.unwrap()).decode()
21745 *
21746 * def clifford_hidden_doctests(): # ««««««««
21747 * """
21748 * Tests for functions that Doctest cannot see.
21749 */
21750 __pyx_t_1 = PyCFunction_NewEx(&__pyx_mdef_8PyClical_9clifford_hidden_doctests, NULL,
__pyx_n_s_PyClical); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1253, __pyx_L1_error)
21751 __Pyx_GOTREF(__pyx_t_1);
21752 if (PyDict_SetItem(__pyx_d, __pyx_n_s_clifford_hidden_doctests, __pyx_t_1) < 0) __PYX_ERR(0, 1253,
__pyx_L1_error)
21753 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21754
21755 /* "PyClical.pyx":1905
21756 *
21757 * # Some abbreviations.
21758 * scalar_epsilon = epsilon # ««««««««
21759 *
21760 * pi = atan(clifford(1.0)) * 4.0
21761 */
21762 __pyx_t_1 = PyFloat_FromDouble(epsilon); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1905,
__pyx_L1_error)
21763 __Pyx_GOTREF(__pyx_t_1);
21764 if (PyDict_SetItem(__pyx_d, __pyx_n_s_scalar_epsilon, __pyx_t_1) < 0) __PYX_ERR(0, 1905,
__pyx_L1_error)
21765 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21766
21767 /* "PyClical.pyx":1907
21768 * scalar_epsilon = epsilon
21769 *
21770 * pi = atan(clifford(1.0)) * 4.0 # ««««««««
21771 * tau = atan(clifford(1.0)) * 8.0
21772 *
21773 */
21774 __pyx_t_1 = __Pyx_PyObject_Call(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_tuple__15,
NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1907, __pyx_L1_error)
21775 __Pyx_GOTREF(__pyx_t_1);
21776 __pyx_t_2 = __pyx_f_8PyClical_atan(__pyx_t_1, 0, NULL); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1907,
__pyx_L1_error)
21777 __Pyx_GOTREF(__pyx_t_2);
21778 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21779 __pyx_t_1 = PyNumber_Multiply(__pyx_t_2, __pyx_float_4_0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0,
1907, __pyx_L1_error)
21780 __Pyx_GOTREF(__pyx_t_1);
21781 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
21782 if (PyDict_SetItem(__pyx_d, __pyx_n_s_pi, __pyx_t_1) < 0) __PYX_ERR(0, 1907, __pyx_L1_error)
21783 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;

```

```

21784
21785 /* "PyClical.pyx":1908
21786 *
21787 * pi = atan(clifford(1.0)) * 4.0
21788 * tau = atan(clifford(1.0)) * 8.0 # ««««««««
21789 *
21790 * cl = clifford
21791 */
21792 __pyx_t_1 = __Pyx_PyObject_Call(((PyObject *)__pyx_ptype_8PyClical_clifford), __pyx_tuple__15,
NULL); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1908, __pyx_L1_error)
21793 __Pyx_GOTREF(__pyx_t_1);
21794 __pyx_t_2 = __pyx_f_8PyClical_atan(__pyx_t_1, 0, NULL); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1908,
__pyx_L1_error)
21795 __Pyx_GOTREF(__pyx_t_2);
21796 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21797 __pyx_t_1 = PyNumber_Multiply(__pyx_t_2, __pyx_float_8_0); if (unlikely(!__pyx_t_1)) __PYX_ERR(0,
1908, __pyx_L1_error)
21798 __Pyx_GOTREF(__pyx_t_1);
21799 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
21800 if (PyDict_SetItem(__pyx_d, __pyx_n_s_tau, __pyx_t_1) < 0) __PYX_ERR(0, 1908, __pyx_L1_error)
21801 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21802
21803 /* "PyClical.pyx":1910
21804 * tau = atan(clifford(1.0)) * 8.0
21805 *
21806 * cl = clifford # ««««««««
21807 * """
21808 * Abbreviation for clifford.
21809 */
21810 if (PyDict_SetItem(__pyx_d, __pyx_n_s_cl, ((PyObject *)__pyx_ptype_8PyClical_clifford)) < 0)
__PYX_ERR(0, 1910, __pyx_L1_error)
21811
21812 /* "PyClical.pyx":1928
21813 * """
21814 *
21815 * ist = index_set # ««««««««
21816 * """
21817 * Abbreviation for index_set.
21818 */
21819 if (PyDict_SetItem(__pyx_d, __pyx_n_s_ist, ((PyObject *)__pyx_ptype_8PyClical_index_set)) < 0)
__PYX_ERR(0, 1928, __pyx_L1_error)
21820
21821 /* "PyClical.pyx":1936
21822 * """
21823 *
21824 * def e(obj): # ««««««««
21825 * """
21826 * Abbreviation for clifford(index_set(obj)).
21827 */
21828 __pyx_t_1 = PyCFunction_NewEx(&__pyx_mdef_8PyClical_89e, NULL, __pyx_n_s_PyClical); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1936, __pyx_L1_error)
21829 __Pyx_GOTREF(__pyx_t_1);
21830 if (PyDict_SetItem(__pyx_d, __pyx_n_s_e, __pyx_t_1) < 0) __PYX_ERR(0, 1936, __pyx_L1_error)
21831 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21832
21833 /* "PyClical.pyx":1949
21834 * return clifford(index_set(obj))
21835 *
21836 * def istpq(p, q): # ««««««««
21837 * """
21838 * Abbreviation for index_set({-q,...p}).
21839 */
21840 __pyx_t_1 = PyCFunction_NewEx(&__pyx_mdef_8PyClical_91istpq, NULL, __pyx_n_s_PyClical); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1949, __pyx_L1_error)
21841 __Pyx_GOTREF(__pyx_t_1);
21842 if (PyDict_SetItem(__pyx_d, __pyx_n_s_istpq, __pyx_t_1) < 0) __PYX_ERR(0, 1949, __pyx_L1_error)
21843 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21844
21845 /* "PyClical.pyx":1958
21846 * return index_set(set(range(-q,p+1)))
21847 *
21848 * ninf3 = e(4) + e(-1) # Null infinity point in 3D Conformal Geometric Algebra [DL]. #
««««««««
21849 * nbar3 = e(4) - e(-1) # Null bar point in 3D Conformal Geometric Algebra [DL].
21850 *
21851 */
21852 __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_e); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1958,
__pyx_L1_error)
21853 __Pyx_GOTREF(__pyx_t_1);
21854 __pyx_t_2 = __Pyx_PyObject_Call(__pyx_t_1, __pyx_tuple__20, NULL); if (unlikely(!__pyx_t_2))
__PYX_ERR(0, 1958, __pyx_L1_error)
21855 __Pyx_GOTREF(__pyx_t_2);
21856 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21857 __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_e); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1958,
__pyx_L1_error)
21858 __Pyx_GOTREF(__pyx_t_1);
21859 __pyx_t_3 = __Pyx_PyObject_Call(__pyx_t_1, __pyx_tuple__21, NULL); if (unlikely(!__pyx_t_3))

```

```

__PYX_ERR(0, 1958, __pyx_l1_error)
21860 __Pyx_GOTREF(__pyx_t_3);
21861 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21862 __pyx_t_1 = PyNumber_Add(__pyx_t_2, __pyx_t_3); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1958,
__pyx_l1_error)
21863 __Pyx_GOTREF(__pyx_t_1);
21864 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
21865 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
21866 if (PyDict_SetItem(__pyx_d, __pyx_n_s_ninf3, __pyx_t_1) < 0) __PYX_ERR(0, 1958, __pyx_l1_error)
21867 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21868
21869 /* "PyCliclcal.pyx":1959
21870 *
21871 * ninf3 = e(4) + e(-1) # Null infinity point in 3D Conformal Geometric Algebra [DL].
21872 * nbar3 = e(4) - e(-1) # Null bar point in 3D Conformal Geometric Algebra [DL]. # ««««««««
21873 *
21874 * # Doctest interface.
21875 */
21876 __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_e); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1959,
__pyx_l1_error)
21877 __Pyx_GOTREF(__pyx_t_1);
21878 __pyx_t_3 = __Pyx_PyObject_Call(__pyx_t_1, __pyx_tuple__20, NULL); if (unlikely(!__pyx_t_3))
__PYX_ERR(0, 1959, __pyx_l1_error)
21879 __Pyx_GOTREF(__pyx_t_3);
21880 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21881 __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_e); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1959,
__pyx_l1_error)
21882 __Pyx_GOTREF(__pyx_t_1);
21883 __pyx_t_2 = __Pyx_PyObject_Call(__pyx_t_1, __pyx_tuple__21, NULL); if (unlikely(!__pyx_t_2))
__PYX_ERR(0, 1959, __pyx_l1_error)
21884 __Pyx_GOTREF(__pyx_t_2);
21885 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21886 __pyx_t_1 = PyNumber_Subtract(__pyx_t_3, __pyx_t_2); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1959,
__pyx_l1_error)
21887 __Pyx_GOTREF(__pyx_t_1);
21888 __Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
21889 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
21890 if (PyDict_SetItem(__pyx_d, __pyx_n_s_nbar3, __pyx_t_1) < 0) __PYX_ERR(0, 1959, __pyx_l1_error)
21891 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21892
21893 /* "PyCliclcal.pyx":1962
21894 *
21895 * # Doctest interface.
21896 * def _test(): # ««««««««
21897 * import PyCliclcal, doctest
21898 * return doctest.testmod(PyCliclcal)
21899 */
21900 __pyx_t_1 = PyCFunction_NewEx(&__pyx_mdef_8PyCliclcal_93_test, NULL, __pyx_n_s_PyCliclcal); if
(unlikely(!__pyx_t_1)) __PYX_ERR(0, 1962, __pyx_l1_error)
21901 __Pyx_GOTREF(__pyx_t_1);
21902 if (PyDict_SetItem(__pyx_d, __pyx_n_s_test, __pyx_t_1) < 0) __PYX_ERR(0, 1962, __pyx_l1_error)
21903 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21904
21905 /* "PyCliclcal.pyx":1966
21906 * return doctest.testmod(PyCliclcal)
21907 *
21908 * if __name__ == "__main__": # ««««««««
21909 * _test()
21910 */
21911 __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_name); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1966,
__pyx_l1_error)
21912 __Pyx_GOTREF(__pyx_t_1);
21913 __pyx_t_4 = (__Pyx_PyUnicode_Equals(__pyx_t_1, __pyx_n_u_main, Py_EQ)); if (unlikely(__pyx_t_4 < 0))
__PYX_ERR(0, 1966, __pyx_l1_error)
21914 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21915 if (__pyx_t_4) {
21916
21917 /* "PyCliclcal.pyx":1967
21918 *
21919 * if __name__ == "__main__":
21920 * _test() # ««««««««
21921 */
21922 __Pyx_GetModuleGlobalName(__pyx_t_1, __pyx_n_s_test); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 1967,
__pyx_l1_error)
21923 __Pyx_GOTREF(__pyx_t_1);
21924 __pyx_t_2 = __Pyx_PyObject_CallNoArg(__pyx_t_1); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1967,
__pyx_l1_error)
21925 __Pyx_GOTREF(__pyx_t_2);
21926 __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
21927 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
21928
21929 /* "PyCliclcal.pyx":1966
21930 * return doctest.testmod(PyCliclcal)
21931 *
21932 * if __name__ == "__main__": # ««««««««
21933 * _test()
21934 */

```



```

21935 }
21936
21937 /* "PyClical.pyx":1
21938 * # -*- coding: utf-8 -*-
21939 * # cython: language_level=3
21940 * # distutils: language = c++
21941 */
21942 __pyx_t_2 = __Pyx_PyDict_NewPresized(111); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 1, __pyx_L1_error)
21943 __Pyx_GOTREF(__pyx_t_2);
21944 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_copy_line_65,
21945 __pyx_kp_u_Copy_this_index_set_object_s_in) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21946 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_setitem_line_179,
21947 __pyx_kp_u_Set_the_value_of_an_index_set_o) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21948 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_getitem_line_191,
21949 __pyx_kp_u_Get_the_value_of_an_index_set_o) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21950 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_iter_line_229,
21951 __pyx_kp_u_Iterate_over_the_indices_of_an) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21952 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_invert_line_240,
21953 __pyx_kp_u_Set_complement_not_print_index) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21954 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_xor_line_249,
21955 __pyx_kp_u_Symmetric_set_difference_exclus) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21956 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_ixor_line_260,
21957 __pyx_kp_u_Symmetric_set_difference_exclus_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21958 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_and_line_271,
21959 __pyx_kp_u_Set_intersection_and_print_inde) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21960 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_1and_line_282,
21961 __pyx_kp_u_Set_intersection_and_x_index_se) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21962 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_or_line_293,
21963 __pyx_kp_u_Set_union_or_print_index_set_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21964 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_ior_line_304,
21965 __pyx_kp_u_Set_union_or_x_index_set_1_x_in) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21966 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_count_line_315,
21967 __pyx_kp_u_Cardinality_Number_of_indices_i) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21968 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_count_neg_line_324,
21969 __pyx_kp_u_Number_of_negative_indices_incl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21970 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_count_pos_line_333,
21971 __pyx_kp_u_Number_of_positive_indices_incl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21972 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_min_line_342,
21973 __pyx_kp_u_Minimum_member_index_set_1_1_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21974 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_max_line_351,
21975 __pyx_kp_u_Maximum_member_index_set_1_1_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21976 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_sign_of_mult_line_366,
21977 __pyx_kp_u_Sign_of_geometric_product_of_tw) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21978 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_sign_of_square_line_37,
21979 __pyx_kp_u_Sign_of_geometric_square_of_a_C) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21980 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_repr_line_384,
21981 __pyx_kp_u_The_official_string_representat) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21982 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_str_line_395,
21983 __pyx_kp_u_The_informal_string_representat) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21984 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_index_set_hidden_doctests_line_4,
21985 __pyx_kp_u_Tests_for_functions_that_Doctes) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21986 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_compare_line_492,
21987 __pyx_kp_u_lexicographic_compare_eg_3_4_5) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21988 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_min_neg_line_504,
21989 __pyx_kp_u_Minimum_negative_index_or_0_if) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21990 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_max_pos_line_513,
21991 __pyx_kp_u_Maximum_positive_index_or_0_if) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21992 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_copy_line_556,
21993 __pyx_kp_u_Copy_this_clifford_object_x_cli) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21994 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_iter_line_638,
21995 __pyx_kp_u_Not_applicable_for_a_in_cliffor) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21996 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_reframe_line_649,
21997 __pyx_kp_u_Put_self_into_a_larger_frame_co) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21998 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_getitem_line_707,
21999 __pyx_kp_u_Subscripting_map_from_index_set) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22000 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_neg_line_722,
22001 __pyx_kp_u_Unary_print_clifford_1_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22002 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_pos_line_731,
22003 __pyx_kp_u_Unary_print_clifford_1_1_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22004 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_add_line_740,
22005 __pyx_kp_u_Geometric_sum_print_clifford_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22006 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_iadd_line_751,
22007 __pyx_kp_u_Geometric_sum_x_clifford_1_x_cl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22008 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_sub_line_760,
22009 __pyx_kp_u_Geometric_difference_print_clif) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22010 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_isub_line_771,
22011 __pyx_kp_u_Geometric_difference_x_clifford) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22012 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_mul_line_780,
22013 __pyx_kp_u_Geometric_product_print_cliffor) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22014 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_imul_line_793,
22015 __pyx_kp_u_Geometric_product_x_clifford_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22016 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_mod_line_806,
22017 __pyx_kp_u_Contraction_print_clifford_1_cl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22018 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_imod_line_821,
22019 __pyx_kp_u_Contraction_x_clifford_1_x_clif) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22020 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_and_line_836,
22021 __pyx_kp_u_Inner_product_print_clifford_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)

```

```

21983 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_iand_line_851,
21984 __pyx_kp_u_Inner_product_x_clifford_1_x_cl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21984 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_xor_line_866,
21985 __pyx_kp_u_Outer_product_print_clifford_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21985 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_ixor_line_881,
21986 __pyx_kp_u_Outer_product_x_clifford_1_x_cl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21986 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_truediv_line_896,
21987 __pyx_kp_u_Geometric_quotient_print_cliffo) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21987 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_idiv_line_911,
21988 __pyx_kp_u_Geometric_quotient_x_clifford_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21988 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_inv_line_926,
21989 __pyx_kp_u_Geometric_multiplicative_invers) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21989 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_or_line_939,
21990 __pyx_kp_u_Transform_left_hand_side_using) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21990 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_ior_line_950,
21991 __pyx_kp_u_Transform_left_hand_side_using_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21991 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_pow_line_961,
21992 __pyx_kp_u_Power_self_to_the_m_x_clifford) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21992 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_pow_line_980,
21993 __pyx_kp_u_Power_self_to_the_m_x_clifford_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21993 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_outer_pow_line_1004,
21994 __pyx_kp_u_Outer_product_power_x_clifford) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21994 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_call_line_1020,
21995 __pyx_kp_u_Pure_grade_vector_part_print_cl) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21995 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_scalar_line_1039,
21996 __pyx_kp_u_Scalar_part_clifford_1_l_1_2_sc) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21996 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_pure_line_1050,
21997 __pyx_kp_u_Pure_part_print_clifford_1_l_1) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21997 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_even_line_1061,
21998 __pyx_kp_u_Even_part_of_multivector_sum_of) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21998 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_odd_line_1070,
21999 __pyx_kp_u_Odd_part_of_multivector_sum_of) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
21999 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_vector_part_line_1079,
22000 __pyx_kp_u_Vector_part_of_multivector_as_a) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22000 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_involute_line_1107,
22001 __pyx_kp_u_Main_involution_each_i_is_repla) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22001 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_reverse_line_1123,
22002 __pyx_kp_u_Reversion_eg_clifford_1_cliffor) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22002 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_conj_line_1138,
22003 __pyx_kp_u_Conjugation_reverse_o_involute) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22003 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_quad_line_1153,
22004 __pyx_kp_u_Quadratic_form_rev_x_x_0_print) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22004 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_norm_line_1164,
22005 __pyx_kp_u_Norm_sum_of_squares_of_coordina) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22005 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_abs_line_1175,
22006 __pyx_kp_u_Absolute_value_square_root_of_n) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22006 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_max_abs_line_1184,
22007 __pyx_kp_u_Maximum_of_absolute_values_of_c) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22007 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_truncated_line_1195,
22008 __pyx_kp_u_Remove_all_terms_of_self_with_r) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22008 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_isinf_line_1206,
22009 __pyx_kp_u_Check_if_a_multivector_contains) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22009 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_isnan_line_1215,
22010 __pyx_kp_u_Check_if_a_multivector_contains_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22010 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_frame_line_1224,
22011 __pyx_kp_u_Subalgebra_generated_by_all_gen) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22011 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_repr_line_1235,
22012 __pyx_kp_u_The_official_string_representat_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22012 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_str_line_1244,
22013 __pyx_kp_u_The_informal_string_representat_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22013 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_clifford_hidden_doctests_line_12,
22014 __pyx_kp_u_Tests_for_functions_that_Doctes_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22014 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_error_squared_tol_line_1337,
22015 __pyx_kp_u_Quadratic_norm_error_tolerance) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22015 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_error_squared_line_1346,
22016 __pyx_kp_u_Relative_or_absolute_error_usin) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22016 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_approx_equal_line_1359,
22017 __pyx_kp_u_Test_for_approximate_equality_o) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22017 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_inv_line_1378,
22018 __pyx_kp_u_Geometric_multiplicative_invers_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22018 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_scalar_line_1393,
22019 __pyx_kp_u_Scalar_part_scalar_clifford_1_l) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22019 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_real_line_1404, __pyx_kp_u_Real_part_synonym_for_scalar_pa)
22020 < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22020 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_imag_line_1415, __pyx_kp_u_Imaginary_part_deprecated_alway)
22021 < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22021 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_pure_line_1426, __pyx_kp_u_Pure_part_print_pure_clifford_1)
22022 < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22022 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_even_line_1437,
22023 __pyx_kp_u_Even_part_of_multivector_sum_of_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22023 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_odd_line_1446, __pyx_kp_u_Odd_part_of_multivector_sum_of_2)
22024 < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22024 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_involute_line_1455,
22025 __pyx_kp_u_Main_involution_each_i_is_repla_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22025 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_reverse_line_1470,
22026 __pyx_kp_u_Reversion_eg_l_2_2_1_print_reve) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22026 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_conj_line_1485,

```

```

__pyx_kp_u_Conjugation_reverse_o_involute_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22027 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_quad_line_1500,
__pyx_kp_u_Quadratic_form_rev_x_x_0_print_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22028 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_norm_line_1511, __pyx_kp_u_norm_sum_of_squares_of_coordina
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22029 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_abs_line_1522, __pyx_kp_u_Absolute_value_of_multivector_m)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22030 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_max_abs_line_1531,
__pyx_kp_u_Maximum_absolute_value_of_coord) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22031 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_pow_line_1543, __pyx_kp_u_Integer_power_of_multivector_ob)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22032 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_outer_pow_line_1567,
__pyx_kp_u_Outer_product_power_of_multivec) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22033 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_complexifier_line_1576,
__pyx_kp_u_Square_root_of_1_which_commutates) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22034 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_sqrt_line_1591, __pyx_kp_u_Square_root_of_multivector_with)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22035 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_exp_line_1614, __pyx_kp_u_Exponential_of_multivector_x_cl)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22036 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_log_line_1628, __pyx_kp_u_Natural_logarithm_of_multivecto)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22037 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_cos_line_1651, __pyx_kp_u_Cosine_of_multivector_with_opti)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22038 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_acos_line_1668, __pyx_kp_u_Inverse_cosine_of_multivector_w)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22039 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_cosh_line_1689, __pyx_kp_u_Hyperbolic_cosine_of_multivecto)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22040 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_acosh_line_1705,
__pyx_kp_u_Inverse_hyperbolic_cosine_of_mu) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22041 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_sin_line_1728, __pyx_kp_u_Sine_of_multivector_with_option)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22042 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_asin_line_1747, __pyx_kp_u_Inverse_sine_of_multivector_wit)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22043 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_sinh_line_1768, __pyx_kp_u_Hyperbolic_sine_of_multivector)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22044 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_asinh_line_1782,
__pyx_kp_u_Inverse_hyperbolic_sine_of_mult) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22045 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_tan_line_1801, __pyx_kp_u_Tangent_of_multivector_with_opt)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22046 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_atan_line_1818, __pyx_kp_u_Inverse_tangent_of_multivector)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22047 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_tanh_line_1835, __pyx_kp_u_Hyperbolic_tangent_of_multivect)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22048 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_atanh_line_1847,
__pyx_kp_u_Inverse_hyperbolic_tangent_of_m) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22049 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_random_clifford_line_1864,
__pyx_kp_u_Random_multivector_within_a_fra) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22050 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_cga3_line_1873, __pyx_kp_u_Convert_Euclidean_3D_multivecto)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22051 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_cga3std_line_1882,
__pyx_kp_u_Convert_CGA3_null_vector_to_sta) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22052 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_agc3_line_1893, __pyx_kp_u_Convert_CGA3_null_vector_to_Euc)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22053 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_e_line_1936, __pyx_kp_u_Abbreviation_for_clifford_index) <
0) __PYX_ERR(0, 1, __pyx_L1_error)
22054 if (PyDict_SetItem(__pyx_t_2, __pyx_kp_u_istpq_line_1949, __pyx_kp_u_Abbreviation_for_index_set_q_p)
< 0) __PYX_ERR(0, 1, __pyx_L1_error)
22055 if (PyDict_SetItem(__pyx_d, __pyx_n_s_test_2, __pyx_t_2) < 0) __PYX_ERR(0, 1, __pyx_L1_error)
22056 __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
22057
22058 /* "string.to_py":55
22059 *
22060 * @cname("__pyx_convert_PyByteArray_string_to_py_std_in_string")
22061 * cdef inline object __pyx_convert_PyByteArray_string_to_py_std_in_string(const string& s):
«««««
22062 * return __Pyx_PyByteArray_FromStringAndSize(s.data(), s.size())
22063 *
22064 */
22065
22066 /---- Wrapped vars code ----*/
22067
22068 goto __pyx_L0;
22069 __pyx_L1_error:;
22070 __Pyx_XDECREF(__pyx_t_1);
22071 __Pyx_XDECREF(__pyx_t_2);
22072 __Pyx_XDECREF(__pyx_t_3);
22073 if (__pyx_m) {
22074 if (__pyx_d) {
22075 __Pyx_AddTraceback("init PyClical", __pyx_clineno, __pyx_lineno, __pyx_filename);
22076 }
22077 Py_CLEAR(__pyx_m);
22078 } else if (!PyErr_Occurred()) {
22079 PyErr_SetString(PyExc_ImportError, "init PyClical");
22080 }
22081 __pyx_L0:;
22082 __Pyx_RefNannyFinishContext();
22083 #if CYTHON_PEP489_MULTI_PHASE_INIT

```

```

22084 return (__pyx_m != NULL) ? 0 : -1;
22085 #elif PY_MAJOR_VERSION >= 3
22086 return __pyx_m;
22087 #else
22088 return;
22089 #endif
22090 }
22091
22092 /* --- Runtime support code --- */
22093 /* Refnanny */
22094 #if CYTHON_REFNANNY
22095 static __Pyx_RefNannyAPIStruct *__Pyx_RefNannyImportAPI(const char *modname) {
22096 PyObject *m = NULL, *p = NULL;
22097 void *r = NULL;
22098 m = PyImport_ImportModule(modname);
22099 if (!m) goto end;
22100 p = PyObject_GetAttrString(m, "RefNannyAPI");
22101 if (!p) goto end;
22102 r = PyLong_AsVoidPtr(p);
22103 end:
22104 Py_XDECREF(p);
22105 Py_XDECREF(m);
22106 return (__Pyx_RefNannyAPIStruct *)r;
22107 }
22108 #endif
22109
22110 /* PyObjectGetAttrStr */
22111 #if CYTHON_USE_TYPE_SLOTS
22112 static CYTHON_INLINE PyObject* __Pyx_PyObject_GetAttrStr(PyObject* obj, PyObject* attr_name) {
22113 PyTypeObject* tp = Py_TYPE(obj);
22114 if (likely(tp->tp_getattro))
22115 return tp->tp_getattro(obj, attr_name);
22116 #if PY_MAJOR_VERSION < 3
22117 if (likely(tp->tp_getattr))
22118 return tp->tp_getattr(obj, PyString_AS_STRING(attr_name));
22119 #endif
22120 return PyObject_GetAttr(obj, attr_name);
22121 }
22122 #endif
22123
22124 /* GetBuiltinName */
22125 static PyObject* __Pyx_GetBuiltinName(PyObject *name) {
22126 PyObject* result = __Pyx_PyObject_GetAttrStr(__pyx_b, name);
22127 if (unlikely(!result)) {
22128 PyErr_Format(PyExc_NameError,
22129 #if PY_MAJOR_VERSION >= 3
22130 "name '%U' is not defined", name);
22131 #else
22132 "name '%.200s' is not defined", PyString_AS_STRING(name));
22133 #endif
22134 }
22135 return result;
22136 }
22137
22138 /* PyCFunctionFastCall */
22139 #if CYTHON_FAST_PYCALL
22140 static CYTHON_INLINE PyObject* __Pyx_PyCFunction_FastCall(PyObject *func_obj, PyObject **args,
22141 Py_ssize_t nargs) {
22142 PyCFunctionObject *func = (PyCFunctionObject*)func_obj;
22143 PyCFunction meth = PyCFunction_GET_FUNCTION(func);
22144 PyObject *self = PyCFunction_GET_SELF(func);
22145 int flags = PyCFunction_GET_FLAGS(func);
22146 assert(PyCFunction_Check(func));
22147 assert(METH_FASTCALL == (flags & ~(METH_CLASS | METH_STATIC | METH_COEXIST | METH_KEYWORDS |
22148 METH_STACKLESS)));
22149 assert(nargs >= 0);
22150 assert(nargs == 0 || args != NULL);
22151 /* _PyCFunction_FastCallDict() must not be called with an exception set,
22152 because it may clear it (directly or indirectly) and so the
22153 caller loses its exception */
22154 assert(!PyErr_Occurred());
22155 if ((PY_VERSION_HEX < 0x030700A0) || unlikely(flags & METH_KEYWORDS)) {
22156 return ((*((__Pyx_PyCFunctionFastWithKeywords)(void*)meth)) (self, args, nargs, NULL));
22157 } else {
22158 return ((*((__Pyx_PyCFunctionFast)(void*)meth)) (self, args, nargs);
22159 }
22160 }
22161 #endif
22162
22163 /* PyFunctionFastCall */
22164 #if CYTHON_FAST_PYCALL
22165 static PyObject* __Pyx_PyFunction_FastCallNoKw(PyCodeObject *co, PyObject **args, Py_ssize_t na,
22166 PyObject *globals) {
22167 PyFrameObject *f;
22168 PyThreadState *tstate = __Pyx_PyThreadState_Current;
22169 PyObject **fastlocals;
22170 Py_ssize_t i;

```

```

22169 PyObject *result;
22170 assert(globals != NULL);
22171 /* XXX Perhaps we should create a specialized
22172 PyFrame_New() that doesn't take locals, but does
22173 take builtins without sanity checking them.
22174 */
22175 assert(tstate != NULL);
22176 f = PyFrame_New(tstate, co, globals, NULL);
22177 if (f == NULL) {
22178 return NULL;
22179 }
22180 fastlocals = __Pyx_PyFrame_GetLocalsplus(f);
22181 for (i = 0; i < na; i++) {
22182 Py_INCREF(*args);
22183 fastlocals[i] = *args++;
22184 }
22185 result = PyEval_EvalFrameEx(f, 0);
22186 ++tstate->recursion_depth;
22187 Py_DECREF(f);
22188 --tstate->recursion_depth;
22189 return result;
22190 }
22191 #if 1 || PY_VERSION_HEX < 0x030600B1
22192 static PyObject *__Pyx_PyFunction_FastCallDict(PyObject *func, PyObject **args, Py_ssize_t nargs,
22193 PyObject *kwargs) {
22194 PyCodeObject *co = (PyCodeObject *)PyFunction_GET_CODE(func);
22195 PyObject *globals = PyFunction_GET_GLOBALS(func);
22196 PyObject *argdefs = PyFunction_GET_DEFAULTS(func);
22197 PyObject *closure;
22198 #if PY_MAJOR_VERSION >= 3
22199 PyObject *kwdefs;
22200 #endif
22201 PyObject *kwtuple, **k;
22202 PyObject **d;
22203 Py_ssize_t nd;
22204 Py_ssize_t nk;
22205 PyObject *result;
22206 assert(kwargs == NULL || PyDict_Check(kwargs));
22207 nk = kwargs ? PyDict_Size(kwargs) : 0;
22208 if (Py_EnterRecursiveCall((char*)" while calling a Python object")) {
22209 return NULL;
22210 }
22211 if (
22212 #if PY_MAJOR_VERSION >= 3
22213 co->co_kwonlyargcount == 0 &&
22214 #endif
22215 likely(kwargs == NULL || nk == 0) &&
22216 co->co_flags == (CO_OPTIMIZED | CO_NEWLOCALS | CO_NOFREE)) {
22217 if (argdefs == NULL && co->co_argcount == nargs) {
22218 result = __Pyx_PyFunction_FastCallNoKw(co, args, nargs, globals);
22219 goto done;
22220 }
22221 else if (nargs == 0 && argdefs != NULL
22222 && co->co_argcount == Py_SIZE(argdefs)) {
22223 /* function called with no arguments, but all parameters have
22224 a default value: use default values as arguments .*/
22225 args = &PyTuple_GET_ITEM(argdefs, 0);
22226 result = __Pyx_PyFunction_FastCallNoKw(co, args, Py_SIZE(argdefs), globals);
22227 goto done;
22228 }
22229 }
22230 if (kwargs != NULL) {
22231 Py_ssize_t pos, i;
22232 kwtuple = PyTuple_New(2 * nk);
22233 if (kwtuple == NULL) {
22234 result = NULL;
22235 goto done;
22236 }
22237 k = &PyTuple_GET_ITEM(kwtuple, 0);
22238 pos = i = 0;
22239 while (PyDict_Next(kwargs, &pos, &k[i], &k[i+1])) {
22240 Py_INCREF(k[i]);
22241 Py_INCREF(k[i+1]);
22242 i += 2;
22243 }
22244 nk = i / 2;
22245 }
22246 else {
22247 kwtuple = NULL;
22248 k = NULL;
22249 }
22250 closure = PyFunction_GET_CLOSURE(func);
22251 #if PY_MAJOR_VERSION >= 3
22252 kwdefs = PyFunction_GET_KW_DEFAULTS(func);
22253 #endif
22254 if (argdefs != NULL) {
22255 d = &PyTuple_GET_ITEM(argdefs, 0);

```

```

22255 nd = Py_SIZE(argdefs);
22256 }
22257 else {
22258 d = NULL;
22259 nd = 0;
22260 }
22261 #if PY_MAJOR_VERSION >= 3
22262 result = PyEval_EvalCodeEx((PyObject*)co, globals, (PyObject *)NULL,
22263 args, (int)nargs,
22264 k, (int)nk,
22265 d, (int)nd, kwdefs, closure);
22266 #else
22267 result = PyEval_EvalCodeEx(co, globals, (PyObject *)NULL,
22268 args, (int)nargs,
22269 k, (int)nk,
22270 d, (int)nd, closure);
22271 #endif
22272 Py_XDECREF(kwtuple);
22273 done:
22274 Py_LeaveRecursiveCall();
22275 return result;
22276 }
22277 #endif
22278 #endif
22279
22280 /* PyObjectCall */
22281 #if CYTHON_COMPILING_IN_CPYTHON
22282 static CYTHON_INLINE PyObject* __Pyx_PyObject_Call(PyObject *func, PyObject *arg, PyObject *kw) {
22283 PyObject *result;
22284 ternaryfunc call = Py_TYPE(func)->tp_call;
22285 if (unlikely(!call))
22286 return PyObject_Call(func, arg, kw);
22287 if (unlikely(Py_EnterRecursiveCall((char*)" while calling a Python object")))
22288 return NULL;
22289 result = (*call)(func, arg, kw);
22290 Py_LeaveRecursiveCall();
22291 if (unlikely(!result) && unlikely(!PyErr_Occurred())) {
22292 PyErr_SetString(
22293 PyExc_SystemError,
22294 "NULL result without error in PyObject_Call");
22295 }
22296 return result;
22297 }
22298 #endif
22299
22300 /* PyObjectCallMethO */
22301 #if CYTHON_COMPILING_IN_CPYTHON
22302 static CYTHON_INLINE PyObject* __Pyx_PyObject_CallMethO(PyObject *func, PyObject *arg) {
22303 PyObject *self, *result;
22304 PyCFunction cfunc;
22305 cfunc = PyCFunction_GET_FUNCTION(func);
22306 self = PyCFunction_GET_SELF(func);
22307 if (unlikely(Py_EnterRecursiveCall((char*)" while calling a Python object")))
22308 return NULL;
22309 result = cfunc(self, arg);
22310 Py_LeaveRecursiveCall();
22311 if (unlikely(!result) && unlikely(!PyErr_Occurred())) {
22312 PyErr_SetString(
22313 PyExc_SystemError,
22314 "NULL result without error in PyObject_Call");
22315 }
22316 return result;
22317 }
22318 #endif
22319
22320 /* PyObjectCallOneArg */
22321 #if CYTHON_COMPILING_IN_CPYTHON
22322 static PyObject* __Pyx_PyObject_CallOneArg(PyObject *func, PyObject *arg) {
22323 PyObject *result;
22324 PyObject *args = PyTuple_New(1);
22325 if (unlikely(!args)) return NULL;
22326 Py_INCREF(arg);
22327 PyTuple_SET_ITEM(args, 0, arg);
22328 result = __Pyx_PyObject_Call(func, args, NULL);
22329 Py_DECREF(args);
22330 return result;
22331 }
22332 static CYTHON_INLINE PyObject* __Pyx_PyObject_CallOneArg(PyObject *func, PyObject *arg) {
22333 #if CYTHON_FAST_PYCALL
22334 if (PyFunction_Check(func)) {
22335 return __Pyx_PyFunction_FastCall(func, &arg, 1);
22336 }
22337 #endif
22338 if (likely(PyCFunction_Check(func))) {
22339 if (likely(PyCFunction_GET_FLAGS(func) & METH_O)) {
22340 return __Pyx_PyObject_CallMethO(func, arg);
22341 }
22342 }
22343 #if CYTHON_FAST_PYCALL

```

```

22342 } else if (__Pyx_PyFastCFunction_Check(func)) {
22343 return __Pyx_PyCFunction_FastCall(func, &arg, 1);
22344 #endif
22345 }
22346 }
22347 return __Pyx_PyObject_CallOneArg(func, arg);
22348 }
22349 #else
22350 static CYTHON_INLINE PyObject* __Pyx_PyObject_CallOneArg(PyObject *func, PyObject *arg) {
22351 PyObject *result;
22352 PyObject *args = PyTuple_Pack(1, arg);
22353 if (unlikely(!args)) return NULL;
22354 result = __Pyx_PyObject_Call(func, args, NULL);
22355 Py_DECREF(args);
22356 return result;
22357 }
22358 #endif
22359
22360 /* PyErrFetchRestore */
22361 #if CYTHON_FAST_THREAD_STATE
22362 static CYTHON_INLINE void __Pyx_ErrRestoreInState(PyThreadState *tstate, PyObject *type, PyObject
22363 *value, PyObject *tb) {
22364 PyObject *tmp_type, *tmp_value, *tmp_tb;
22365 tmp_type = tstate->curexc_type;
22366 tmp_value = tstate->curexc_value;
22367 tmp_tb = tstate->curexc_traceback;
22368 tstate->curexc_type = type;
22369 tstate->curexc_value = value;
22370 tstate->curexc_traceback = tb;
22371 Py_XDECREF(tmp_type);
22372 Py_XDECREF(tmp_value);
22373 Py_XDECREF(tmp_tb);
22374 }
22375 static CYTHON_INLINE void __Pyx_ErrFetchInState(PyThreadState *tstate, PyObject **type, PyObject
22376 **value, PyObject **tb) {
22377 *type = tstate->curexc_type;
22378 *value = tstate->curexc_value;
22379 *tb = tstate->curexc_traceback;
22380 tstate->curexc_type = 0;
22381 tstate->curexc_value = 0;
22382 tstate->curexc_traceback = 0;
22383 }
22384 #endif
22385
22386 /* WriteUnraisableException */
22387 static void __Pyx_WriteUnraisable(const char *name, CYTHON_UNUSED int clineno,
22388 CYTHON_UNUSED int lineno, CYTHON_UNUSED const char *filename,
22389 int full_traceback, CYTHON_UNUSED int nogil) {
22390 PyObject *old_exc, *old_val, *old_tb;
22391 PyObject *ctx;
22392 __Pyx_PyThreadState_declare
22393 #ifdef WITH_THREAD
22394 PyGILState_STATE state;
22395 if (nogil)
22396 state = PyGILState_Ensure();
22397 #endif
22398 #ifdef _MSC_VER
22399 else state = (PyGILState_STATE)-1;
22400 #endif
22401 #endif
22402 __Pyx_PyThreadState_assign
22403 __Pyx_ErrFetch(&old_exc, &old_val, &old_tb);
22404 if (full_traceback) {
22405 Py_XINCREf(old_exc);
22406 Py_XINCREf(old_val);
22407 Py_XINCREf(old_tb);
22408 __Pyx_ErrRestore(old_exc, old_val, old_tb);
22409 PyErr_PrintEx(1);
22410 }
22411 #if PY_MAJOR_VERSION < 3
22412 ctx = PyString_FromString(name);
22413 #else
22414 ctx = PyUnicode_FromString(name);
22415 #endif
22416 __Pyx_ErrRestore(old_exc, old_val, old_tb);
22417 if (!ctx) {
22418 PyErr_WriteUnraisable(Py_None);
22419 } else {
22420 PyErr_WriteUnraisable(ctx);
22421 Py_DECREF(ctx);
22422 }
22423 #ifdef WITH_THREAD
22424 if (nogil)
22425 PyGILState_Release(state);
22426 #endif
22427 }
22428
22429 /* PyDictVersioning */

```



```

22427 #if CYTHON_USE_DICT_VERSIONS && CYTHON_USE_TYPE_SLOTS
22428 static CYTHON_INLINE PY_UINT64_T __Pyx_get_tp_dict_version(PyObject *obj) {
22429 PyObject *dict = Py_TYPE(obj)->tp_dict;
22430 return likely(dict) ? __PYX_GET_DICT_VERSION(dict) : 0;
22431 }
22432 static CYTHON_INLINE PY_UINT64_T __Pyx_get_object_dict_version(PyObject *obj) {
22433 PyObject **dictptr = NULL;
22434 Py_ssize_t offset = Py_TYPE(obj)->tp_dictoffset;
22435 if (offset) {
22436 #if CYTHON_COMPILING_IN_CPYTHON
22437 dictptr = (likely(offset > 0)) ? (PyObject **) ((char *)obj + offset) :
 _PyObject_GetDictPtr(obj);
22438 #else
22439 dictptr = _PyObject_GetDictPtr(obj);
22440 #endif
22441 }
22442 return (dictptr && *dictptr) ? __PYX_GET_DICT_VERSION(*dictptr) : 0;
22443 }
22444 static CYTHON_INLINE int __Pyx_object_dict_version_matches(PyObject* obj, PY_UINT64_T tp_dict_version,
PY_UINT64_T obj_dict_version) {
22445 PyObject *dict = Py_TYPE(obj)->tp_dict;
22446 if (unlikely(!dict) || unlikely(tp_dict_version != __PYX_GET_DICT_VERSION(dict)))
22447 return 0;
22448 return obj_dict_version == __Pyx_get_object_dict_version(obj);
22449 }
22450 #endif
22451
22452 /* PyObjectCallNoArg */
22453 #if CYTHON_COMPILING_IN_CPYTHON
22454 static CYTHON_INLINE PyObject* __Pyx_PyObject_CallNoArg(PyObject *func) {
22455 #if CYTHON_FAST_PYCALL
22456 if (PyFunction_Check(func)) {
22457 return __Pyx_PyFunction_FastCall(func, NULL, 0);
22458 }
22459 #endif
22460 #ifdef __Pyx_CyFunction_USED
22461 if (likely(PyCFunction_Check(func) || __Pyx_CyFunction_Check(func)))
22462 #else
22463 if (likely(PyCFunction_Check(func)))
22464 #endif
22465 {
22466 if (likely(PyCFunction_GET_FLAGS(func) & METH_NOARGS)) {
22467 return __Pyx_PyObject_CallMethO(func, NULL);
22468 }
22469 }
22470 return __Pyx_PyObject_Call(func, __pyx_empty_tuple, NULL);
22471 }
22472 #endif
22473
22474 /* RaiseDoubleKeywords */
22475 static void __Pyx_RaiseDoubleKeywordsError(
22476 const char* func_name,
22477 PyObject* kw_name)
22478 {
22479 PyErr_Format(PyExc_TypeError,
22480 #if PY_MAJOR_VERSION >= 3
22481 "%s() got multiple values for keyword argument '%U'", func_name, kw_name);
22482 #else
22483 "%s() got multiple values for keyword argument '%s'", func_name,
22484 PyString_AsString(kw_name));
22485 #endif
22486 }
22487
22488 /* ParseKeywords */
22489 static int __Pyx_ParseOptionalKeywords(
22490 PyObject *kwds,
22491 PyObject **argnames[],
22492 PyObject *kwds2,
22493 PyObject *values[],
22494 Py_ssize_t num_pos_args,
22495 const char* function_name)
22496 {
22497 PyObject *key = 0, *value = 0;
22498 Py_ssize_t pos = 0;
22499 PyObject*** name;
22500 PyObject*** first_kw_arg = argnames + num_pos_args;
22501 while (PyDict_Next(kwds, &pos, &key, &value)) {
22502 name = first_kw_arg;
22503 while (*name && (**name != key)) name++;
22504 if (*name) {
22505 values[name-argnames] = value;
22506 continue;
22507 }
22508 name = first_kw_arg;
22509 #if PY_MAJOR_VERSION < 3
22510 if (likely(PyString_Check(key))) {
22511 while (*name) {

```



```

22512 if ((CYTHON_COMPILING_IN_PYPY || PyString_GET_SIZE(**name) == PyString_GET_SIZE(key))
22513 && _PyString_Eq(**name, key)) {
22514 values[name-argnames] = value;
22515 break;
22516 }
22517 name++;
22518 }
22519 if (*name) continue;
22520 else {
22521 PyObject*** argname = argnames;
22522 while (argname != first_kw_arg) {
22523 if (**argname == key) || (
22524 (CYTHON_COMPILING_IN_PYPY || PyString_GET_SIZE(**argname) ==
PyString_GET_SIZE(key))
22525 && _PyString_Eq(**argname, key))) {
22526 goto arg_passed_twice;
22527 }
22528 argname++;
22529 }
22530 }
22531 } else
22532 #endif
22533 if (likely(PyUnicode_Check(key))) {
22534 while (*name) {
22535 int cmp = (**name == key) ? 0 :
22536 #if !CYTHON_COMPILING_IN_PYPY && PY_MAJOR_VERSION >= 3
22537 (__Pyx_PyUnicode_GET_LENGTH(**name) != __Pyx_PyUnicode_GET_LENGTH(key)) ? 1 :
22538 #endif
22539 PyUnicode_Compare(**name, key);
22540 if (cmp < 0 && unlikely(PyErr_Occurred())) goto bad;
22541 if (cmp == 0) {
22542 values[name-argnames] = value;
22543 break;
22544 }
22545 name++;
22546 }
22547 if (*name) continue;
22548 else {
22549 PyObject*** argname = argnames;
22550 while (argname != first_kw_arg) {
22551 int cmp = (**argname == key) ? 0 :
22552 #if !CYTHON_COMPILING_IN_PYPY && PY_MAJOR_VERSION >= 3
22553 (__Pyx_PyUnicode_GET_LENGTH(**argname) != __Pyx_PyUnicode_GET_LENGTH(key)) ? 1
22554 :
22555 #endif
22556 PyUnicode_Compare(**argname, key);
22557 if (cmp < 0 && unlikely(PyErr_Occurred())) goto bad;
22558 if (cmp == 0) goto arg_passed_twice;
22559 argname++;
22560 }
22561 } else
22562 goto invalid_keyword_type;
22563 if (kwds2) {
22564 if (unlikely(PyDict_SetItem(kwds2, key, value))) goto bad;
22565 } else {
22566 goto invalid_keyword;
22567 }
22568 }
22569 return 0;
22570 arg_passed_twice:
22571 __Pyx_RaiseDoubleKeywordsError(function_name, key);
22572 goto bad;
22573 invalid_keyword_type:
22574 PyErr_Format(PyExc_TypeError,
22575 "%.200s() keywords must be strings", function_name);
22576 goto bad;
22577 invalid_keyword:
22578 PyErr_Format(PyExc_TypeError,
22579 #if PY_MAJOR_VERSION < 3
22580 "%.200s() got an unexpected keyword argument '%.200s'",
22581 function_name, PyString_AsString(key));
22582 #else
22583 "%s() got an unexpected keyword argument '%U'",
22584 function_name, key);
22585 #endif
22586 bad:
22587 return -1;
22588 }
22589
22590 /* RaiseArgTupleInvalid */
22591 static void __Pyx_RaiseArgtupleInvalid(
22592 const char* func_name,
22593 int exact,
22594 Py_ssize_t num_min,
22595 Py_ssize_t num_max,
22596 Py_ssize_t num_found)

```

```

22597 {
22598 Py_ssize_t num_expected;
22599 const char *more_or_less;
22600 if (num_found < num_min) {
22601 num_expected = num_min;
22602 more_or_less = "at least";
22603 } else {
22604 num_expected = num_max;
22605 more_or_less = "at most";
22606 }
22607 if (exact) {
22608 more_or_less = "exactly";
22609 }
22610 PyErr_Format(PyExc_TypeError,
22611 "%.200s() takes %.8s %" CYTHON_FORMAT_SSIZE_T "d positional argument%.1s (%"
CYTHON_FORMAT_SSIZE_T "d given)",
22612 func_name, more_or_less, num_expected,
22613 (num_expected == 1) ? "" : "s", num_found);
22614 }
22615
22616 /* GetModuleGlobalName */
22617 #if CYTHON_USE_DICT_VERSIONS
22618 static PyObject *__Pyx_GetModuleGlobalName(PyObject *name, PY_UINT64_T *dict_version, PyObject
**dict_cached_value)
22619 #else
22620 static CYTHON_INLINE PyObject *__Pyx_GetModuleGlobalName(PyObject *name)
22621 #endif
22622 {
22623 PyObject *result;
22624 #if !CYTHON_AVOID_BORROWED_REFS
22625 #if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x030500A1
22626 result = _PyDict_GetItem_KnownHash(__pyx_d, name, ((PyASCIIObject *) name)->hash);
22627 __PYX_UPDATE_DICT_CACHE(__pyx_d, result, *dict_cached_value, *dict_version)
22628 if (likely(result)) {
22629 return __Pyx_NewRef(result);
22630 } else if (unlikely(PyErr_Occurred())) {
22631 return NULL;
22632 }
22633 #else
22634 result = PyDict_GetItem(__pyx_d, name);
22635 __PYX_UPDATE_DICT_CACHE(__pyx_d, result, *dict_cached_value, *dict_version)
22636 if (likely(result)) {
22637 return __Pyx_NewRef(result);
22638 }
22639 #endif
22640 #else
22641 result = PyObject_GetItem(__pyx_d, name);
22642 __PYX_UPDATE_DICT_CACHE(__pyx_d, result, *dict_cached_value, *dict_version)
22643 if (likely(result)) {
22644 return __Pyx_NewRef(result);
22645 }
22646 PyErr_Clear();
22647 #endif
22648 return __Pyx_GetBuiltinName(name);
22649 }
22650
22651 /* GetTopmostException */
22652 #if CYTHON_USE_EXC_INFO_STACK
22653 static _PyErr_StackItem *
22654 __Pyx_PyErr_GetTopmostException(PyThreadState *tstate)
22655 {
22656 _PyErr_StackItem *exc_info = tstate->exc_info;
22657 while ((exc_info->exc_type == NULL || exc_info->exc_type == Py_None) &&
22658 exc_info->previous_item != NULL)
22659 {
22660 exc_info = exc_info->previous_item;
22661 }
22662 return exc_info;
22663 }
22664 #endif
22665
22666 /* SaveResetException */
22667 #if CYTHON_FAST_THREAD_STATE
22668 static CYTHON_INLINE void __Pyx_ExceptionSave(PyThreadState *tstate, PyObject **type, PyObject
**value, PyObject **tb) {
22669 #if CYTHON_USE_EXC_INFO_STACK
22670 _PyErr_StackItem *exc_info = __Pyx_PyErr_GetTopmostException(tstate);
22671 *type = exc_info->exc_type;
22672 *value = exc_info->exc_value;
22673 *tb = exc_info->exc_traceback;
22674 #else
22675 *type = tstate->exc_type;
22676 *value = tstate->exc_value;
22677 *tb = tstate->exc_traceback;
22678 #endif
22679 Py_XINCREF(*type);
22680 Py_XINCREF(*value);

```

```

22681 Py_XINCREf(*tb);
22682 }
22683 static CYTHON_INLINE void __Pyx__ExceptionReset(PyThreadState *tstate, PyObject *type, PyObject
*value, PyObject *tb) {
22684 PyObject *tmp_type, *tmp_value, *tmp_tb;
22685 #if CYTHON_USE_EXC_INFO_STACK
22686 _PyErr_StackItem *exc_info = tstate->exc_info;
22687 tmp_type = exc_info->exc_type;
22688 tmp_value = exc_info->exc_value;
22689 tmp_tb = exc_info->exc_traceback;
22690 exc_info->exc_type = type;
22691 exc_info->exc_value = value;
22692 exc_info->exc_traceback = tb;
22693 #else
22694 tmp_type = tstate->exc_type;
22695 tmp_value = tstate->exc_value;
22696 tmp_tb = tstate->exc_traceback;
22697 tstate->exc_type = type;
22698 tstate->exc_value = value;
22699 tstate->exc_traceback = tb;
22700 #endif
22701 Py_XDECREF(tmp_type);
22702 Py_XDECREF(tmp_value);
22703 Py_XDECREF(tmp_tb);
22704 }
22705 #endif
22706
22707 /* PyErrExceptionMatches */
22708 #if CYTHON_FAST_THREAD_STATE
22709 static int __Pyx_PyErr_ExceptionMatchesTuple(PyObject *exc_type, PyObject *tuple) {
22710 Py_ssize_t i, n;
22711 n = PyTuple_GET_SIZE(tuple);
22712 #if PY_MAJOR_VERSION >= 3
22713 for (i=0; i<n; i++) {
22714 if (exc_type == PyTuple_GET_ITEM(tuple, i)) return 1;
22715 }
22716 #endif
22717 for (i=0; i<n; i++) {
22718 if (__Pyx_PyErr_GivenExceptionMatches(exc_type, PyTuple_GET_ITEM(tuple, i))) return 1;
22719 }
22720 return 0;
22721 }
22722 static CYTHON_INLINE int __Pyx_PyErr_ExceptionMatchesInState(PyThreadState* tstate, PyObject* err) {
22723 PyObject *exc_type = tstate->curexc_type;
22724 if (exc_type == err) return 1;
22725 if (unlikely(!exc_type)) return 0;
22726 if (unlikely(PyTuple_Check(err)))
22727 return __Pyx_PyErr_ExceptionMatchesTuple(exc_type, err);
22728 return __Pyx_PyErr_GivenExceptionMatches(exc_type, err);
22729 }
22730 #endif
22731
22732 /* GetException */
22733 #if CYTHON_FAST_THREAD_STATE
22734 static int __Pyx_GetException(PyThreadState *tstate, PyObject **type, PyObject **value, PyObject
**tb)
22735 #else
22736 static int __Pyx_GetException(PyObject **type, PyObject **value, PyObject **tb)
22737 #endif
22738 {
22739 PyObject *local_type, *local_value, *local_tb;
22740 #if CYTHON_FAST_THREAD_STATE
22741 PyObject *tmp_type, *tmp_value, *tmp_tb;
22742 local_type = tstate->curexc_type;
22743 local_value = tstate->curexc_value;
22744 local_tb = tstate->curexc_traceback;
22745 tstate->curexc_type = 0;
22746 tstate->curexc_value = 0;
22747 tstate->curexc_traceback = 0;
22748 #else
22749 PyErr_Fetch(&local_type, &local_value, &local_tb);
22750 #endif
22751 PyErr_NormalizeException(&local_type, &local_value, &local_tb);
22752 #if CYTHON_FAST_THREAD_STATE
22753 if (unlikely(tstate->curexc_type))
22754 #else
22755 if (unlikely(PyErr_Occurred()))
22756 #endif
22757 goto bad;
22758 #if PY_MAJOR_VERSION >= 3
22759 if (local_tb) {
22760 if (unlikely(PyException_SetTraceback(local_value, local_tb) < 0))
22761 goto bad;
22762 }
22763 #endif
22764 Py_XINCREf(local_tb);
22765 Py_XINCREf(local_type);

```

```

22766 Py_XINCREf(local_value);
22767 *type = local_type;
22768 *value = local_value;
22769 *tb = local_tb;
22770 #if CYTHON_FAST_THREAD_STATE
22771 #if CYTHON_USE_EXC_INFO_STACK
22772 {
22773 _PyErr_StackItem *exc_info = tstate->exc_info;
22774 tmp_type = exc_info->exc_type;
22775 tmp_value = exc_info->exc_value;
22776 tmp_tb = exc_info->exc_traceback;
22777 exc_info->exc_type = local_type;
22778 exc_info->exc_value = local_value;
22779 exc_info->exc_traceback = local_tb;
22780 }
22781 #else
22782 tmp_type = tstate->exc_type;
22783 tmp_value = tstate->exc_value;
22784 tmp_tb = tstate->exc_traceback;
22785 tstate->exc_type = local_type;
22786 tstate->exc_value = local_value;
22787 tstate->exc_traceback = local_tb;
22788 #endif
22789 Py_XDECREF(tmp_type);
22790 Py_XDECREF(tmp_value);
22791 Py_XDECREF(tmp_tb);
22792 #else
22793 PyErr_SetExcInfo(local_type, local_value, local_tb);
22794 #endif
22795 return 0;
22796 bad:
22797 *type = 0;
22798 *value = 0;
22799 *tb = 0;
22800 Py_XDECREF(local_type);
22801 Py_XDECREF(local_value);
22802 Py_XDECREF(local_tb);
22803 return -1;
22804 }
22805
22806 /* RaiseException */
22807 #if PY_MAJOR_VERSION < 3
22808 static void __Pyx_Raise(PyObject *type, PyObject *value, PyObject *tb,
22809 CYTHON_UNUSED PyObject *cause) {
22810 __Pyx_PyThreadState_declare
22811 Py_XINCREf(type);
22812 if (!value || value == Py_None)
22813 value = NULL;
22814 else
22815 Py_INCREF(value);
22816 if (!tb || tb == Py_None)
22817 tb = NULL;
22818 else {
22819 Py_INCREF(tb);
22820 if (!PyTraceBack_Check(tb)) {
22821 PyErr_SetString(PyExc_TypeError,
22822 "raise: arg 3 must be a traceback or None");
22823 goto raise_error;
22824 }
22825 }
22826 if (PyType_Check(type)) {
22827 #if CYTHON_COMPILING_IN_PYPY
22828 if (!value) {
22829 Py_INCREF(Py_None);
22830 value = Py_None;
22831 }
22832 #endif
22833 PyErr_NormalizeException(&type, &value, &tb);
22834 } else {
22835 if (value) {
22836 PyErr_SetString(PyExc_TypeError,
22837 "instance exception may not have a separate value");
22838 goto raise_error;
22839 }
22840 value = type;
22841 type = (PyObject*) Py_TYPE(type);
22842 Py_INCREF(type);
22843 if (!PyType_IsSubtype((PyTypeObject *)type, (PyTypeObject *)PyExc_BaseException)) {
22844 PyErr_SetString(PyExc_TypeError,
22845 "raise: exception class must be a subclass of BaseException");
22846 goto raise_error;
22847 }
22848 }
22849 __Pyx_PyThreadState_assign
22850 __Pyx_ErrRestore(type, value, tb);
22851 return;
22852 raise_error:

```

```

22853 Py_XDECREF(value);
22854 Py_XDECREF(type);
22855 Py_XDECREF(tb);
22856 return;
22857 }
22858 #else
22859 static void __Pyx_Raise(PyObject *type, PyObject *value, PyObject *tb, PyObject *cause) {
22860 PyObject* owned_instance = NULL;
22861 if (tb == Py_None) {
22862 tb = 0;
22863 } else if (tb && !PyTraceBack_Check(tb)) {
22864 PyErr_SetString(PyExc_TypeError,
22865 "raise: arg 3 must be a traceback or None");
22866 goto bad;
22867 }
22868 if (value == Py_None)
22869 value = 0;
22870 if (PyExceptionInstance_Check(type)) {
22871 if (value) {
22872 PyErr_SetString(PyExc_TypeError,
22873 "instance exception may not have a separate value");
22874 goto bad;
22875 }
22876 value = type;
22877 type = (PyObject*) Py_TYPE(value);
22878 } else if (PyExceptionClass_Check(type)) {
22879 PyObject *instance_class = NULL;
22880 if (value && PyExceptionInstance_Check(value)) {
22881 instance_class = (PyObject*) Py_TYPE(value);
22882 if (instance_class != type) {
22883 int is_subclass = PyObject_IsSubclass(instance_class, type);
22884 if (!is_subclass) {
22885 instance_class = NULL;
22886 } else if (unlikely(is_subclass == -1)) {
22887 goto bad;
22888 } else {
22889 type = instance_class;
22890 }
22891 }
22892 }
22893 if (!instance_class) {
22894 PyObject *args;
22895 if (!value)
22896 args = PyTuple_New(0);
22897 else if (PyTuple_Check(value)) {
22898 Py_INCREF(value);
22899 args = value;
22900 } else
22901 args = PyTuple_Pack(1, value);
22902 if (!args)
22903 goto bad;
22904 owned_instance = PyObject_Call(type, args, NULL);
22905 Py_DECREF(args);
22906 if (!owned_instance)
22907 goto bad;
22908 value = owned_instance;
22909 if (!PyExceptionInstance_Check(value)) {
22910 PyErr_Format(PyExc_TypeError,
22911 "calling %R should have returned an instance of "
22912 "BaseException, not %R",
22913 type, Py_TYPE(value));
22914 goto bad;
22915 }
22916 }
22917 } else {
22918 PyErr_SetString(PyExc_TypeError,
22919 "raise: exception class must be a subclass of BaseException");
22920 goto bad;
22921 }
22922 if (cause) {
22923 PyObject *fixed_cause;
22924 if (cause == Py_None) {
22925 fixed_cause = NULL;
22926 } else if (PyExceptionClass_Check(cause)) {
22927 fixed_cause = PyObject_CallObject(cause, NULL);
22928 if (fixed_cause == NULL)
22929 goto bad;
22930 } else if (PyExceptionInstance_Check(cause)) {
22931 fixed_cause = cause;
22932 Py_INCREF(fixed_cause);
22933 } else {
22934 PyErr_SetString(PyExc_TypeError,
22935 "exception causes must derive from "
22936 "BaseException");
22937 goto bad;
22938 }
22939 PyException_SetCause(value, fixed_cause);

```

```

22940 }
22941 PyErr_SetObject(type, value);
22942 if (tb) {
22943 #if CYTHON_COMPILING_IN_PYPY
22944 PyObject *tmp_type, *tmp_value, *tmp_tb;
22945 PyErr_Fetch(&tmp_type, &tmp_value, &tmp_tb);
22946 Py_INCREF(tb);
22947 PyErr_Restore(tmp_type, tmp_value, tb);
22948 Py_XDECREF(tmp_tb);
22949 #else
22950 PyThreadState *tstate = __Pyx_PyThreadState_Current;
22951 PyObject* tmp_tb = tstate->curexc_traceback;
22952 if (tb != tmp_tb) {
22953 Py_INCREF(tb);
22954 tstate->curexc_traceback = tb;
22955 Py_XDECREF(tmp_tb);
22956 }
22957 #endif
22958 }
22959 bad:
22960 Py_XDECREF(owned_instance);
22961 return;
22962 }
22963 #endif
22964
22965 /* PyObjectCall2Args */
22966 static CYTHON_UNUSED PyObject* __Pyx_PyObject_Call2Args(PyObject* function, PyObject* arg1, PyObject*
arg2) {
22967 PyObject *args, *result = NULL;
22968 #if CYTHON_FAST_PYCALL
22969 if (PyFunction_Check(function)) {
22970 PyObject *args[2] = {arg1, arg2};
22971 return __Pyx_PyFunction_FastCall(function, args, 2);
22972 }
22973 #endif
22974 #if CYTHON_FAST_PYCCALL
22975 if (__Pyx_PyFastCFunction_Check(function)) {
22976 PyObject *args[2] = {arg1, arg2};
22977 return __Pyx_PyCFunction_FastCall(function, args, 2);
22978 }
22979 #endif
22980 args = PyTuple_New(2);
22981 if (unlikely(!args)) goto done;
22982 Py_INCREF(arg1);
22983 PyTuple_SET_ITEM(args, 0, arg1);
22984 Py_INCREF(arg2);
22985 PyTuple_SET_ITEM(args, 1, arg2);
22986 Py_INCREF(function);
22987 result = __Pyx_PyObject_Call(function, args, NULL);
22988 Py_DECREF(args);
22989 Py_DECREF(function);
22990 done:
22991 return result;
22992 }
22993
22994 /* PyIntBinop */
22995 #if !CYTHON_COMPILING_IN_PYPY
22996 static PyObject* __Pyx_PyInt_AddObjC(PyObject *op1, PyObject *op2, CYTHON_UNUSED long intval, int
inplace, int zerodivision_check) {
22997 (void)inplace;
22998 (void)zerodivision_check;
22999 #if PY_MAJOR_VERSION < 3
23000 if (likely(PyInt_CheckExact(op1))) {
23001 const long b = intval;
23002 long x;
23003 long a = PyInt_AS_LONG(op1);
23004 x = (long)((unsigned long)a + b);
23005 if (likely((x^a) >= 0 || (x^b) >= 0))
23006 return PyInt_FromLong(x);
23007 return PyLong_Type.tp_as_number->nb_add(op1, op2);
23008 }
23009 #endif
23010 #if CYTHON_USE_PYLONG_INTERNALS
23011 if (likely(PyLong_CheckExact(op1))) {
23012 const long b = intval;
23013 long a, x;
23014 #ifdef HAVE_LONG_LONG
23015 const PY_LONG_LONG llb = intval;
23016 PY_LONG_LONG lla, llx;
23017 #endif
23018 const digit* digits = ((PyLongObject*)op1)->ob_digit;
23019 const Py_ssize_t size = Py_SIZE(op1);
23020 if (likely(__Pyx_sst_abs(size) <= 1)) {
23021 a = likely(size) ? digits[0] : 0;
23022 if (size == -1) a = -a;
23023 } else {
23024 switch (size) {

```

```

23025 case -2:
23026 if (8 * sizeof(long) - 1 > 2 * PyLong_SHIFT) {
23027 a = -(long) (((((unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned
long)digits[0]));
23028 break;
23029 #ifdef HAVE_LONG_LONG
23030 } else if (8 * sizeof(PY_LONG_LONG) - 1 > 2 * PyLong_SHIFT) {
23031 lla = -(PY_LONG_LONG) (((((unsigned PY_LONG_LONG)digits[1]) « PyLong_SHIFT) |
(unsigned PY_LONG_LONG)digits[0]));
23032 goto long_long;
23033 #endif
23034 }
23035 CYTHON_FALLTHROUGH;
23036 case 2:
23037 if (8 * sizeof(long) - 1 > 2 * PyLong_SHIFT) {
23038 a = (long) (((((unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned
long)digits[0]));
23039 break;
23040 #ifdef HAVE_LONG_LONG
23041 } else if (8 * sizeof(PY_LONG_LONG) - 1 > 2 * PyLong_SHIFT) {
23042 lla = (PY_LONG_LONG) (((((unsigned PY_LONG_LONG)digits[1]) « PyLong_SHIFT) |
(unsigned PY_LONG_LONG)digits[0]));
23043 goto long_long;
23044 #endif
23045 }
23046 CYTHON_FALLTHROUGH;
23047 case -3:
23048 if (8 * sizeof(long) - 1 > 3 * PyLong_SHIFT) {
23049 a = -(long) ((((((unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned
long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0]));
23050 break;
23051 #ifdef HAVE_LONG_LONG
23052 } else if (8 * sizeof(PY_LONG_LONG) - 1 > 3 * PyLong_SHIFT) {
23053 lla = -(PY_LONG_LONG) ((((((unsigned PY_LONG_LONG)digits[2]) « PyLong_SHIFT)
| (unsigned PY_LONG_LONG)digits[1]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[0]));
23054 goto long_long;
23055 #endif
23056 }
23057 CYTHON_FALLTHROUGH;
23058 case 3:
23059 if (8 * sizeof(long) - 1 > 3 * PyLong_SHIFT) {
23060 a = (long) ((((((unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned
long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0]));
23061 break;
23062 #ifdef HAVE_LONG_LONG
23063 } else if (8 * sizeof(PY_LONG_LONG) - 1 > 3 * PyLong_SHIFT) {
23064 lla = (PY_LONG_LONG) ((((((unsigned PY_LONG_LONG)digits[2]) « PyLong_SHIFT)
| (unsigned PY_LONG_LONG)digits[1]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[0]));
23065 goto long_long;
23066 #endif
23067 }
23068 CYTHON_FALLTHROUGH;
23069 case -4:
23070 if (8 * sizeof(long) - 1 > 4 * PyLong_SHIFT) {
23071 a = -(long) (((((((((unsigned long)digits[3]) « PyLong_SHIFT) | (unsigned
long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned
long)digits[0]));
23072 break;
23073 #ifdef HAVE_LONG_LONG
23074 } else if (8 * sizeof(PY_LONG_LONG) - 1 > 4 * PyLong_SHIFT) {
23075 lla = -(PY_LONG_LONG) (((((((((unsigned PY_LONG_LONG)digits[3]) «
PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[2]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[1]) «
PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[0]));
23076 goto long_long;
23077 #endif
23078 }
23079 CYTHON_FALLTHROUGH;
23080 case 4:
23081 if (8 * sizeof(long) - 1 > 4 * PyLong_SHIFT) {
23082 a = (long) (((((((((unsigned long)digits[3]) « PyLong_SHIFT) | (unsigned
long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned
long)digits[0]));
23083 break;
23084 #ifdef HAVE_LONG_LONG
23085 } else if (8 * sizeof(PY_LONG_LONG) - 1 > 4 * PyLong_SHIFT) {
23086 lla = (PY_LONG_LONG) (((((((((unsigned PY_LONG_LONG)digits[3]) « PyLong_SHIFT)
| (unsigned PY_LONG_LONG)digits[2]) « PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[1]) «
PyLong_SHIFT) | (unsigned PY_LONG_LONG)digits[0]));
23087 goto long_long;
23088 #endif
23089 }
23090 CYTHON_FALLTHROUGH;
23091 default: return PyLong_Type.tp_as_number->nb_add(op1, op2);
23092 }
23093 }
23094 x = a + b;
23095 return PyLong_FromLong(x);

```

```

23096 #ifdef HAVE_LONG_LONG
23097 long_long:
23098 llx = lla + llb;
23099 return PyLong_FromLongLong(llx);
23100 #endif
23101
23102
23103 }
23104 #endif
23105 if (PyFloat_CheckExact(op1)) {
23106 const long b = intval;
23107 double a = PyFloat_AS_DOUBLE(op1);
23108 double result;
23109 PyFPE_START_PROTECT("add", return NULL)
23110 result = ((double)a) + (double)b;
23111 PyFPE_END_PROTECT(result)
23112 return PyFloat_FromDouble(result);
23113 }
23114 return (inplace ? PyNumber_InPlaceAdd : PyNumber_Add)(op1, op2);
23115 }
23116 #endif
23117
23118 /* decode_c_bytes */
23119 static CYTHON_INLINE PyObject* __Pyx_decode_c_bytes(
23120 const char* cstring, Py_ssize_t length, Py_ssize_t start, Py_ssize_t stop,
23121 const char* encoding, const char* errors,
23122 PyObject* (*decode_func)(const char *s, Py_ssize_t size, const char *errors)) {
23123 if (unlikely((start < 0) | (stop < 0))) {
23124 if (start < 0) {
23125 start += length;
23126 if (start < 0)
23127 start = 0;
23128 }
23129 if (stop < 0)
23130 stop += length;
23131 }
23132 if (stop > length)
23133 stop = length;
23134 if (unlikely(stop <= start))
23135 return __Pyx_NewRef(__pyx_empty_unicode);
23136 length = stop - start;
23137 cstring += start;
23138 if (decode_func) {
23139 return decode_func(cstring, length, errors);
23140 } else {
23141 return PyUnicode_Decode(cstring, length, encoding, errors);
23142 }
23143 }
23144
23145 /* SwapException */
23146 #if CYTHON_FAST_THREAD_STATE
23147 static CYTHON_INLINE void __Pyx_ExceptionSwap(PyThreadState *tstate, PyObject **type, PyObject
**value, PyObject **tb) {
23148 PyObject *tmp_type, *tmp_value, *tmp_tb;
23149 #if CYTHON_USE_EXC_INFO_STACK
23150 _PyErr_StackItem *exc_info = tstate->exc_info;
23151 tmp_type = exc_info->exc_type;
23152 tmp_value = exc_info->exc_value;
23153 tmp_tb = exc_info->exc_traceback;
23154 exc_info->exc_type = *type;
23155 exc_info->exc_value = *value;
23156 exc_info->exc_traceback = *tb;
23157 #else
23158 tmp_type = tstate->exc_type;
23159 tmp_value = tstate->exc_value;
23160 tmp_tb = tstate->exc_traceback;
23161 tstate->exc_type = *type;
23162 tstate->exc_value = *value;
23163 tstate->exc_traceback = *tb;
23164 #endif
23165 *type = tmp_type;
23166 *value = tmp_value;
23167 *tb = tmp_tb;
23168 }
23169 #else
23170 static CYTHON_INLINE void __Pyx_ExceptionSwap(PyObject **type, PyObject **value, PyObject **tb) {
23171 PyObject *tmp_type, *tmp_value, *tmp_tb;
23172 PyErr_GetExcInfo(&tmp_type, &tmp_value, &tmp_tb);
23173 PyErr_SetExcInfo(*type, *value, *tb);
23174 *type = tmp_type;
23175 *value = tmp_value;
23176 *tb = tmp_tb;
23177 }
23178 #endif
23179
23180 /* SetItemInt */
23181 static int __Pyx_SetItemInt_Generic(PyObject *o, PyObject *j, PyObject *v) {

```



```

23182 int r;
23183 if (!j) return -1;
23184 r = PyObject_SetItem(o, j, v);
23185 Py_DECREF(j);
23186 return r;
23187 }
23188 static CYTHON_INLINE int __Pyx_SetItemInt_Fast(PyObject *o, Py_ssize_t i, PyObject *v, int is_list,
23189 CYTHON_NCP_UNUSED int wraparound, CYTHON_NCP_UNUSED int
 boundscheck) {
23190 #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS && CYTHON_USE_TYPE_SLOTS
23191 if (is_list || PyList_CheckExact(o)) {
23192 Py_ssize_t n = (!wraparound) ? i : ((likely(i >= 0)) ? i : i + PyList_GET_SIZE(o));
23193 if ((!boundscheck) || likely(__Pyx_is_valid_index(n, PyList_GET_SIZE(o)))) {
23194 PyObject* old = PyList_GET_ITEM(o, n);
23195 Py_INCREF(v);
23196 PyList_SET_ITEM(o, n, v);
23197 Py_DECREF(old);
23198 return 1;
23199 }
23200 } else {
23201 PySequenceMethods *m = Py_TYPE(o)->tp_as_sequence;
23202 if (likely(m && m->sq_ass_item)) {
23203 if (wraparound && unlikely(i < 0) && likely(m->sq_length)) {
23204 Py_ssize_t l = m->sq_length(o);
23205 if (likely(l >= 0)) {
23206 i += l;
23207 } else {
23208 if (!PyErr_ExceptionMatches(PyExc_OverflowError))
23209 return -1;
23210 PyErr_Clear();
23211 }
23212 }
23213 return m->sq_ass_item(o, i, v);
23214 }
23215 }
23216 #else
23217 #if CYTHON_COMPILING_IN_PYPY
23218 if (is_list || (PySequence_Check(o) && !PyDict_Check(o)))
23219 #else
23220 if (is_list || PySequence_Check(o))
23221 #endif
23222 {
23223 return PySequence_SetItem(o, i, v);
23224 }
23225 #endif
23226 return __Pyx_SetItemInt_Generic(o, PyInt_FromSsize_t(i), v);
23227 }
23228
23229 /* ArgTypeTest */
23230 static int __Pyx_ArgTypeTest(PyObject *obj, PyTypeObject *type, const char *name, int exact)
23231 {
23232 if (unlikely(!type)) {
23233 PyErr_SetString(PyExc_SystemError, "Missing type object");
23234 return 0;
23235 }
23236 else if (exact) {
23237 #if PY_MAJOR_VERSION == 2
23238 if ((type == &PyBaseString_Type) && likely(__Pyx_PyBaseString_CheckExact(obj))) return 1;
23239 #endif
23240 }
23241 else {
23242 if (likely(__Pyx_TypeCheck(obj, type))) return 1;
23243 }
23244 PyErr_Format(PyExc_TypeError,
23245 "Argument '%.200s' has incorrect type (expected %.200s, got %.200s)",
23246 name, type->tp_name, Py_TYPE(obj)->tp_name);
23247 return 0;
23248 }
23249
23250 /* Import */
23251 static PyObject *__Pyx_Import(PyObject *name, PyObject *from_list, int level) {
23252 PyObject *empty_list = 0;
23253 PyObject *module = 0;
23254 PyObject *global_dict = 0;
23255 PyObject *empty_dict = 0;
23256 PyObject *list;
23257 #if PY_MAJOR_VERSION < 3
23258 PyObject *py_import;
23259 py_import = __Pyx_PyObject_GetAttrStr(__pyx_b, __pyx_n_s_import);
23260 if (!py_import)
23261 goto bad;
23262 #endif
23263 if (from_list)
23264 list = from_list;
23265 else {
23266 empty_list = PyList_New(0);
23267 if (!empty_list)

```

```

23268 goto bad;
23269 list = empty_list;
23270 }
23271 global_dict = PyModule_GetDict(__pyx_m);
23272 if (!global_dict)
23273 goto bad;
23274 empty_dict = PyDict_New();
23275 if (!empty_dict)
23276 goto bad;
23277 {
23278 #if PY_MAJOR_VERSION >= 3
23279 if (level == -1) {
23280 if ((1) && (strchr(__Pyx_MODULE_NAME, '.'))) {
23281 module = PyImport_ImportModuleLevelObject(
23282 name, global_dict, empty_dict, list, 1);
23283 if (!module) {
23284 if (!PyErr_ExceptionMatches(PyExc_ImportError))
23285 goto bad;
23286 PyErr_Clear();
23287 }
23288 }
23289 level = 0;
23290 }
23291 #endif
23292 if (!module) {
23293 #if PY_MAJOR_VERSION < 3
23294 PyObject *py_level = PyInt_FromLong(level);
23295 if (!py_level)
23296 goto bad;
23297 module = PyObject_CallFunctionObjArgs(py_import,
23298 name, global_dict, empty_dict, list, py_level, (PyObject *)NULL);
23299 Py_DECREF(py_level);
23300 #else
23301 module = PyImport_ImportModuleLevelObject(
23302 name, global_dict, empty_dict, list, level);
23303 #endif
23304 }
23305 }
23306 bad:
23307 #if PY_MAJOR_VERSION < 3
23308 Py_XDECREF(py_import);
23309 #endif
23310 Py_XDECREF(empty_list);
23311 Py_XDECREF(empty_dict);
23312 return module;
23313 }
23314
23315 /* PyObject_GenericGetAttrNoDict */
23316 #if CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP && PY_VERSION_HEX < 0x03070000
23317 static PyObject * __Pyx_RaiseGenericGetAttributeError(PyTypeObject *tp, PyObject *attr_name) {
23318 PyErr_Format(PyExc_AttributeError,
23319 #if PY_MAJOR_VERSION >= 3
23320 "%50s' object has no attribute '%U'",
23321 tp->tp_name, attr_name);
23322 #else
23323 "%50s' object has no attribute '%.400s'",
23324 tp->tp_name, PyString_AS_STRING(attr_name));
23325 #endif
23326 return NULL;
23327 }
23328 static CYTHON_INLINE PyObject* __Pyx_PyObject_GenericGetAttrNoDict(PyObject* obj, PyObject* attr_name)
23329 {
23330 PyObject *descr;
23331 PyTypeObject *tp = Py_TYPE(obj);
23332 if (unlikely(!PyString_Check(attr_name))) {
23333 return PyObject_GenericGetAttr(obj, attr_name);
23334 }
23335 assert(!tp->tp_dictoffset);
23336 descr = _PyType_Lookup(tp, attr_name);
23337 if (unlikely(!descr)) {
23338 return __Pyx_RaiseGenericGetAttributeError(tp, attr_name);
23339 }
23340 Py_INCREF(descr);
23341 #if PY_MAJOR_VERSION < 3
23342 if (likely(PyType_HasFeature(Py_TYPE(descr), Py_TPFLAGS_HAVE_CLASS)))
23343 #endif
23344 {
23345 descrgetfunc f = Py_TYPE(descr)->tp_descr_get;
23346 if (unlikely(!f)) {
23347 PyObject *res = f(descr, obj, (PyObject *)tp);
23348 Py_DECREF(descr);
23349 return res;
23350 }
23351 }
23352 return descr;
23353 }
23354 #endif

```

```

23354
23355 /* PyObject_GenericGetAttr */
23356 #if CYTHON_USE_TYPE_SLOTS && CYTHON_USE_PYTYPE_LOOKUP && PY_VERSION_HEX < 0x03070000
23357 static PyObject* __Pyx_PyObject_GenericGetAttr(PyObject* obj, PyObject* attr_name) {
23358 if (unlikely(Py_TYPE(obj)->tp_dictoffset)) {
23359 return PyObject_GenericGetAttr(obj, attr_name);
23360 }
23361 return __Pyx_PyObject_GenericGetAttrNoDict(obj, attr_name);
23362 }
23363 #endif
23364
23365 /* SetVTable */
23366 static int __Pyx_SetVtable(PyObject *dict, void *vtable) {
23367 #if PY_VERSION_HEX >= 0x02070000
23368 PyObject *ob = PyCapsule_New(vtable, 0, 0);
23369 #else
23370 PyObject *ob = PyCObject_FromVoidPtr(vtable, 0);
23371 #endif
23372 if (!ob)
23373 goto bad;
23374 if (PyDict_SetItem(dict, __pyx_n_s_pyx_vtable, ob) < 0)
23375 goto bad;
23376 Py_DECREF(ob);
23377 return 0;
23378 bad:
23379 Py_XDECREF(ob);
23380 return -1;
23381 }
23382
23383 /* PyObjectGetAttrStrNoError */
23384 static void __Pyx_PyObject_GetAttrStr_ClearAttributeError(void) {
23385 __Pyx_PyThreadState_declare
23386 __Pyx_PyThreadState_assign
23387 if (likely(__Pyx_PyErr_ExceptionMatches(PyExc_AttributeError)))
23388 __Pyx_PyErr_Clear();
23389 }
23390 static CYTHON_INLINE PyObject* __Pyx_PyObject_GetAttrStrNoError(PyObject* obj, PyObject* attr_name) {
23391 PyObject *result;
23392 #if CYTHON_COMPILING_IN_CPYTHON && CYTHON_USE_TYPE_SLOTS && PY_VERSION_HEX >= 0x030700B1
23393 PyTypeObject* tp = Py_TYPE(obj);
23394 if (likely(tp->tp_getattro == PyObject_GenericGetAttr)) {
23395 return PyObject_GenericGetAttrWithDict(obj, attr_name, NULL, 1);
23396 }
23397 #endif
23398 result = __Pyx_PyObject_GetAttrStr(obj, attr_name);
23399 if (unlikely(!result)) {
23400 __Pyx_PyObject_GetAttrStr_ClearAttributeError();
23401 }
23402 return result;
23403 }
23404
23405 /* SetupReduce */
23406 static int __Pyx_setup_reduce_is_named(PyObject* meth, PyObject* name) {
23407 int ret;
23408 PyObject *name_attr;
23409 name_attr = __Pyx_PyObject_GetAttrStr(meth, __pyx_n_s_name);
23410 if (likely(name_attr)) {
23411 ret = PyObject_RichCompareBool(name_attr, name, Py_EQ);
23412 } else {
23413 ret = -1;
23414 }
23415 if (unlikely(ret < 0)) {
23416 PyErr_Clear();
23417 ret = 0;
23418 }
23419 Py_XDECREF(name_attr);
23420 return ret;
23421 }
23422 static int __Pyx_setup_reduce(PyObject* type_obj) {
23423 int ret = 0;
23424 PyObject *object_reduce = NULL;
23425 PyObject *object_reduce_ex = NULL;
23426 PyObject *reduce = NULL;
23427 PyObject *reduce_ex = NULL;
23428 PyObject *reduce_cython = NULL;
23429 PyObject *setstate = NULL;
23430 PyObject *setstate_cython = NULL;
23431 #if CYTHON_USE_PYTYPE_LOOKUP
23432 if (_PyType_Lookup((PyTypeObject*)type_obj, __pyx_n_s_getstate)) goto __PYX_GOOD;
23433 #else
23434 if (PyObject_HasAttr(type_obj, __pyx_n_s_getstate)) goto __PYX_GOOD;
23435 #endif
23436 #if CYTHON_USE_PYTYPE_LOOKUP
23437 object_reduce_ex = _PyType_Lookup(&PyBaseObject_Type, __pyx_n_s_reduce_ex); if (!object_reduce_ex)
23438 goto __PYX_BAD;
23439 #else
23439 object_reduce_ex = __Pyx_PyObject_GetAttrStr((PyObject*)&PyBaseObject_Type, __pyx_n_s_reduce_ex);

```

```

 if (!object_reduce_ex) goto __PYX_BAD;
23440 #endif
23441 reduce_ex = __Pyx_PyObject_GetAttrStr(type_obj, __pyx_n_s_reduce_ex); if (unlikely(!reduce_ex))
 goto __PYX_BAD;
23442 if (reduce_ex == object_reduce_ex) {
23443 #if CYTHON_USE_PYTYPE_LOOKUP
23444 object_reduce = _PyType_Lookup(&PyBaseObject_Type, __pyx_n_s_reduce); if (!object_reduce) goto
 __PYX_BAD;
23445 #else
23446 object_reduce = __Pyx_PyObject_GetAttrStr((PyObject*)&PyBaseObject_Type, __pyx_n_s_reduce); if
 (!object_reduce) goto __PYX_BAD;
23447 #endif
23448 reduce = __Pyx_PyObject_GetAttrStr(type_obj, __pyx_n_s_reduce); if (unlikely(!reduce)) goto
 __PYX_BAD;
23449 if (reduce == object_reduce || __Pyx_setup_reduce_is_named(reduce, __pyx_n_s_reduce_cython)) {
23450 reduce_cython = __Pyx_PyObject_GetAttrStrNoError(type_obj, __pyx_n_s_reduce_cython);
23451 if (likely(reduce_cython)) {
23452 ret = PyDict_SetItem(((PyTypeObject*)type_obj)->tp_dict, __pyx_n_s_reduce,
reduce_cython); if (unlikely(ret < 0)) goto __PYX_BAD;
23453 ret = PyDict_DelItem(((PyTypeObject*)type_obj)->tp_dict, __pyx_n_s_reduce_cython); if
 (unlikely(ret < 0)) goto __PYX_BAD;
23454 } else if (reduce == object_reduce || PyErr_Occurred()) {
23455 goto __PYX_BAD;
23456 }
23457 setstate = __Pyx_PyObject_GetAttrStr(type_obj, __pyx_n_s_setstate);
23458 if (!setstate) PyErr_Clear();
23459 if (!setstate || __Pyx_setup_reduce_is_named(setstate, __pyx_n_s_setstate_cython)) {
23460 setstate_cython = __Pyx_PyObject_GetAttrStrNoError(type_obj,
 __pyx_n_s_setstate_cython);
23461 if (likely(setstate_cython)) {
23462 ret = PyDict_SetItem(((PyTypeObject*)type_obj)->tp_dict, __pyx_n_s_setstate,
 setstate_cython); if (unlikely(ret < 0)) goto __PYX_BAD;
23463 ret = PyDict_DelItem(((PyTypeObject*)type_obj)->tp_dict,
 __pyx_n_s_setstate_cython); if (unlikely(ret < 0)) goto __PYX_BAD;
23464 } else if (!setstate || PyErr_Occurred()) {
23465 goto __PYX_BAD;
23466 }
23467 }
23468 PyType_Modified((PyTypeObject*)type_obj);
23469 }
23470 }
23471 goto __PYX_GOOD;
23472 __PYX_BAD:
23473 if (!PyErr_Occurred())
23474 PyErr_Format(PyExc_RuntimeError, "Unable to initialize pickling for %s",
 ((PyTypeObject*)type_obj)->tp_name);
23475 ret = -1;
23476 __PYX_GOOD:
23477 #if !CYTHON_USE_PYTYPE_LOOKUP
23478 Py_XDECREF(object_reduce);
23479 Py_XDECREF(object_reduce_ex);
23480 #endif
23481 Py_XDECREF(reduce);
23482 Py_XDECREF(reduce_ex);
23483 Py_XDECREF(reduce_cython);
23484 Py_XDECREF(setstate);
23485 Py_XDECREF(setstate_cython);
23486 return ret;
23487 }
23488
23489 /* BytesEquals */
23490 static CYTHON_INLINE int __Pyx_PyBytes_Equals(PyObject* s1, PyObject* s2, int equals) {
23491 #if CYTHON_COMPILING_IN_PYPY
23492 return PyObject_RichCompareBool(s1, s2, equals);
23493 #else
23494 if (s1 == s2) {
23495 return (equals == Py_EQ);
23496 } else if (PyBytes_CheckExact(s1) & PyBytes_CheckExact(s2)) {
23497 const char *ps1, *ps2;
23498 Py_ssize_t length = PyBytes_GET_SIZE(s1);
23499 if (length != PyBytes_GET_SIZE(s2))
23500 return (equals == Py_NE);
23501 ps1 = PyBytes_AS_STRING(s1);
23502 ps2 = PyBytes_AS_STRING(s2);
23503 if (ps1[0] != ps2[0]) {
23504 return (equals == Py_NE);
23505 } else if (length == 1) {
23506 return (equals == Py_EQ);
23507 } else {
23508 int result;
23509 #if CYTHON_USE_UNICODE_INTERNALS
23510 Py_hash_t hash1, hash2;
23511 hash1 = ((PyBytesObject*)s1)->ob_shash;
23512 hash2 = ((PyBytesObject*)s2)->ob_shash;
23513 if (hash1 != hash2 && hash1 != -1 && hash2 != -1) {
23514 return (equals == Py_NE);
23515 }

```

```

23516 #endif
23517 result = memcmp(ps1, ps2, (size_t)length);
23518 return (equals == Py_EQ) ? (result == 0) : (result != 0);
23519 }
23520 } else if ((s1 == Py_None) & PyBytes_CheckExact(s2)) {
23521 return (equals == Py_NE);
23522 } else if ((s2 == Py_None) & PyBytes_CheckExact(s1)) {
23523 return (equals == Py_NE);
23524 } else {
23525 int result;
23526 PyObject* py_result = PyObject_RichCompare(s1, s2, equals);
23527 if (!py_result)
23528 return -1;
23529 result = __Pyx_PyObject_IsTrue(py_result);
23530 Py_DECREF(py_result);
23531 return result;
23532 }
23533 #endif
23534 }
23535
23536 /* UnicodeEquals */
23537 static CYTHON_INLINE int __Pyx_PyUnicode_Equals(PyObject* s1, PyObject* s2, int equals) {
23538 #if CYTHON_COMPILING_IN_PYPY
23539 return PyObject_RichCompareBool(s1, s2, equals);
23540 #else
23541 #if PY_MAJOR_VERSION < 3
23542 PyObject* owned_ref = NULL;
23543 #endif
23544 int s1_is_unicode, s2_is_unicode;
23545 if (s1 == s2) {
23546 goto return_eq;
23547 }
23548 s1_is_unicode = PyUnicode_CheckExact(s1);
23549 s2_is_unicode = PyUnicode_CheckExact(s2);
23550 #if PY_MAJOR_VERSION < 3
23551 if ((s1_is_unicode & (!s2_is_unicode)) && PyString_CheckExact(s2)) {
23552 owned_ref = PyUnicode_FromObject(s2);
23553 if (unlikely(!owned_ref))
23554 return -1;
23555 s2 = owned_ref;
23556 s2_is_unicode = 1;
23557 } else if ((s2_is_unicode & (!s1_is_unicode)) && PyString_CheckExact(s1)) {
23558 owned_ref = PyUnicode_FromObject(s1);
23559 if (unlikely(!owned_ref))
23560 return -1;
23561 s1 = owned_ref;
23562 s1_is_unicode = 1;
23563 } else if (((!s2_is_unicode) & (!s1_is_unicode))) {
23564 return __Pyx_PyBytes_Equals(s1, s2, equals);
23565 }
23566 #endif
23567 if (s1_is_unicode & s2_is_unicode) {
23568 Py_ssize_t length;
23569 int kind;
23570 void *data1, *data2;
23571 if (unlikely(__Pyx_PyUnicode_READY(s1) < 0) || unlikely(__Pyx_PyUnicode_READY(s2) < 0))
23572 return -1;
23573 length = __Pyx_PyUnicode_GET_LENGTH(s1);
23574 if (length != __Pyx_PyUnicode_GET_LENGTH(s2)) {
23575 goto return_ne;
23576 }
23577 #if CYTHON_USE_UNICODE_INTERNALS
23578 {
23579 Py_hash_t hash1, hash2;
23580 #if CYTHON_PEP393_ENABLED
23581 hash1 = ((PyASCIIObject*)s1)->hash;
23582 hash2 = ((PyASCIIObject*)s2)->hash;
23583 #else
23584 hash1 = ((PyUnicodeObject*)s1)->hash;
23585 hash2 = ((PyUnicodeObject*)s2)->hash;
23586 #endif
23587 if (hash1 != hash2 && hash1 != -1 && hash2 != -1) {
23588 goto return_ne;
23589 }
23590 }
23591 #endif
23592 kind = __Pyx_PyUnicode_KIND(s1);
23593 if (kind != __Pyx_PyUnicode_KIND(s2)) {
23594 goto return_ne;
23595 }
23596 data1 = __Pyx_PyUnicode_DATA(s1);
23597 data2 = __Pyx_PyUnicode_DATA(s2);
23598 if (__Pyx_PyUnicode_READ(kind, data1, 0) != __Pyx_PyUnicode_READ(kind, data2, 0)) {
23599 goto return_ne;
23600 } else if (length == 1) {
23601 goto return_eq;
23602 } else {

```

```

23603 int result = memcmp(data1, data2, (size_t)(length * kind));
23604 #if PY_MAJOR_VERSION < 3
23605 Py_XDECREF(owned_ref);
23606 #endif
23607 return (equals == Py_EQ) ? (result == 0) : (result != 0);
23608 }
23609 } else if ((s1 == Py_None) & s2_is_unicode) {
23610 goto return_ne;
23611 } else if ((s2 == Py_None) & s1_is_unicode) {
23612 goto return_ne;
23613 } else {
23614 int result;
23615 PyObject* py_result = PyObject_RichCompare(s1, s2, equals);
23616 #if PY_MAJOR_VERSION < 3
23617 Py_XDECREF(owned_ref);
23618 #endif
23619 if (!py_result)
23620 return -1;
23621 result = __Pyx_PyObject_IsTrue(py_result);
23622 Py_DECREF(py_result);
23623 return result;
23624 }
23625 return_eq:
23626 #if PY_MAJOR_VERSION < 3
23627 Py_XDECREF(owned_ref);
23628 #endif
23629 return (equals == Py_EQ);
23630 return_ne:
23631 #if PY_MAJOR_VERSION < 3
23632 Py_XDECREF(owned_ref);
23633 #endif
23634 return (equals == Py_NE);
23635 #endif
23636 }
23637
23638 /* CLineInTraceback */
23639 #ifndef CYTHON_CLINE_IN_TRACEBACK
23640 static int __Pyx_CLineForTraceback(CYTHON_NCP_UNUSED PyThreadState *tstate, int c_line) {
23641 PyObject *use_cline;
23642 PyObject *ptype, *pvalue, *ptraceback;
23643 #if CYTHON_COMPILING_IN_CPYTHON
23644 PyObject **cython_runtime_dict;
23645 #endif
23646 if (unlikely(!__pyx_cython_runtime)) {
23647 return c_line;
23648 }
23649 __Pyx_ErrFetchInState(tstate, &ptype, &pvalue, &ptraceback);
23650 #if CYTHON_COMPILING_IN_CPYTHON
23651 cython_runtime_dict = _PyObject_GetDictPtr(__pyx_cython_runtime);
23652 if (likely(cython_runtime_dict)) {
23653 __PYX_PY_DICT_LOOKUP_IF_MODIFIED(
23654 use_cline, *cython_runtime_dict,
23655 __Pyx_PyDict_GetItemStr(*cython_runtime_dict, __pyx_n_s_cline_in_traceback))
23656 } else
23657 #endif
23658 {
23659 PyObject *use_cline_obj = __Pyx_PyObject_GetAttrStr(__pyx_cython_runtime,
23660 __pyx_n_s_cline_in_traceback);
23661 if (use_cline_obj) {
23662 use_cline = PyObject_Not(use_cline_obj) ? Py_False : Py_True;
23663 Py_DECREF(use_cline_obj);
23664 } else {
23665 PyErr_Clear();
23666 use_cline = NULL;
23667 }
23668 }
23669 if (!use_cline) {
23670 c_line = 0;
23671 (void) PyObject_SetAttr(__pyx_cython_runtime, __pyx_n_s_cline_in_traceback, Py_False);
23672 }
23673 else if (use_cline == Py_False || (use_cline != Py_True && PyObject_Not(use_cline) != 0)) {
23674 c_line = 0;
23675 }
23676 __Pyx_ErrRestoreInState(tstate, ptype, pvalue, ptraceback);
23677 return c_line;
23678 #endif
23679
23680 /* CodeObjectCache */
23681 static int __pyx_bisect_code_objects(__Pyx_CodeObjectCacheEntry* entries, int count, int code_line) {
23682 int start = 0, mid = 0, end = count - 1;
23683 if (end >= 0 && code_line > entries[end].code_line) {
23684 return count;
23685 }
23686 while (start < end) {
23687 mid = start + (end - start) / 2;
23688 if (code_line < entries[mid].code_line) {

```

```

23689 end = mid;
23690 } else if (code_line > entries[mid].code_line) {
23691 start = mid + 1;
23692 } else {
23693 return mid;
23694 }
23695 }
23696 if (code_line <= entries[mid].code_line) {
23697 return mid;
23698 } else {
23699 return mid + 1;
23700 }
23701 }
23702 static PyCodeObject* __pyx_find_code_object(int code_line) {
23703 PyCodeObject* code_object;
23704 int pos;
23705 if (unlikely(!code_line) || unlikely(!__pyx_code_cache.entries)) {
23706 return NULL;
23707 }
23708 pos = __pyx_bisect_code_objects(__pyx_code_cache.entries, __pyx_code_cache.count, code_line);
23709 if (unlikely(pos >= __pyx_code_cache.count) || unlikely(__pyx_code_cache.entries[pos].code_line !=
code_line)) {
23710 return NULL;
23711 }
23712 code_object = __pyx_code_cache.entries[pos].code_object;
23713 Py_INCREF(code_object);
23714 return code_object;
23715 }
23716 static void __pyx_insert_code_object(int code_line, PyCodeObject* code_object) {
23717 int pos, i;
23718 __Pyx_CodeObjectCacheEntry* entries = __pyx_code_cache.entries;
23719 if (unlikely(!code_line)) {
23720 return;
23721 }
23722 if (unlikely(!entries)) {
23723 entries = (__Pyx_CodeObjectCacheEntry*)PyMem_Malloc(64*sizeof(__Pyx_CodeObjectCacheEntry));
23724 if (likely(entries)) {
23725 __pyx_code_cache.entries = entries;
23726 __pyx_code_cache.max_count = 64;
23727 __pyx_code_cache.count = 1;
23728 entries[0].code_line = code_line;
23729 entries[0].code_object = code_object;
23730 Py_INCREF(code_object);
23731 }
23732 return;
23733 }
23734 pos = __pyx_bisect_code_objects(__pyx_code_cache.entries, __pyx_code_cache.count, code_line);
23735 if ((pos < __pyx_code_cache.count) && unlikely(__pyx_code_cache.entries[pos].code_line ==
code_line)) {
23736 PyCodeObject* tmp = entries[pos].code_object;
23737 entries[pos].code_object = code_object;
23738 Py_DECREF(tmp);
23739 return;
23740 }
23741 if (__pyx_code_cache.count == __pyx_code_cache.max_count) {
23742 int new_max = __pyx_code_cache.max_count + 64;
23743 entries = (__Pyx_CodeObjectCacheEntry*)PyMem_Realloc(
23744 __pyx_code_cache.entries, ((size_t)new_max) * sizeof(__Pyx_CodeObjectCacheEntry));
23745 if (unlikely(!entries)) {
23746 return;
23747 }
23748 __pyx_code_cache.entries = entries;
23749 __pyx_code_cache.max_count = new_max;
23750 }
23751 for (i=__pyx_code_cache.count; i>pos; i--) {
23752 entries[i] = entries[i-1];
23753 }
23754 entries[pos].code_line = code_line;
23755 entries[pos].code_object = code_object;
23756 __pyx_code_cache.count++;
23757 Py_INCREF(code_object);
23758 }
23759
23760 /* AddTraceback */
23761 #include "compile.h"
23762 #include "frameobject.h"
23763 #include "traceback.h"
23764 static PyCodeObject* __Pyx_CreateCodeObjectForTraceback(
23765 const char *funcname, int c_line,
23766 int py_line, const char *filename) {
23767 PyCodeObject *py_code = NULL;
23768 PyObject *py_funcname = NULL;
23769 #if PY_MAJOR_VERSION < 3
23770 PyObject *py_srcfile = NULL;
23771 py_srcfile = PyString_FromString(filename);
23772 if (!py_srcfile) goto bad;
23773 #endif

```

```

23774 if (c_line) {
23775 #if PY_MAJOR_VERSION < 3
23776 py_funcname = PyString_FromFormat("%s (%s:%d)", funcname, __pyx_cfilenm, c_line);
23777 if (!py_funcname) goto bad;
23778 #else
23779 py_funcname = PyUnicode_FromFormat("%s (%s:%d)", funcname, __pyx_cfilenm, c_line);
23780 if (!py_funcname) goto bad;
23781 funcname = PyUnicode_AsUTF8(py_funcname);
23782 if (!funcname) goto bad;
23783 #endif
23784 }
23785 else {
23786 #if PY_MAJOR_VERSION < 3
23787 py_funcname = PyString_FromString(funcname);
23788 if (!py_funcname) goto bad;
23789 #endif
23790 }
23791 #if PY_MAJOR_VERSION < 3
23792 py_code = __Pyx_PyCode_New(
23793 0,
23794 0,
23795 0,
23796 0,
23797 0,
23798 __pyx_empty_bytes, /*PyObject *code,*/
23799 __pyx_empty_tuple, /*PyObject *consts,*/
23800 __pyx_empty_tuple, /*PyObject *names,*/
23801 __pyx_empty_tuple, /*PyObject *varnames,*/
23802 __pyx_empty_tuple, /*PyObject *freevars,*/
23803 __pyx_empty_tuple, /*PyObject *cellvars,*/
23804 py_srcfile, /*PyObject *filename,*/
23805 py_funcname, /*PyObject *name,*/
23806 py_line,
23807 __pyx_empty_bytes /*PyObject *notab*/
23808);
23809 Py_DECREF(py_srcfile);
23810 #else
23811 py_code = PyCode_NewEmpty(filename, funcname, py_line);
23812 #endif
23813 Py_XDECREF(py_funcname); // XDECREF since it's only set on Py3 if cline
23814 return py_code;
23815 bad:
23816 Py_XDECREF(py_funcname);
23817 #if PY_MAJOR_VERSION < 3
23818 Py_XDECREF(py_srcfile);
23819 #endif
23820 return NULL;
23821 }
23822 static void __Pyx_AddTraceback(const char *funcname, int c_line,
23823 int py_line, const char *filename) {
23824 PyCodeObject *py_code = 0;
23825 PyFrameObject *py_frame = 0;
23826 PyThreadState *tstate = __Pyx_PyThreadState_Current;
23827 if (c_line) {
23828 c_line = __Pyx_CLineForTraceback(tstate, c_line);
23829 }
23830 py_code = __pyx_find_code_object(c_line ? -c_line : py_line);
23831 if (!py_code) {
23832 py_code = __Pyx_CreateCodeObjectForTraceback(
23833 funcname, c_line, py_line, filename);
23834 if (!py_code) goto bad;
23835 __pyx_insert_code_object(c_line ? -c_line : py_line, py_code);
23836 }
23837 py_frame = PyFrame_New(
23838 tstate, /*PyThreadState *tstate,*/
23839 py_code, /*PyObject *code,*/
23840 __pyx_d, /*PyObject *globals,*/
23841 0 /*PyObject *locals*/
23842);
23843 if (!py_frame) goto bad;
23844 __Pyx_PyFrame_SetLineNumber(py_frame, py_line);
23845 PyTraceBack_Here(py_frame);
23846 bad:
23847 Py_XDECREF(py_code);
23848 Py_XDECREF(py_frame);
23849 }
23850
23851 /* CIntFromPyVerify */
23852 #define __PYX_VERIFY_RETURN_INT(target_type, func_type, func_value)\
23853 __PYX_VERIFY_RETURN_INT(target_type, func_type, func_value, 0)
23854 #define __PYX_VERIFY_RETURN_INT_EXC(target_type, func_type, func_value)\
23855 __PYX_VERIFY_RETURN_INT(target_type, func_type, func_value, 1)
23856 #define __PYX_VERIFY_RETURN_INT(target_type, func_type, func_value, exc)\
23857 {\
23858 func_type value = func_value;\
23859 if (sizeof(target_type) < sizeof(func_type)) {\
23860 if (unlikely(value != (func_type) (target_type) value)) {\

```



```

23861 func_type zero = 0;\
23862 if (exc && unlikely(value == (func_type)-1 && PyErr_Occurred()))\
23863 return (target_type) -1;\
23864 if (is_unsigned && unlikely(value < zero))\
23865 goto raise_neg_overflow;\
23866 else\
23867 goto raise_overflow;\
23868 }\
23869 }\
23870 return (target_type) value;\
23871 }
23872
23873 /* CIntFromPy */
23874 static CYTHON_INLINE int __Pyx_PyInt_As_int(PyObject *x) {
23875 #ifdef __Pyx_HAS_GCC_DIAGNOSTIC
23876 #pragma GCC diagnostic push
23877 #pragma GCC diagnostic ignored "-Wconversion"
23878 #endif
23879 const int neg_one = (int) -1, const_zero = (int) 0;
23880 #ifdef __Pyx_HAS_GCC_DIAGNOSTIC
23881 #pragma GCC diagnostic pop
23882 #endif
23883 const int is_unsigned = neg_one > const_zero;
23884 #if PY_MAJOR_VERSION < 3
23885 if (likely(PyInt_Check(x))) {
23886 if (sizeof(int) < sizeof(long)) {
23887 __PYX_VERIFY_RETURN_INT(int, long, PyInt_AS_LONG(x))
23888 } else {
23889 long val = PyInt_AS_LONG(x);
23890 if (is_unsigned && unlikely(val < 0)) {
23891 goto raise_neg_overflow;
23892 }
23893 return (int) val;
23894 }
23895 } else
23896 #endif
23897 if (likely(PyLong_Check(x))) {
23898 if (is_unsigned) {
23899 #if CYTHON_USE_PYLONG_INTERNALS
23900 const digit* digits = ((PyLongObject*)x)->ob_digit;
23901 switch (Py_SIZE(x)) {
23902 case 0: return (int) 0;
23903 case 1: __PYX_VERIFY_RETURN_INT(int, digit, digits[0])
23904 case 2:
23905 if (8 * sizeof(int) > 1 * PyLong_SHIFT) {
23906 if (8 * sizeof(unsigned long) > 2 * PyLong_SHIFT) {
23907 __PYX_VERIFY_RETURN_INT(int, unsigned long, (((unsigned long)digits[1])
23908 « PyLong_SHIFT) | (unsigned long)digits[0]))
23909 } else if (8 * sizeof(int) >= 2 * PyLong_SHIFT) {
23910 return (int) (((int)digits[1]) « PyLong_SHIFT) | (int)digits[0]);
23911 }
23912 } break;
23913 case 3:
23914 if (8 * sizeof(int) > 2 * PyLong_SHIFT) {
23915 if (8 * sizeof(unsigned long) > 3 * PyLong_SHIFT) {
23916 __PYX_VERIFY_RETURN_INT(int, unsigned long, ((((((unsigned
long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned
long)digits[0])))
23917 } else if (8 * sizeof(int) >= 3 * PyLong_SHIFT) {
23918 return (int) ((((((int)digits[2]) « PyLong_SHIFT) | (int)digits[1]) «
PyLong_SHIFT) | (int)digits[0]));
23919 }
23920 } break;
23921 case 4:
23922 if (8 * sizeof(int) > 3 * PyLong_SHIFT) {
23923 if (8 * sizeof(unsigned long) > 4 * PyLong_SHIFT) {
23924 __PYX_VERIFY_RETURN_INT(int, unsigned long, (((((((unsigned
long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned
long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
23925 } else if (8 * sizeof(int) >= 4 * PyLong_SHIFT) {
23926 return (int) (((((((int)digits[3]) « PyLong_SHIFT) | (int)digits[2]) «
PyLong_SHIFT) | (int)digits[1]) « PyLong_SHIFT) | (int)digits[0]));
23927 }
23928 } break;
23929 }
23930 } break;
23931 }
23932 #endif
23933 #if CYTHON_COMPILING_IN_CPYTHON
23934 if (unlikely(Py_SIZE(x) < 0)) {
23935 goto raise_neg_overflow;
23936 }
23937 } else
23938 {
23939 int result = PyObject_RichCompareBool(x, Py_False, Py_LT);
23940 if (unlikely(result < 0))

```

```

23941 return (int) -1;
23942 if (unlikely(result == 1))
23943 goto raise_neg_overflow;
23944 }
23945 #endif
23946 if (sizeof(int) <= sizeof(unsigned long)) {
23947 __PYX_VERIFY_RETURN_INT_EXC(int, unsigned long, PyLong_AsUnsignedLong(x))
23948 #ifdef HAVE_LONG_LONG
23949 } else if (sizeof(int) <= sizeof(unsigned PY_LONG_LONG)) {
23950 __PYX_VERIFY_RETURN_INT_EXC(int, unsigned PY_LONG_LONG, PyLong_AsUnsignedLongLong(x))
23951 #endif
23952 }
23953 } else {
23954 #if CYTHON_USE_PYLONG_INTERNALS
23955 const digit* digits = ((PyLongObject*)x)->ob_digit;
23956 switch (Py_SIZE(x)) {
23957 case 0: return (int) 0;
23958 case -1: __PYX_VERIFY_RETURN_INT(int, sdigit, (sdigit) -(sdigit)digits[0]))
23959 case 1: __PYX_VERIFY_RETURN_INT(int, digit, +digits[0])
23960 case -2:
23961 if (8 * sizeof(int) - 1 > 1 * PyLong_SHIFT) {
23962 if (8 * sizeof(unsigned long) > 2 * PyLong_SHIFT) {
23963 __PYX_VERIFY_RETURN_INT(int, long, -(long) (((unsigned long)digits[1]) «
23964 PyLong_SHIFT) | (unsigned long)digits[0]))
23965 } else if (8 * sizeof(int) - 1 > 2 * PyLong_SHIFT) {
23966 return (int) (((int)-1)*((((int)digits[1]) « PyLong_SHIFT) |
23967 (int)digits[0]]));
23968 }
23969 break;
23970 case 2:
23971 if (8 * sizeof(int) > 1 * PyLong_SHIFT) {
23972 if (8 * sizeof(unsigned long) > 2 * PyLong_SHIFT) {
23973 __PYX_VERIFY_RETURN_INT(int, unsigned long, (((unsigned long)digits[1])
23974 « PyLong_SHIFT) | (unsigned long)digits[0]))
23975 } else if (8 * sizeof(int) - 1 > 2 * PyLong_SHIFT) {
23976 return (int) (((((int)digits[1]) « PyLong_SHIFT) | (int)digits[0]]));
23977 }
23978 break;
23979 case -3:
23980 if (8 * sizeof(int) - 1 > 2 * PyLong_SHIFT) {
23981 if (8 * sizeof(unsigned long) > 3 * PyLong_SHIFT) {
23982 __PYX_VERIFY_RETURN_INT(int, long, -(long) (((((unsigned long)digits[2])
23983 « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0]))
23984 } else if (8 * sizeof(int) - 1 > 3 * PyLong_SHIFT) {
23985 return (int) (((int)-1)*((((int)digits[2]) « PyLong_SHIFT) |
23986 (int)digits[1]) « PyLong_SHIFT) | (int)digits[0]]));
23987 }
23988 break;
23989 case 3:
23990 if (8 * sizeof(int) > 2 * PyLong_SHIFT) {
23991 if (8 * sizeof(unsigned long) > 3 * PyLong_SHIFT) {
23992 __PYX_VERIFY_RETURN_INT(int, unsigned long, (((((unsigned
23993 long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned
23994 long)digits[0]))
23995 } else if (8 * sizeof(int) - 1 > 3 * PyLong_SHIFT) {
23996 return (int) (((((((int)digits[2]) « PyLong_SHIFT) | (int)digits[1]) «
23997 PyLong_SHIFT) | (int)digits[0]]));
23998 }
23999 break;
3000 case -4:
3001 if (8 * sizeof(int) - 1 > 3 * PyLong_SHIFT) {
3002 if (8 * sizeof(unsigned long) > 4 * PyLong_SHIFT) {
3003 __PYX_VERIFY_RETURN_INT(int, long, -(long) (((((((unsigned
3004 long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned
3005 long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0]))
3006 } else if (8 * sizeof(int) - 1 > 4 * PyLong_SHIFT) {
3007 return (int) (((((((int)digits[3]) « PyLong_SHIFT) |
3008 (int)digits[2]) « PyLong_SHIFT) | (int)digits[1]) « PyLong_SHIFT) | (int)digits[0]]));
3009 }
3010 break;
3011 case 4:
3012 if (8 * sizeof(int) > 3 * PyLong_SHIFT) {
3013 if (8 * sizeof(unsigned long) > 4 * PyLong_SHIFT) {
3014 __PYX_VERIFY_RETURN_INT(int, unsigned long, (((((((unsigned
3015 long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned
3016 long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0]))
3017 } else if (8 * sizeof(int) - 1 > 4 * PyLong_SHIFT) {
3018 return (int) (((((((int)digits[3]) « PyLong_SHIFT) | (int)digits[2]) «
3019 PyLong_SHIFT) | (int)digits[1]) « PyLong_SHIFT) | (int)digits[0]]));
3020 }
3021 break;

```

```

24014 }
24015 #endif
24016 if (sizeof(int) <= sizeof(long)) {
24017 __PYX_VERIFY_RETURN_INT_EXC(int, long, PyLong_AsLong(x))
24018 #ifdef HAVE_LONG_LONG
24019 } else if (sizeof(int) <= sizeof(PY_LONG_LONG)) {
24020 __PYX_VERIFY_RETURN_INT_EXC(int, PY_LONG_LONG, PyLong_AsLongLong(x))
24021 #endif
24022 }
24023 }
24024 {
24025 #if CYTHON_COMPILING_IN_PYPY && !defined(_PyLong_AsByteArray)
24026 PyErr_SetString(PyExc_RuntimeError,
24027 "_PyLong_AsByteArray() not available in PyPy, cannot convert large
numbers");
24028 #else
24029 int val;
24030 PyObject *v = __Pyx_PyNumber_IntOrLong(x);
24031 #if PY_MAJOR_VERSION < 3
24032 if (likely(v) && !PyLong_Check(v)) {
24033 PyObject *tmp = v;
24034 v = PyNumber_Long(tmp);
24035 Py_DECREF(tmp);
24036 }
24037 #endif
24038 if (likely(v)) {
24039 int one = 1; int is_little = (int)*(unsigned char *)&one;
24040 unsigned char *bytes = (unsigned char *)&val;
24041 int ret = _PyLong_AsByteArray((PyLongObject *)v,
24042 bytes, sizeof(val),
24043 is_little, !is_unsigned);
24044 Py_DECREF(v);
24045 if (likely(!ret))
24046 return val;
24047 }
24048 #endif
24049 return (int) -1;
24050 }
24051 } else {
24052 int val;
24053 PyObject *tmp = __Pyx_PyNumber_IntOrLong(x);
24054 if (!tmp) return (int) -1;
24055 val = __Pyx_PyInt_As_int(tmp);
24056 Py_DECREF(tmp);
24057 return val;
24058 }
24059 raise_overflow:
24060 PyErr_SetString(PyExc_OverflowError,
24061 "value too large to convert to int");
24062 return (int) -1;
24063 raise_neg_overflow:
24064 PyErr_SetString(PyExc_OverflowError,
24065 "can't convert negative value to int");
24066 return (int) -1;
24067 }
24068
24069 /* CIntToPy */
24070 static CYTHON_INLINE PyObject* __Pyx_PyInt_From_int(int value) {
24071 #ifdef __Pyx_HAS_GCC_DIAGNOSTIC
24072 #pragma GCC diagnostic push
24073 #pragma GCC diagnostic ignored "-Wconversion"
24074 #endif
24075 const int neg_one = (int) -1, const_zero = (int) 0;
24076 #ifdef __Pyx_HAS_GCC_DIAGNOSTIC
24077 #pragma GCC diagnostic pop
24078 #endif
24079 const int is_unsigned = neg_one > const_zero;
24080 if (is_unsigned) {
24081 if (sizeof(int) < sizeof(long)) {
24082 return PyInt_FromLong((long) value);
24083 } else if (sizeof(int) <= sizeof(unsigned long)) {
24084 return PyLong_FromUnsignedLong((unsigned long) value);
24085 #ifdef HAVE_LONG_LONG
24086 } else if (sizeof(int) <= sizeof(unsigned PY_LONG_LONG)) {
24087 return PyLong_FromUnsignedLongLong((unsigned PY_LONG_LONG) value);
24088 #endif
24089 }
24090 } else {
24091 if (sizeof(int) <= sizeof(long)) {
24092 return PyInt_FromLong((long) value);
24093 #ifdef HAVE_LONG_LONG
24094 } else if (sizeof(int) <= sizeof(PY_LONG_LONG)) {
24095 return PyLong_FromLongLong((PY_LONG_LONG) value);
24096 #endif
24097 }
24098 }
24099 {

```

```

24100 int one = 1; int little = (int)*(unsigned char *)&one;
24101 unsigned char *bytes = (unsigned char *)&value;
24102 return _PyLong_FromByteArray(bytes, sizeof(int),
24103 little, !is_unsigned);
24104 }
24105 }
24106
24107 /* CIntToPy */
24108 static CYTHON_INLINE PyObject* __Pyx_PyInt_From_long(long value) {
24109 #ifdef __Pyx_HAS_GCC_DIAGNOSTIC
24110 #pragma GCC diagnostic push
24111 #pragma GCC diagnostic ignored "-Wconversion"
24112 #endif
24113 const long neg_one = (long) -1, const_zero = (long) 0;
24114 #ifdef __Pyx_HAS_GCC_DIAGNOSTIC
24115 #pragma GCC diagnostic pop
24116 #endif
24117 const int is_unsigned = neg_one > const_zero;
24118 if (is_unsigned) {
24119 if (sizeof(long) < sizeof(unsigned long)) {
24120 return PyInt_FromLong((long) value);
24121 } else if (sizeof(long) <= sizeof(unsigned long)) {
24122 return PyLong_FromUnsignedLong((unsigned long) value);
24123 #ifdef HAVE_LONG_LONG
24124 } else if (sizeof(long) <= sizeof(unsigned PY_LONG_LONG)) {
24125 return PyLong_FromUnsignedLongLong((unsigned PY_LONG_LONG) value);
24126 #endif
24127 }
24128 } else {
24129 if (sizeof(long) <= sizeof(long)) {
24130 return PyInt_FromLong((long) value);
24131 #ifdef HAVE_LONG_LONG
24132 } else if (sizeof(long) <= sizeof(PY_LONG_LONG)) {
24133 return PyLong_FromLongLong((PY_LONG_LONG) value);
24134 #endif
24135 }
24136 }
24137 {
24138 int one = 1; int little = (int)*(unsigned char *)&one;
24139 unsigned char *bytes = (unsigned char *)&value;
24140 return _PyLong_FromByteArray(bytes, sizeof(long),
24141 little, !is_unsigned);
24142 }
24143 }
24144
24145 /* CIntFromPy */
24146 static CYTHON_INLINE long __Pyx_PyInt_As_long(PyObject *x) {
24147 #ifdef __Pyx_HAS_GCC_DIAGNOSTIC
24148 #pragma GCC diagnostic push
24149 #pragma GCC diagnostic ignored "-Wconversion"
24150 #endif
24151 const long neg_one = (long) -1, const_zero = (long) 0;
24152 #ifdef __Pyx_HAS_GCC_DIAGNOSTIC
24153 #pragma GCC diagnostic pop
24154 #endif
24155 const int is_unsigned = neg_one > const_zero;
24156 #if PY_MAJOR_VERSION < 3
24157 if (likely(PyInt_Check(x))) {
24158 if (sizeof(long) < sizeof(unsigned long)) {
24159 __PYX_VERIFY_RETURN_INT(long, long, PyInt_AS_LONG(x))
24160 } else {
24161 long val = PyInt_AS_LONG(x);
24162 if (is_unsigned && unlikely(val < 0)) {
24163 goto raise_neg_overflow;
24164 }
24165 return (long) val;
24166 }
24167 } else
24168 #endif
24169 if (likely(PyLong_Check(x))) {
24170 if (is_unsigned) {
24171 #if CYTHON_USE_PYLONG_INTERNALS
24172 const digit* digits = ((PyLongObject*)x)->ob_digit;
24173 switch (Py_SIZE(x)) {
24174 case 0: return (long) 0;
24175 case 1: __PYX_VERIFY_RETURN_INT(long, digit, digits[0])
24176 case 2:
24177 if (8 * sizeof(long) > 1 * PyLong_SHIFT) {
24178 if (8 * sizeof(unsigned long) > 2 * PyLong_SHIFT) {
24179 __PYX_VERIFY_RETURN_INT(long, unsigned long, (((unsigned long)digits[1])
24180 « PyLong_SHIFT) | (unsigned long)digits[0]))
24181 } else if (8 * sizeof(long) >= 2 * PyLong_SHIFT) {
24182 return (long) (((((long)digits[1]) « PyLong_SHIFT) | (long)digits[0]));
24183 }
24184 }
24185 break;
24186 case 3:

```

```

24186 if (8 * sizeof(long) > 2 * PyLong_SHIFT) {
24187 if (8 * sizeof(unsigned long) > 3 * PyLong_SHIFT) {
24188 __PYX_VERIFY_RETURN_INT(long, unsigned long, ((((((unsigned
long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned
long)digits[0])))
24189 } else if (8 * sizeof(long) >= 3 * PyLong_SHIFT) {
24190 return (long) ((((((long)digits[2]) « PyLong_SHIFT) | (long)digits[1]) «
PyLong_SHIFT) | (long)digits[0]));
24191 }
24192 }
24193 break;
24194 case 4:
24195 if (8 * sizeof(long) > 3 * PyLong_SHIFT) {
24196 if (8 * sizeof(unsigned long) > 4 * PyLong_SHIFT) {
24197 __PYX_VERIFY_RETURN_INT(long, unsigned long, (((((((((unsigned
long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned
long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
24198 } else if (8 * sizeof(long) >= 4 * PyLong_SHIFT) {
24199 return (long) (((((((((long)digits[3]) « PyLong_SHIFT) | (long)digits[2])
« PyLong_SHIFT) | (long)digits[1]) « PyLong_SHIFT) | (long)digits[0]));
24200 }
24201 }
24202 break;
24203 }
24204 #endif
24205 #if CYTHON_COMPILING_IN_CPYTHON
24206 if (unlikely(Py_SIZE(x) < 0)) {
24207 goto raise_neg_overflow;
24208 }
24209 #else
24210 {
24211 int result = PyObject_RichCompareBool(x, Py_False, Py_LT);
24212 if (unlikely(result < 0))
24213 return (long) -1;
24214 if (unlikely(result == 1))
24215 goto raise_neg_overflow;
24216 }
24217 #endif
24218 if (sizeof(long) <= sizeof(unsigned long)) {
24219 __PYX_VERIFY_RETURN_INT_EXC(long, unsigned long, PyLong_AsUnsignedLong(x))
24220 #ifdef HAVE_LONG_LONG
24221 } else if (sizeof(long) <= sizeof(unsigned PY_LONG_LONG)) {
24222 __PYX_VERIFY_RETURN_INT_EXC(long, unsigned PY_LONG_LONG, PyLong_AsUnsignedLongLong(x))
24223 #endif
24224 }
24225 } else {
24226 #if CYTHON_USE_PYLONG_INTERNALS
24227 const digit* digits = ((PyLongObject*)x)->ob_digit;
24228 switch (Py_SIZE(x)) {
24229 case 0: return (long) 0;
24230 case -1: __PYX_VERIFY_RETURN_INT(long, sdigit, (sdigit) -(sdigit)digits[0]))
24231 case 1: __PYX_VERIFY_RETURN_INT(long, digit, +digits[0])
24232 case -2:
24233 if (8 * sizeof(long) - 1 > 1 * PyLong_SHIFT) {
24234 if (8 * sizeof(unsigned long) > 2 * PyLong_SHIFT) {
24235 __PYX_VERIFY_RETURN_INT(long, long, -(long) (((((unsigned long)digits[1])
« PyLong_SHIFT) | (unsigned long)digits[0])))
24236 } else if (8 * sizeof(long) - 1 > 2 * PyLong_SHIFT) {
24237 return (long) (((long)-1)*(((long)digits[1]) « PyLong_SHIFT) |
(long)digits[0]));
24238 }
24239 break;
24240 }
24241 case 2:
24242 if (8 * sizeof(long) > 1 * PyLong_SHIFT) {
24243 if (8 * sizeof(unsigned long) > 2 * PyLong_SHIFT) {
24244 __PYX_VERIFY_RETURN_INT(long, unsigned long, (((((unsigned long)digits[1])
« PyLong_SHIFT) | (unsigned long)digits[0])))
24245 } else if (8 * sizeof(long) - 1 > 2 * PyLong_SHIFT) {
24246 return (long) (((((long)digits[1]) « PyLong_SHIFT) | (long)digits[0]));
24247 }
24248 break;
24249 }
24250 case -3:
24251 if (8 * sizeof(long) - 1 > 2 * PyLong_SHIFT) {
24252 if (8 * sizeof(unsigned long) > 3 * PyLong_SHIFT) {
24253 __PYX_VERIFY_RETURN_INT(long, long, -(long) (((((((unsigned
long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned
long)digits[0])))
24254 } else if (8 * sizeof(long) - 1 > 3 * PyLong_SHIFT) {
24255 return (long) (((long)-1)*((((((long)digits[2]) « PyLong_SHIFT) |
(long)digits[1]) « PyLong_SHIFT) | (long)digits[0]));
24256 }
24257 break;
24258 }
24259 case 3:
24260 if (8 * sizeof(long) > 2 * PyLong_SHIFT) {

```

```

24261 if (8 * sizeof(unsigned long) > 3 * PyLong_SHIFT) {
24262 __PYX_VERIFY_RETURN_INT(long, unsigned long, ((((((unsigned
long)digits[2]) « PyLong_SHIFT) | (unsigned long)digits[1]) « PyLong_SHIFT) | (unsigned
long)digits[0])))
24263 } else if (8 * sizeof(long) - 1 > 3 * PyLong_SHIFT) {
24264 return (long) ((((((long)digits[2]) « PyLong_SHIFT) | (long)digits[1]) «
PyLong_SHIFT) | (long)digits[0]));
24265 }
24266 }
24267 break;
24268 case -4:
24269 if (8 * sizeof(long) - 1 > 3 * PyLong_SHIFT) {
24270 if (8 * sizeof(unsigned long) > 4 * PyLong_SHIFT) {
24271 __PYX_VERIFY_RETURN_INT(long, long, -(long) (((((((unsigned
long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned
long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
24272 } else if (8 * sizeof(long) - 1 > 4 * PyLong_SHIFT) {
24273 return (long) (((((long)-1)*(((((((long)digits[3]) « PyLong_SHIFT) |
(long)digits[2]) « PyLong_SHIFT) | (long)digits[1]) « PyLong_SHIFT) | (long)digits[0])));
24274 }
24275 }
24276 break;
24277 case 4:
24278 if (8 * sizeof(long) > 3 * PyLong_SHIFT) {
24279 if (8 * sizeof(unsigned long) > 4 * PyLong_SHIFT) {
24280 __PYX_VERIFY_RETURN_INT(long, unsigned long, (((((((unsigned
long)digits[3]) « PyLong_SHIFT) | (unsigned long)digits[2]) « PyLong_SHIFT) | (unsigned
long)digits[1]) « PyLong_SHIFT) | (unsigned long)digits[0])))
24281 } else if (8 * sizeof(long) - 1 > 4 * PyLong_SHIFT) {
24282 return (long) ((((((((((long)digits[3]) « PyLong_SHIFT) | (long)digits[2])
« PyLong_SHIFT) | (long)digits[1]) « PyLong_SHIFT) | (long)digits[0])));
24283 }
24284 }
24285 break;
24286 }
24287 #endif
24288 if (sizeof(long) <= sizeof(long)) {
24289 __PYX_VERIFY_RETURN_INT_EXC(long, long, PyLong_AsLong(x))
24290 #ifdef HAVE_LONG_LONG
24291 } else if (sizeof(long) <= sizeof(PY_LONG_LONG)) {
24292 __PYX_VERIFY_RETURN_INT_EXC(long, PY_LONG_LONG, PyLong_AsLongLong(x))
24293 #endif
24294 }
24295 }
24296 {
24297 #if CYTHON_COMPILING_IN_PYPY && !defined(_PyLong_AsByteArray)
24298 PyErr_SetString(PyExc_RuntimeError,
24299 "_PyLong_AsByteArray() not available in PyPy, cannot convert large
numbers");
24300 #else
24301 long val;
24302 PyObject *v = __Pyx_PyNumber_IntOrLong(x);
24303 #if PY_MAJOR_VERSION < 3
24304 if (likely(v) && !PyLong_Check(v)) {
24305 PyObject *tmp = v;
24306 v = PyNumber_Long(tmp);
24307 Py_DECREF(tmp);
24308 }
24309 #endif
24310 if (likely(v)) {
24311 int one = 1; int is_little = (int)*(unsigned char *)&one;
24312 unsigned char *bytes = (unsigned char *)&val;
24313 int ret = _PyLong_AsByteArray((PyLongObject *)v,
24314 bytes, sizeof(val),
24315 is_little, !is_unsigned);
24316 Py_DECREF(v);
24317 if (likely(!ret))
24318 return val;
24319 }
24320 #endif
24321 return (long) -1;
24322 }
24323 } else {
24324 long val;
24325 PyObject *tmp = __Pyx_PyNumber_IntOrLong(x);
24326 if (!tmp) return (long) -1;
24327 val = __Pyx_PyInt_As_long(tmp);
24328 Py_DECREF(tmp);
24329 return val;
24330 }
24331 raise_overflow:
24332 PyErr_SetString(PyExc_OverflowError,
24333 "value too large to convert to long");
24334 return (long) -1;
24335 raise_neg_overflow:
24336 PyErr_SetString(PyExc_OverflowError,
24337 "can't convert negative value to long");

```

```

24338 return (long) -1;
24339 }
24340
24341 /* FastTypeChecks */
24342 #if CYTHON_COMPILING_IN_CPYTHON
24343 static int __Pyx_InBases(PyTypeObject *a, PyTypeObject *b) {
24344 while (a) {
24345 a = a->tp_base;
24346 if (a == b)
24347 return 1;
24348 }
24349 return b == &PyBaseObject_Type;
24350 }
24351 static CYTHON_INLINE int __Pyx_IsSubtype(PyTypeObject *a, PyTypeObject *b) {
24352 PyObject *mro;
24353 if (a == b) return 1;
24354 mro = a->tp_mro;
24355 if (likely(mro)) {
24356 Py_ssize_t i, n;
24357 n = PyTuple_GET_SIZE(mro);
24358 for (i = 0; i < n; i++) {
24359 if (PyTuple_GET_ITEM(mro, i) == (PyObject *)b)
24360 return 1;
24361 }
24362 return 0;
24363 }
24364 return __Pyx_InBases(a, b);
24365 }
24366 #if PY_MAJOR_VERSION == 2
24367 static int __Pyx_inner_PyErr_GivenExceptionMatches2(PyObject *err, PyObject* exc_type1, PyObject*
exc_type2) {
24368 PyObject *exception, *value, *tb;
24369 int res;
24370 __Pyx_PyThreadState_declare
24371 __Pyx_PyThreadState_assign
24372 __Pyx_ErrFetch(&exception, &value, &tb);
24373 res = exc_type1 ? PyObject_IsSubclass(err, exc_type1) : 0;
24374 if (unlikely(res == -1)) {
24375 PyErr_WriteUnraisable(err);
24376 res = 0;
24377 }
24378 if (!res) {
24379 res = PyObject_IsSubclass(err, exc_type2);
24380 if (unlikely(res == -1)) {
24381 PyErr_WriteUnraisable(err);
24382 res = 0;
24383 }
24384 }
24385 __Pyx_ErrRestore(exception, value, tb);
24386 return res;
24387 }
24388 #else
24389 static CYTHON_INLINE int __Pyx_inner_PyErr_GivenExceptionMatches2(PyObject *err, PyObject* exc_type1,
PyObject* exc_type2) {
24390 int res = exc_type1 ? __Pyx_IsSubtype((PyTypeObject*)err, (PyTypeObject*)exc_type1) : 0;
24391 if (!res) {
24392 res = __Pyx_IsSubtype((PyTypeObject*)err, (PyTypeObject*)exc_type2);
24393 }
24394 return res;
24395 }
24396 #endif
24397 static int __Pyx_PyErr_GivenExceptionMatchesTuple(PyObject *exc_type, PyObject *tuple) {
24398 Py_ssize_t i, n;
24399 assert(PyExceptionClass_Check(exc_type));
24400 n = PyTuple_GET_SIZE(tuple);
24401 #if PY_MAJOR_VERSION >= 3
24402 for (i=0; i<n; i++) {
24403 if (exc_type == PyTuple_GET_ITEM(tuple, i)) return 1;
24404 }
24405 #endif
24406 for (i=0; i<n; i++) {
24407 PyObject *t = PyTuple_GET_ITEM(tuple, i);
24408 #if PY_MAJOR_VERSION < 3
24409 if (likely(exc_type == t)) return 1;
24410 #endif
24411 if (likely(PyExceptionClass_Check(t))) {
24412 if (__Pyx_inner_PyErr_GivenExceptionMatches2(exc_type, NULL, t)) return 1;
24413 } else {
24414 }
24415 }
24416 return 0;
24417 }
24418 static CYTHON_INLINE int __Pyx_PyErr_GivenExceptionMatches(PyObject *err, PyObject* exc_type) {
24419 if (likely(err == exc_type)) return 1;
24420 if (likely(PyExceptionClass_Check(err))) {
24421 if (likely(PyExceptionClass_Check(exc_type))) {
24422 return __Pyx_inner_PyErr_GivenExceptionMatches2(err, NULL, exc_type);

```

```

24423 } else if (likely(PyTuple_Check(exc_type))) {
24424 return __Pyx_PyErr_GivenExceptionMatchesTuple(err, exc_type);
24425 } else {
24426 }
24427 }
24428 return PyErr_GivenExceptionMatches(err, exc_type);
24429 }
24430 static CYTHON_INLINE int __Pyx_PyErr_GivenExceptionMatches2(PyObject *err, PyObject *exc_type1,
24431 PyObject *exc_type2) {
24432 assert(PyExceptionClass_Check(exc_type1));
24433 assert(PyExceptionClass_Check(exc_type2));
24434 if (likely(err == exc_type1 || err == exc_type2)) return 1;
24435 if (likely(PyExceptionClass_Check(err))) {
24436 return __Pyx_inner_PyErr_GivenExceptionMatches2(err, exc_type1, exc_type2);
24437 }
24438 return (PyErr_GivenExceptionMatches(err, exc_type1) || PyErr_GivenExceptionMatches(err,
24439 exc_type2));
24440 }
24441 #endif
24442 /* FetchCommonType */
24443 static PyTypeObject* __Pyx_FetchCommonType(PyTypeObject* type) {
24444 PyObject* fake_module;
24445 PyTypeObject* cached_type = NULL;
24446 fake_module = PyImport_AddModule((char*) "_cython_" CYTHON_ABI);
24447 if (!fake_module) return NULL;
24448 Py_INCREF(fake_module);
24449 cached_type = (PyTypeObject*) PyObject_GetAttrString(fake_module, type->tp_name);
24450 if (cached_type) {
24451 if (!PyType_Check((PyObject*)cached_type)) {
24452 PyErr_Format(PyExc_TypeError,
24453 "Shared Cython type %.200s is not a type object",
24454 type->tp_name);
24455 goto bad;
24456 }
24457 if (cached_type->tp_basicsize != type->tp_basicsize) {
24458 PyErr_Format(PyExc_TypeError,
24459 "Shared Cython type %.200s has the wrong size, try recompiling",
24460 type->tp_name);
24461 goto bad;
24462 }
24463 } else {
24464 if (!PyErr_ExceptionMatches(PyExc_AttributeError)) goto bad;
24465 PyErr_Clear();
24466 if (PyType_Ready(type) < 0) goto bad;
24467 if (PyObject_SetAttrString(fake_module, type->tp_name, (PyObject*) type) < 0)
24468 goto bad;
24469 Py_INCREF(type);
24470 cached_type = type;
24471 }
24472 done:
24473 Py_DECREF(fake_module);
24474 return cached_type;
24475 bad:
24476 Py_XDECREF(cached_type);
24477 cached_type = NULL;
24478 goto done;
24479 }
24480 /* PyObjectGetMethod */
24481 static int __Pyx_PyObject_GetMethod(PyObject *obj, PyObject *name, PyObject **method) {
24482 PyObject *attr;
24483 #if CYTHON_UNPACK_METHODS && CYTHON_COMPILING_IN_CPYTHON && CYTHON_USE_PYTYPE_LOOKUP
24484 PyTypeObject *tp = Py_TYPE(obj);
24485 PyObject *descr;
24486 descrgetfunc f = NULL;
24487 PyObject **dictptr, *dict;
24488 int meth_found = 0;
24489 assert (*method == NULL);
24490 if (unlikely(tp->tp_getattro != PyObject_GenericGetAttr)) {
24491 attr = __Pyx_PyObject_GetAttrStr(obj, name);
24492 goto try_unpack;
24493 }
24494 if (unlikely(tp->tp_dict == NULL) && unlikely(PyType_Ready(tp) < 0)) {
24495 return 0;
24496 }
24497 descr = _PyType_Lookup(tp, name);
24498 if (likely(descr != NULL)) {
24499 Py_INCREF(descr);
24500 #if PY_MAJOR_VERSION >= 3
24501 #ifdef __Pyx_CyFunction_USED
24502 if (likely(PyFunction_Check(descr) || (Py_TYPE(descr) == &PyMethodDescr_Type) ||
24503 __Pyx_CyFunction_Check(descr)))
24504 #else
24505 if (likely(PyFunction_Check(descr) || (Py_TYPE(descr) == &PyMethodDescr_Type)))
24506 #endif
24507 #else

```



```

24507 #ifdef __Pyx_CyFunction_USED
24508 if (likely(PyFunction_Check(descr) || __Pyx_CyFunction_Check(descr)))
24509 #else
24510 if (likely(PyFunction_Check(descr)))
24511 #endif
24512 #endif
24513 {
24514 meth_found = 1;
24515 } else {
24516 f = Py_TYPE(descr)->tp_descr_get;
24517 if (f != NULL && PyDescr_IsData(descr)) {
24518 attr = f(descr, obj, (PyObject *)Py_TYPE(obj));
24519 Py_DECREF(descr);
24520 goto try_unpack;
24521 }
24522 }
24523 }
24524 dictptr = _PyObject_GetDictPtr(obj);
24525 if (dictptr != NULL && (dict = *dictptr) != NULL) {
24526 Py_INCREF(dict);
24527 attr = __Pyx_PyDict_GetItemStr(dict, name);
24528 if (attr != NULL) {
24529 Py_INCREF(attr);
24530 Py_DECREF(dict);
24531 Py_XDECREF(descr);
24532 goto try_unpack;
24533 }
24534 Py_DECREF(dict);
24535 }
24536 if (meth_found) {
24537 *method = descr;
24538 return 1;
24539 }
24540 if (f != NULL) {
24541 attr = f(descr, obj, (PyObject *)Py_TYPE(obj));
24542 Py_DECREF(descr);
24543 goto try_unpack;
24544 }
24545 if (descr != NULL) {
24546 *method = descr;
24547 return 0;
24548 }
24549 PyErr_Format(PyExc_AttributeError,
24550 #if PY_MAJOR_VERSION >= 3
24551 "'%.50s' object has no attribute '%U'",
24552 tp->tp_name, name);
24553 #else
24554 "'%.50s' object has no attribute '%.400s'",
24555 tp->tp_name, PyString_AS_STRING(name));
24556 #endif
24557 return 0;
24558 #else
24559 attr = __Pyx_PyObject_GetAttrStr(obj, name);
24560 goto try_unpack;
24561 #endif
24562 try_unpack:
24563 #if CYTHON_UNPACK_METHODS
24564 if (likely(attr) && PyMethod_Check(attr) && likely(PyMethod_GET_SELF(attr) == obj)) {
24565 PyObject *function = PyMethod_GET_FUNCTION(attr);
24566 Py_INCREF(function);
24567 Py_DECREF(attr);
24568 *method = function;
24569 return 1;
24570 }
24571 #endif
24572 *method = attr;
24573 return 0;
24574 }
24575
24576 /* PyObjectCallMethod1 */
24577 static PyObject* __Pyx__PyObject_CallMethod1(PyObject* method, PyObject* arg) {
24578 PyObject *result = __Pyx_PyObject_CallOneArg(method, arg);
24579 Py_DECREF(method);
24580 return result;
24581 }
24582 static PyObject* __Pyx_PyObject_CallMethod1(PyObject* obj, PyObject* method_name, PyObject* arg) {
24583 PyObject *method = NULL, *result;
24584 int is_method = __Pyx_PyObject_GetMethod(obj, method_name, &method);
24585 if (likely(is_method)) {
24586 result = __Pyx_PyObject_Call2Args(method, obj, arg);
24587 Py_DECREF(method);
24588 return result;
24589 }
24590 if (unlikely(!method)) return NULL;
24591 return __Pyx__PyObject_CallMethod1(method, arg);
24592 }
24593

```

```

24594 /* CoroutineBase */
24595 #include <structmember.h>
24596 #include <frameobject.h>
24597 #define __Pyx_Coroutine_Undelegate(gen) Py_CLEAR((gen)->yieldfrom)
24598 static int __Pyx_PyGen_FetchStopIterationValue(CYTHON_UNUSED PyThreadState *__pyx_tstate, PyObject
**pvalue) {
24599 PyObject *et, *ev, *tb;
24600 PyObject *value = NULL;
24601 __Pyx_ErrFetch(&et, &ev, &tb);
24602 if (!et) {
24603 Py_XDECREF(tb);
24604 Py_XDECREF(ev);
24605 Py_INCREF(Py_None);
24606 *pvalue = Py_None;
24607 return 0;
24608 }
24609 if (likely(et == PyExc_StopIteration)) {
24610 if (!ev) {
24611 Py_INCREF(Py_None);
24612 value = Py_None;
24613 }
24614 #if PY_VERSION_HEX >= 0x030300A0
24615 else if (Py_TYPE(ev) == (PyTypeObject*)PyExc_StopIteration) {
24616 value = ((PyStopIterationObject *)ev)->value;
24617 Py_INCREF(value);
24618 Py_DECREF(ev);
24619 }
24620 #endif
24621 else if (unlikely(PyTuple_Check(ev))) {
24622 if (PyTuple_GET_SIZE(ev) >= 1) {
24623 #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
24624 value = PyTuple_GET_ITEM(ev, 0);
24625 Py_INCREF(value);
24626 #else
24627 value = PySequence_ITEM(ev, 0);
24628 #endif
24629 } else {
24630 Py_INCREF(Py_None);
24631 value = Py_None;
24632 }
24633 Py_DECREF(ev);
24634 }
24635 else if (!__Pyx_TypeCheck(ev, (PyTypeObject*)PyExc_StopIteration)) {
24636 value = ev;
24637 }
24638 if (likely(value)) {
24639 Py_XDECREF(tb);
24640 Py_DECREF(et);
24641 *pvalue = value;
24642 return 0;
24643 }
24644 } else if (!__Pyx_PyErr_GivenExceptionMatches(et, PyExc_StopIteration)) {
24645 __Pyx_ErrRestore(et, ev, tb);
24646 return -1;
24647 }
24648 PyErr_NormalizeException(&et, &ev, &tb);
24649 if (unlikely(!PyObject_TypeCheck(ev, (PyTypeObject*)PyExc_StopIteration))) {
24650 __Pyx_ErrRestore(et, ev, tb);
24651 return -1;
24652 }
24653 Py_XDECREF(tb);
24654 Py_DECREF(et);
24655 #if PY_VERSION_HEX >= 0x030300A0
24656 value = ((PyStopIterationObject *)ev)->value;
24657 Py_INCREF(value);
24658 Py_DECREF(ev);
24659 #else
24660 {
24661 PyObject* args = __Pyx_PyObject_GetAttrStr(ev, __pyx_n_s_args);
24662 Py_DECREF(ev);
24663 if (likely(args)) {
24664 value = PySequence_GetItem(args, 0);
24665 Py_DECREF(args);
24666 }
24667 if (unlikely(!value)) {
24668 __Pyx_ErrRestore(NULL, NULL, NULL);
24669 Py_INCREF(Py_None);
24670 value = Py_None;
24671 }
24672 }
24673 #endif
24674 *pvalue = value;
24675 return 0;
24676 }
24677 static CYTHON_INLINE
24678 void __Pyx_Coroutine_ExceptionClear(__Pyx_ExcInfoStruct *exc_state) {
24679 PyObject *t, *v, *tb;

```

```

24680 t = exc_state->exc_type;
24681 v = exc_state->exc_value;
24682 tb = exc_state->exc_traceback;
24683 exc_state->exc_type = NULL;
24684 exc_state->exc_value = NULL;
24685 exc_state->exc_traceback = NULL;
24686 Py_XDECREF(t);
24687 Py_XDECREF(v);
24688 Py_XDECREF(tb);
24689 }
24690 #define __Pyx_Coroutine_AlreadyRunningError(gen) (__Pyx_Coroutine_AlreadyRunningError(gen),
(PyObject*)NULL)
24691 static void __Pyx_Coroutine_AlreadyRunningError(CYTHON_UNUSED __pyx_CoroutineObject *gen) {
24692 const char *msg;
24693 if ((0)) {
24694 #ifdef __Pyx_Coroutine_USED
24695 } else if (__Pyx_Coroutine_Check((PyObject*)gen)) {
24696 msg = "coroutine already executing";
24697 #endif
24698 #ifdef __Pyx_AsyncGen_USED
24699 } else if (__Pyx_AsyncGen_CheckExact((PyObject*)gen)) {
24700 msg = "async generator already executing";
24701 #endif
24702 } else {
24703 msg = "generator already executing";
24704 }
24705 PyErr_SetString(PyExc_ValueError, msg);
24706 }
24707 #define __Pyx_Coroutine_NotStartedError(gen) (__Pyx_Coroutine_NotStartedError(gen), (PyObject*)NULL)
24708 static void __Pyx_Coroutine_NotStartedError(CYTHON_UNUSED PyObject *gen) {
24709 const char *msg;
24710 if ((0)) {
24711 #ifdef __Pyx_Coroutine_USED
24712 } else if (__Pyx_Coroutine_Check(gen)) {
24713 msg = "can't send non-None value to a just-started coroutine";
24714 #endif
24715 #ifdef __Pyx_AsyncGen_USED
24716 } else if (__Pyx_AsyncGen_CheckExact(gen)) {
24717 msg = "can't send non-None value to a just-started async generator";
24718 #endif
24719 } else {
24720 msg = "can't send non-None value to a just-started generator";
24721 }
24722 PyErr_SetString(PyExc_TypeError, msg);
24723 }
24724 #define __Pyx_Coroutine_AlreadyTerminatedError(gen, value, closing)
(__Pyx_Coroutine_AlreadyTerminatedError(gen, value, closing), (PyObject*)NULL)
24725 static void __Pyx_Coroutine_AlreadyTerminatedError(CYTHON_UNUSED PyObject *gen, PyObject *value,
CYTHON_UNUSED int closing) {
24726 #ifdef __Pyx_Coroutine_USED
24727 if (!closing && __Pyx_Coroutine_Check(gen)) {
24728 PyErr_SetString(PyExc_RuntimeError, "cannot reuse already awaited coroutine");
24729 } else
24730 #endif
24731 if (value) {
24732 #ifdef __Pyx_AsyncGen_USED
24733 if (__Pyx_AsyncGen_CheckExact(gen))
24734 PyErr_SetNone(__Pyx_PyExc_StopAsyncIteration);
24735 else
24736 #endif
24737 PyErr_SetNone(PyExc_StopIteration);
24738 }
24739 }
24740 static
24741 PyObject *__Pyx_Coroutine_SendEx(__pyx_CoroutineObject *self, PyObject *value, int closing) {
24742 __Pyx_PyThreadState_declare
24743 PyThreadState *tstate;
24744 __Pyx_ExcInfoStruct *exc_state;
24745 PyObject *retval;
24746 assert(!self->is_running);
24747 if (unlikely(self->resume_label == 0)) {
24748 if (unlikely(value && value != Py_None)) {
24749 return __Pyx_Coroutine_NotStartedError((PyObject*)self);
24750 }
24751 }
24752 if (unlikely(self->resume_label == -1)) {
24753 return __Pyx_Coroutine_AlreadyTerminatedError((PyObject*)self, value, closing);
24754 }
24755 #if CYTHON_FAST_THREAD_STATE
24756 __Pyx_PyThreadState_assign
24757 tstate = __pyx_tstate;
24758 #else
24759 tstate = __Pyx_PyThreadState_Current;
24760 #endif
24761 exc_state = &self->gi_exc_state;
24762 if (exc_state->exc_type) {
24763 #if CYTHON_COMPILING_IN_PYPY || CYTHON_COMPILING_IN_PYSTON

```

```

24764 #else
24765 if (exc_state->exc_traceback) {
24766 PyTracebackObject *tb = (PyTracebackObject *) exc_state->exc_traceback;
24767 PyFrameObject *f = tb->tb_frame;
24768 assert(f->f_back == NULL);
24769 #if PY_VERSION_HEX >= 0x030B00A1
24770 f->f_back = PyThreadState_GetFrame(tstate);
24771 #else
24772 Py_XINCREf(tstate->frame);
24773 f->f_back = tstate->frame;
24774 #endif
24775 }
24776 #endif
24777 }
24778 #if CYTHON_USE_EXC_INFO_STACK
24779 exc_state->previous_item = tstate->exc_info;
24780 tstate->exc_info = exc_state;
24781 #else
24782 if (exc_state->exc_type) {
24783 __Pyx_ExceptionSwap(&exc_state->exc_type, &exc_state->exc_value, &exc_state->exc_traceback);
24784 } else {
24785 __Pyx_Coroutine_ExceptionClear(exc_state);
24786 __Pyx_ExceptionSave(&exc_state->exc_type, &exc_state->exc_value, &exc_state->exc_traceback);
24787 }
24788 #endif
24789 self->is_running = 1;
24790 retval = self->body((PyObject *) self, tstate, value);
24791 self->is_running = 0;
24792 #if CYTHON_USE_EXC_INFO_STACK
24793 exc_state = &self->gi_exc_state;
24794 tstate->exc_info = exc_state->previous_item;
24795 exc_state->previous_item = NULL;
24796 __Pyx_Coroutine_ResetFrameBackpointer(exc_state);
24797 #endif
24798 return retval;
24799 }
24800 static CYTHON_INLINE void __Pyx_Coroutine_ResetFrameBackpointer(__Pyx_ExcInfoStruct *exc_state) {
24801 PyObject *exc_tb = exc_state->exc_traceback;
24802 if (likely(exc_tb)) {
24803 #if CYTHON_COMPILING_IN_PYPY || CYTHON_COMPILING_IN_PYSTON
24804 #else
24805 PyTracebackObject *tb = (PyTracebackObject *) exc_tb;
24806 PyFrameObject *f = tb->tb_frame;
24807 Py_CLEAR(f->f_back);
24808 #endif
24809 }
24810 }
24811 static CYTHON_INLINE
24812 PyObject * __Pyx_Coroutine_MethodReturn(CYTHON_UNUSED PyObject* gen, PyObject *retval) {
24813 if (unlikely(!retval)) {
24814 __Pyx_PyThreadState_declare
24815 __Pyx_PyThreadState_assign
24816 if (!__Pyx_PyErr_Occurred()) {
24817 PyObject *exc = PyExc_StopIteration;
24818 #ifdef __Pyx_AsyncGen_USED
24819 if (__Pyx_AsyncGen_CheckExact(gen))
24820 exc = __Pyx_PyExc_StopAsyncIteration;
24821 #endif
24822 __Pyx_PyErr_SetNone(exc);
24823 }
24824 }
24825 return retval;
24826 }
24827 #if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x03030000 && (defined(__linux__) ||
PY_VERSION_HEX >= 0x030600B3)
24828 static CYTHON_INLINE
24829 PyObject * __Pyx_PyGen_Send(PyGenObject *gen, PyObject *arg) {
24830 #if PY_VERSION_HEX <= 0x030A00A1
24831 return _PyGen_Send(gen, arg);
24832 #else
24833 PyObject *result;
24834 if (PyIter_Send((PyObject*)gen, arg ? arg : Py_None, &result) == PYGEN_RETURN) {
24835 if (PyAsyncGen_CheckExact(gen)) {
24836 assert(result == Py_None);
24837 PyErr_SetNone(PyExc_StopAsyncIteration);
24838 }
24839 else if (result == Py_None) {
24840 PyErr_SetNone(PyExc_StopIteration);
24841 }
24842 else {
24843 _PyGen_SetStopIterationValue(result);
24844 }
24845 Py_CLEAR(result);
24846 }
24847 return result;
24848 #endif
24849 }

```

```

24850 #endif
24851 static CYTHON_INLINE
24852 PyObject *__Pyx_Coroutine_FinishDelegation(__pyx_CoroutineObject *gen) {
24853 PyObject *ret;
24854 PyObject *val = NULL;
24855 __Pyx_Coroutine_Undelegate(gen);
24856 __Pyx_PyGen__FetchStopIterationValue(__Pyx_PyThreadState_Current, &val);
24857 ret = __Pyx_Coroutine_SendEx(gen, val, 0);
24858 Py_XDECREF(val);
24859 return ret;
24860 }
24861 static PyObject *__Pyx_Coroutine_Send(PyObject *self, PyObject *value) {
24862 PyObject *retval;
24863 __pyx_CoroutineObject *gen = (__pyx_CoroutineObject*) self;
24864 PyObject *yf = gen->yieldfrom;
24865 if (unlikely(gen->is_running))
24866 return __Pyx_Coroutine_AlreadyRunningError(gen);
24867 if (yf) {
24868 PyObject *ret;
24869 gen->is_running = 1;
24870 #ifdef __Pyx_Generator_USED
24871 if (__Pyx_Generator_CheckExact(yf)) {
24872 ret = __Pyx_Coroutine_Send(yf, value);
24873 } else
24874 #endif
24875 #ifdef __Pyx_Coroutine_USED
24876 if (__Pyx_Coroutine_Check(yf)) {
24877 ret = __Pyx_Coroutine_Send(yf, value);
24878 } else
24879 #endif
24880 #ifdef __Pyx_AsyncGen_USED
24881 if (__pyx_PyAsyncGenASend_CheckExact(yf)) {
24882 ret = __Pyx_async_gen_asend_send(yf, value);
24883 } else
24884 #endif
24885 #if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x03030000 && (defined(__linux__) ||
PY_VERSION_HEX >= 0x030600B3)
24886 if (PyGen_CheckExact(yf)) {
24887 ret = __Pyx_PyGen_Send((PyGenObject*)yf, value == Py_None ? NULL : value);
24888 } else
24889 #endif
24890 #if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x03050000 && defined(PyCoro_CheckExact)
&& (defined(__linux__) || PY_VERSION_HEX >= 0x030600B3)
24891 if (PyCoro_CheckExact(yf)) {
24892 ret = __Pyx_PyGen_Send((PyGenObject*)yf, value == Py_None ? NULL : value);
24893 } else
24894 #endif
24895 {
24896 if (value == Py_None)
24897 ret = Py_TYPE(yf)->tp_iternext(yf);
24898 else
24899 ret = __Pyx_PyObject_CallMethod1(yf, __pyx_n_s_send, value);
24900 }
24901 gen->is_running = 0;
24902 if (likely(ret)) {
24903 return ret;
24904 }
24905 retval = __Pyx_Coroutine_FinishDelegation(gen);
24906 } else {
24907 retval = __Pyx_Coroutine_SendEx(gen, value, 0);
24908 }
24909 return __Pyx_Coroutine_MethodReturn(self, retval);
24910 }
24911 static int __Pyx_Coroutine_CloseIter(__pyx_CoroutineObject *gen, PyObject *yf) {
24912 PyObject *retval = NULL;
24913 int err = 0;
24914 #ifdef __Pyx_Generator_USED
24915 if (__Pyx_Generator_CheckExact(yf)) {
24916 retval = __Pyx_Coroutine_Close(yf);
24917 if (!retval)
24918 return -1;
24919 } else
24920 #endif
24921 #ifdef __Pyx_Coroutine_USED
24922 if (__Pyx_Coroutine_Check(yf)) {
24923 retval = __Pyx_Coroutine_Close(yf);
24924 if (!retval)
24925 return -1;
24926 } else
24927 #endif
24928 if (__Pyx_CoroutineAwait_CheckExact(yf)) {
24929 retval = __Pyx_CoroutineAwait_Close((__pyx_CoroutineAwaitObject*)yf, NULL);
24930 if (!retval)
24931 return -1;
24932 } else
24933 #endif
24934 #ifdef __Pyx_AsyncGen_USED
24935 if (__pyx_PyAsyncGenASend_CheckExact(yf)) {

```

```

24935 retval = __Pyx_async_gen_asend_close(yf, NULL);
24936 } else
24937 if (__pyx_PyAsyncGenAThrow_CheckExact(yf)) {
24938 retval = __Pyx_async_gen_athrow_close(yf, NULL);
24939 } else
24940 #endif
24941 {
24942 PyObject *meth;
24943 gen->is_running = 1;
24944 meth = __Pyx_PyObject_GetAttrStr(yf, __pyx_n_s_close);
24945 if (unlikely(!meth)) {
24946 if (!PyErr_ExceptionMatches(PyExc_AttributeError)) {
24947 PyErr_WriteUnraisable(yf);
24948 }
24949 PyErr_Clear();
24950 } else {
24951 retval = PyObject_CallFunction(meth, NULL);
24952 Py_DECREF(meth);
24953 if (!retval)
24954 err = -1;
24955 }
24956 gen->is_running = 0;
24957 }
24958 Py_XDECREF(retval);
24959 return err;
24960 }
24961 static PyObject *__Pyx_Generator_Next(PyObject *self) {
24962 __pyx_CoroutineObject *gen = (__pyx_CoroutineObject*) self;
24963 PyObject *yf = gen->yieldfrom;
24964 if (unlikely(gen->is_running))
24965 return __Pyx_Coroutine_AlreadyRunningError(gen);
24966 if (yf) {
24967 PyObject *ret;
24968 gen->is_running = 1;
24969 #ifdef __Pyx_Generator_USED
24970 if (__Pyx_Generator_CheckExact(yf)) {
24971 ret = __Pyx_Generator_Next(yf);
24972 } else
24973 #endif
24974 #if CYTHON_COMPILING_IN_CPYTHON && PY_VERSION_HEX >= 0x03030000 && (defined(__linux__) ||
24975 PY_VERSION_HEX >= 0x030600B3)
24976 if (PyGen_CheckExact(yf)) {
24977 ret = __Pyx_PyGen_Send((PyGenObject*)yf, NULL);
24978 } else
24979 #endif
24980 #ifdef __Pyx_Coroutine_USED
24981 if (__Pyx_Coroutine_Check(yf)) {
24982 ret = __Pyx_Coroutine_Send(yf, Py_None);
24983 } else
24984 #endif
24985 ret = Py_TYPE(yf)->tp_iternext(yf);
24986 gen->is_running = 0;
24987 if (likely(ret)) {
24988 return ret;
24989 }
24990 return __Pyx_Coroutine_FinishDelegation(gen);
24991 }
24992 return __Pyx_Coroutine_SendEx(gen, Py_None, 0);
24993 }
24994 static PyObject *__Pyx_Coroutine_Close_Method(PyObject *self, CYTHON_UNUSED PyObject *arg) {
24995 return __Pyx_Coroutine_Close(self);
24996 }
24997 static PyObject *__Pyx_Coroutine_Close(PyObject *self) {
24998 __pyx_CoroutineObject *gen = (__pyx_CoroutineObject *) self;
24999 PyObject *retval, *raised_exception;
25000 PyObject *yf = gen->yieldfrom;
25001 int err = 0;
25002 if (unlikely(gen->is_running))
25003 return __Pyx_Coroutine_AlreadyRunningError(gen);
25004 if (yf) {
25005 Py_INCREF(yf);
25006 err = __Pyx_Coroutine_CloseIter(gen, yf);
25007 __Pyx_Coroutine_Undelegate(gen);
25008 Py_DECREF(yf);
25009 }
25010 if (err == 0)
25011 PyErr_SetNone(PyExc_GeneratorExit);
25012 retval = __Pyx_Coroutine_SendEx(gen, NULL, 1);
25013 if (unlikely(retval)) {
25014 const char *msg;
25015 Py_DECREF(retval);
25016 if ((0)) {
25017 #ifdef __Pyx_Coroutine_USED
25018 } else if (__Pyx_Coroutine_Check(self)) {
25019 msg = "coroutine ignored GeneratorExit";
25020 #endif
25021 #ifdef __Pyx_AsyncGen_USED

```

```

25021 } else if (__Pyx_AsyncGen_CheckExact(self)) {
25022 #if PY_VERSION_HEX < 0x03060000
25023 msg = "async generator ignored GeneratorExit - might require Python 3.6+ finalisation (PEP
525)";
25024 #else
25025 msg = "async generator ignored GeneratorExit";
25026 #endif
25027 #endif
25028 } else {
25029 msg = "generator ignored GeneratorExit";
25030 }
25031 PyErr_SetString(PyExc_RuntimeError, msg);
25032 return NULL;
25033 }
25034 raised_exception = PyErr_Occurred();
25035 if (likely(!raised_exception || __Pyx_PyErr_GivenExceptionMatches2(raised_exception,
PyExc_GeneratorExit, PyExc_StopIteration))) {
25036 if (raised_exception) PyErr_Clear();
25037 Py_INCREF(Py_None);
25038 return Py_None;
25039 }
25040 return NULL;
25041 }
25042 static PyObject *__Pyx_Coroutine_Throw(PyObject *self, PyObject *typ, PyObject *val, PyObject *tb,
PyObject *args, int close_on_genexit) {
25043 __pyx_CoroutineObject *gen = (__pyx_CoroutineObject *) self;
25044 PyObject *yf = gen->yieldfrom;
25045 if (unlikely(gen->is_running))
25046 return __Pyx_Coroutine_AlreadyRunningError(gen);
25047 if (yf) {
25048 PyObject *ret;
25049 Py_INCREF(yf);
25050 if (__Pyx_PyErr_GivenExceptionMatches(typ, PyExc_GeneratorExit) && close_on_genexit) {
25051 int err = __Pyx_Coroutine_CloseIter(gen, yf);
25052 Py_DECREF(yf);
25053 __Pyx_Coroutine_Undelegate(gen);
25054 if (err < 0)
25055 return __Pyx_Coroutine_MethodReturn(self, __Pyx_Coroutine_SendEx(gen, NULL, 0));
25056 goto throw_here;
25057 }
25058 gen->is_running = 1;
25059 if (0
#ifdef __Pyx_Generator_USED
|| __Pyx_Generator_CheckExact(yf)
#endif
#ifdef __Pyx_Coroutine_USED
|| __Pyx_Coroutine_Check(yf)
#endif
) {
25060 ret = __Pyx_Coroutine_Throw(yf, typ, val, tb, args, close_on_genexit);
25061 #ifdef __Pyx_Coroutine_USED
25062 } else if (__Pyx_CoroutineAwait_CheckExact(yf)) {
25063 ret = __Pyx_Coroutine_Throw(((__pyx_CoroutineAwaitObject*)yf)->coroutine, typ, val, tb,
args, close_on_genexit);
25064 #endif
25065 } else {
25066 PyObject *meth = __Pyx_PyObject_GetAttrStr(yf, __pyx_n_s_throw);
25067 if (unlikely(!meth)) {
25068 Py_DECREF(yf);
25069 if (!PyErr_ExceptionMatches(PyExc_AttributeError)) {
25070 gen->is_running = 0;
25071 return NULL;
25072 }
25073 PyErr_Clear();
25074 __Pyx_Coroutine_Undelegate(gen);
25075 gen->is_running = 0;
25076 goto throw_here;
25077 }
25078 if (likely(args)) {
25079 ret = PyObject_CallObject(meth, args);
25080 } else {
25081 ret = PyObject_CallFunctionObjArgs(meth, typ, val, tb, NULL);
25082 }
25083 Py_DECREF(meth);
25084 gen->is_running = 0;
25085 Py_DECREF(yf);
25086 if (!ret) {
25087 ret = __Pyx_Coroutine_FinishDelegation(gen);
25088 }
25089 return __Pyx_Coroutine_MethodReturn(self, ret);
25090 }
25091 }
25092 throw_here:
25093 __Pyx_Raise(typ, val, tb, NULL);
25094 return __Pyx_Coroutine_MethodReturn(self, __Pyx_Coroutine_SendEx(gen, NULL, 0));
25095 }
25096 static PyObject *__Pyx_Coroutine_Throw(PyObject *self, PyObject *args) {

```

```

25105 PyObject *typ;
25106 PyObject *val = NULL;
25107 PyObject *tb = NULL;
25108 if (!PyArg_UnpackTuple(args, (char *)"throw", 1, 3, &typ, &val, &tb))
25109 return NULL;
25110 return __Pyx_Coroutine_Throw(self, typ, val, tb, args, 1);
25111 }
25112 static CYTHON_INLINE int __Pyx_Coroutine_traverse_excstate(__Pyx_ExcInfoStruct *exc_state, visitproc
 visit, void *arg) {
25113 Py_VISIT(exc_state->exc_type);
25114 Py_VISIT(exc_state->exc_value);
25115 Py_VISIT(exc_state->exc_traceback);
25116 return 0;
25117 }
25118 static int __Pyx_Coroutine_traverse(__pyx_CoroutineObject *gen, visitproc visit, void *arg) {
25119 Py_VISIT(gen->closure);
25120 Py_VISIT(gen->classobj);
25121 Py_VISIT(gen->yieldfrom);
25122 return __Pyx_Coroutine_traverse_excstate(&gen->gi_exc_state, visit, arg);
25123 }
25124 static int __Pyx_Coroutine_clear(PyObject *self) {
25125 __pyx_CoroutineObject *gen = (__pyx_CoroutineObject *) self;
25126 Py_CLEAR(gen->closure);
25127 Py_CLEAR(gen->classobj);
25128 Py_CLEAR(gen->yieldfrom);
25129 __Pyx_Coroutine_ExceptionClear(&gen->gi_exc_state);
25130 #ifdef __Pyx_AsyncGen_USED
25131 if (__Pyx_AsyncGen_CheckExact(self)) {
25132 Py_CLEAR(((__pyx_PyAsyncGenObject*)gen)->ag_finalizer);
25133 }
25134 #endif
25135 Py_CLEAR(gen->gi_code);
25136 Py_CLEAR(gen->gi_frame);
25137 Py_CLEAR(gen->gi_name);
25138 Py_CLEAR(gen->gi_qualname);
25139 Py_CLEAR(gen->gi_modulename);
25140 return 0;
25141 }
25142 static void __Pyx_Coroutine_dealloc(PyObject *self) {
25143 __pyx_CoroutineObject *gen = (__pyx_CoroutineObject *) self;
25144 PyObject_GC_UnTrack(gen);
25145 if (gen->gi_weakreflist != NULL)
25146 PyObject_ClearWeakRefs(self);
25147 if (gen->resume_label >= 0) {
25148 PyObject_GC_Track(self);
25149 #if PY_VERSION_HEX >= 0x030400a1 && CYTHON_USE_TP_FINALIZE
25150 if (PyObject_CallFinalizerFromDealloc(self))
25151 #else
25152 Py_TYPE(gen)->tp_del(self);
25153 if (Py_REFCNT(self) > 0)
25154 #endif
25155 {
25156 return;
25157 }
25158 PyObject_GC_UnTrack(self);
25159 }
25160 #ifdef __Pyx_AsyncGen_USED
25161 if (__Pyx_AsyncGen_CheckExact(self)) {
25162 /* We have to handle this case for asynchronous generators
25163 right here, because this code has to be between UNTRACK
25164 and GC_Del. */
25165 Py_CLEAR(((__pyx_PyAsyncGenObject*)self)->ag_finalizer);
25166 }
25167 #endif
25168 __Pyx_Coroutine_clear(self);
25169 PyObject_GC_Del(gen);
25170 }
25171 static void __Pyx_Coroutine_del(PyObject *self) {
25172 PyObject *error_type, *error_value, *error_traceback;
25173 __pyx_CoroutineObject *gen = (__pyx_CoroutineObject *) self;
25174 __Pyx_PyThreadState_declare
25175 if (gen->resume_label < 0) {
25176 return;
25177 }
25178 #if !CYTHON_USE_TP_FINALIZE
25179 assert(self->ob_refcnt == 0);
25180 __Pyx_SET_REFCNT(self, 1);
25181 #endif
25182 __Pyx_PyThreadState_assign
25183 __Pyx_ErrFetch(&error_type, &error_value, &error_traceback);
25184 #ifdef __Pyx_AsyncGen_USED
25185 if (__Pyx_AsyncGen_CheckExact(self)) {
25186 __pyx_PyAsyncGenObject *agen = (__pyx_PyAsyncGenObject*)self;
25187 PyObject *finalizer = agen->ag_finalizer;
25188 if (finalizer && !agen->ag_closed) {
25189 PyObject *res = __Pyx_PyObject_CallOneArg(finalizer, self);
25190 if (unlikely(!res)) {

```



```

25191 PyErr_WriteUnraisable(self);
25192 } else {
25193 Py_DECREF(res);
25194 }
25195 __Pyx_ErrRestore(error_type, error_value, error_traceback);
25196 return;
25197 }
25198 }
25199 #endif
25200 if (unlikely(gen->resume_label == 0 && !error_value)) {
25201 #ifdef __Pyx_Coroutine_USED
25202 #ifdef __Pyx_Generator_USED
25203 if (!__Pyx_Generator_CheckExact(self))
25204 #endif
25205 {
25206 PyObject_GC_UnTrack(self);
25207 #if PY_MAJOR_VERSION >= 3 || defined(PyErr_WarnFormat)
25208 if (unlikely(PyErr_WarnFormat(PyExc_RuntimeWarning, 1, "coroutine '%.50S' was never awaited",
gen->gi_qualname) < 0))
25209 PyErr_WriteUnraisable(self);
25210 #else
25211 {PyObject *msg;
25212 char *cmsg;
25213 #if CYTHON_COMPILING_IN_PYPY
25214 msg = NULL;
25215 cmsg = (char*) "coroutine was never awaited";
25216 #else
25217 char *cname;
25218 PyObject *qualname;
25219 qualname = gen->gi_qualname;
25220 cname = PyString_AS_STRING(qualname);
25221 msg = PyString_FromFormat("coroutine '%.50s' was never awaited", cname);
25222 if (unlikely(!msg)) {
25223 PyErr_Clear();
25224 cmsg = (char*) "coroutine was never awaited";
25225 } else {
25226 cmsg = PyString_AS_STRING(msg);
25227 }
25228 #endif
25229 if (unlikely(PyErr_WarnEx(PyExc_RuntimeWarning, cmsg, 1) < 0))
25230 PyErr_WriteUnraisable(self);
25231 Py_XDECREF(msg);}
25232 #endif
25233 PyObject_GC_Track(self);
25234 }
25235 #endif
25236 } else {
25237 PyObject *res = __Pyx_Coroutine_Close(self);
25238 if (unlikely(!res)) {
25239 if (PyErr_Occurred())
25240 PyErr_WriteUnraisable(self);
25241 } else {
25242 Py_DECREF(res);
25243 }
25244 }
25245 __Pyx_ErrRestore(error_type, error_value, error_traceback);
25246 #if !CYTHON_USE_TP_FINALIZE
25247 assert(Py_REFCNT(self) > 0);
25248 if (--self->ob_refcnt == 0) {
25249 return;
25250 }
25251 {
25252 Py_ssize_t refcnt = Py_REFCNT(self);
25253 __Pyx_NewReference(self);
25254 __Pyx_SET_REFCNT(self, refcnt);
25255 }
25256 #if CYTHON_COMPILING_IN_CPYTHON
25257 assert(PyType_IS_GC(Py_TYPE(self)) &&
25258 __Pyx_AS_GC(self)->gc.gc_refs != __PyGC_REFS_UNTRACKED);
25259 __Pyx_DEC_REFTOTAL;
25260 #endif
25261 #ifdef COUNT_ALLOCS
25262 --Py_TYPE(self)->tp_frees;
25263 --Py_TYPE(self)->tp_allocs;
25264 #endif
25265 #endif
25266 }
25267 static PyObject *
25268 __Pyx_Coroutine_get_name(__pyx_CoroutineObject *self, CYTHON_UNUSED void *context)
25269 {
25270 PyObject *name = self->gi_name;
25271 if (unlikely(!name)) name = Py_None;
25272 Py_INCREF(name);
25273 return name;
25274 }
25275 static int
25276 __Pyx_Coroutine_set_name(__pyx_CoroutineObject *self, PyObject *value, CYTHON_UNUSED void *context)

```

```

25277 {
25278 PyObject *tmp;
25279 #if PY_MAJOR_VERSION >= 3
25280 if (unlikely(value == NULL || !PyUnicode_Check(value)))
25281 #else
25282 if (unlikely(value == NULL || !PyString_Check(value)))
25283 #endif
25284 {
25285 PyErr_SetString(PyExc_TypeError,
25286 "__name__ must be set to a string object");
25287 return -1;
25288 }
25289 tmp = self->gi_name;
25290 Py_INCREF(value);
25291 self->gi_name = value;
25292 Py_XDECREF(tmp);
25293 return 0;
25294 }
25295 static PyObject *
25296 __Pyx_Coroutine_get_qualname(__pyx_CoroutineObject *self, CYTHON_UNUSED void *context)
25297 {
25298 PyObject *name = self->gi_qualname;
25299 if (unlikely(!name)) name = Py_None;
25300 Py_INCREF(name);
25301 return name;
25302 }
25303 static int
25304 __Pyx_Coroutine_set_qualname(__pyx_CoroutineObject *self, PyObject *value, CYTHON_UNUSED void
25305 *context)
25306 {
25307 PyObject *tmp;
25308 #if PY_MAJOR_VERSION >= 3
25309 if (unlikely(value == NULL || !PyUnicode_Check(value)))
25310 #else
25311 if (unlikely(value == NULL || !PyString_Check(value)))
25312 #endif
25313 {
25314 PyErr_SetString(PyExc_TypeError,
25315 "__qualname__ must be set to a string object");
25316 return -1;
25317 }
25318 tmp = self->gi_qualname;
25319 Py_INCREF(value);
25320 self->gi_qualname = value;
25321 Py_XDECREF(tmp);
25322 return 0;
25323 }
25324 static PyObject *
25325 __Pyx_Coroutine_get_frame(__pyx_CoroutineObject *self, CYTHON_UNUSED void *context)
25326 {
25327 PyObject *frame = self->gi_frame;
25328 if (!frame) {
25329 if (unlikely(!self->gi_code)) {
25330 Py_RETURN_NONE;
25331 }
25332 frame = (PyObject *) PyFrame_New(
25333 PyThreadState_Get(), /*PyThreadState *tstate,*/
25334 (PyCodeObject*) self->gi_code, /*PyCodeObject *code,*/
25335 __pyx_d, /*PyObject *globals,*/
25336 0 /*PyObject *locals*/
25337);
25338 if (unlikely(!frame))
25339 return NULL;
25340 self->gi_frame = frame;
25341 }
25342 Py_INCREF(frame);
25343 return frame;
25344 }
25345 static __pyx_CoroutineObject * __Pyx__Coroutine_New(
25346 PyTypeObject* type, __pyx_coroutine_body_t body, PyObject *code, PyObject *closure,
25347 PyObject *name, PyObject *qualname, PyObject *module_name) {
25348 __pyx_CoroutineObject *gen = PyObject_GC_New(__pyx_CoroutineObject, type);
25349 if (unlikely(!gen))
25350 return NULL;
25351 return __Pyx__Coroutine_NewInit(gen, body, code, closure, name, qualname, module_name);
25352 }
25353 static __pyx_CoroutineObject * __Pyx__Coroutine_NewInit(
25354 __pyx_CoroutineObject *gen, __pyx_coroutine_body_t body, PyObject *code, PyObject
25355 *closure,
25356 PyObject *name, PyObject *qualname, PyObject *module_name) {
25357 gen->body = body;
25358 gen->closure = closure;
25359 Py_XINCREF(closure);
25360 gen->is_running = 0;
25361 gen->resume_label = 0;
25362 gen->classobj = NULL;
25363 gen->yieldfrom = NULL;

```

```

25362 gen->gi_exc_state.exc_type = NULL;
25363 gen->gi_exc_state.exc_value = NULL;
25364 gen->gi_exc_state.exc_traceback = NULL;
25365 #if CYTHON_USE_EXC_INFO_STACK
25366 gen->gi_exc_state.previous_item = NULL;
25367 #endif
25368 gen->gi_weakreflist = NULL;
25369 Py_XINCREF(qualname);
25370 gen->gi_qualname = qualname;
25371 Py_XINCREF(name);
25372 gen->gi_name = name;
25373 Py_XINCREF(module_name);
25374 gen->gi_modulename = module_name;
25375 Py_XINCREF(code);
25376 gen->gi_code = code;
25377 gen->gi_frame = NULL;
25378 PyObject_GC_Track(gen);
25379 return gen;
25380 }
25381
25382 /* PatchModuleWithCoroutine */
25383 static PyObject* __Pyx_Coroutine_patch_module(PyObject* module, const char* py_code) {
25384 #if defined(__Pyx_Generator_USED) || defined(__Pyx_Coroutine_USED)
25385 int result;
25386 PyObject *globals, *result_obj;
25387 globals = PyDict_New(); if (unlikely(!globals)) goto ignore;
25388 result = PyDict_SetItemString(globals, "_cython_coroutine_type",
25389 #ifdef __Pyx_Coroutine_USED
25390 (PyObject*)__pyx_CoroutineType);
25391 #else
25392 Py_None);
25393 #endif
25394 if (unlikely(result < 0)) goto ignore;
25395 result = PyDict_SetItemString(globals, "_cython_generator_type",
25396 #ifdef __Pyx_Generator_USED
25397 (PyObject*)__pyx_GeneratorType);
25398 #else
25399 Py_None);
25400 #endif
25401 if (unlikely(result < 0)) goto ignore;
25402 if (unlikely(PyDict_SetItemString(globals, "_module", module) < 0)) goto ignore;
25403 if (unlikely(PyDict_SetItemString(globals, "__builtins__", __pyx_b) < 0)) goto ignore;
25404 result_obj = PyRun_String(py_code, Py_file_input, globals, globals);
25405 if (unlikely(!result_obj)) goto ignore;
25406 Py_DECREF(result_obj);
25407 Py_DECREF(globals);
25408 return module;
25409 ignore:
25410 Py_XDECREF(globals);
25411 PyErr_WriteUnraisable(module);
25412 if (unlikely(PyErr_WarnEx(PyExc_RuntimeWarning, "Cython module failed to patch module with custom
type", 1) < 0)) {
25413 Py_DECREF(module);
25414 module = NULL;
25415 }
25416 #else
25417 py_code++;
25418 #endif
25419 return module;
25420 }
25421
25422 /* PatchGeneratorABC */
25423 #ifndef CYTHON_REGISTER_ABCS
25424 #define CYTHON_REGISTER_ABCS 1
25425 #endif
25426 #if defined(__Pyx_Generator_USED) || defined(__Pyx_Coroutine_USED)
25427 static PyObject* __Pyx_patch_abc_module(PyObject *module);
25428 static PyObject* __Pyx_patch_abc_module(PyObject *module) {
25429 module = __Pyx_Coroutine_patch_module(
25430 module, ""
25431 "if _cython_generator_type is not None:\n"
25432 " try: Generator = _module.Generator\n"
25433 " except AttributeError: pass\n"
25434 " else: Generator.register(_cython_generator_type)\n"
25435 "if _cython_coroutine_type is not None:\n"
25436 " try: Coroutine = _module.Coroutine\n"
25437 " except AttributeError: pass\n"
25438 " else: Coroutine.register(_cython_coroutine_type)\n"
25439);
25440 return module;
25441 }
25442 #endif
25443 static int __Pyx_patch_abc(void) {
25444 #if defined(__Pyx_Generator_USED) || defined(__Pyx_Coroutine_USED)
25445 static int abc_patched = 0;
25446 if (CYTHON_REGISTER_ABCS && !abc_patched) {
25447 PyObject *module;

```

```

25448 module = PyImport_ImportModule((PY_MAJOR_VERSION >= 3) ? "collections.abc" : "collections");
25449 if (!module) {
25450 PyErr_WriteUnraisable(NULL);
25451 if (unlikely(PyErr_WarnEx(PyExc_RuntimeWarning,
25452 ((PY_MAJOR_VERSION >= 3) ?
25453 "Cython module failed to register with collections.abc module" :
25454 "Cython module failed to register with collections module"), 1) < 0)) {
25455 return -1;
25456 }
25457 } else {
25458 module = __Pyx_patch_abc_module(module);
25459 abc_patched = 1;
25460 if (unlikely(!module))
25461 return -1;
25462 Py_DECREF(module);
25463 }
25464 module = PyImport_ImportModule("backports_abc");
25465 if (module) {
25466 module = __Pyx_patch_abc_module(module);
25467 Py_XDECREF(module);
25468 }
25469 if (!module) {
25470 PyErr_Clear();
25471 }
25472 }
25473 #else
25474 if ((0)) __Pyx_Coroutine_patch_module(NULL, NULL);
25475 #endif
25476 return 0;
25477 }
25478
25479 /* Generator */
25480 static PyMethodDef __pyx_Generator_methods[] = {
25481 {"send", (PyCFunction) __Pyx_Coroutine_Send, METH_O,
25482 (char*) PyDoc_STR("send(arg) -> send 'arg' into generator,\nreturn next yielded value or raise
25483 StopIteration.")},
25484 {"throw", (PyCFunction) __Pyx_Coroutine_Throw, METH_VARARGS,
25485 (char*) PyDoc_STR("throw(typ[,val[,tb]]) -> raise exception in generator,\nreturn next yielded
25486 value or raise StopIteration.")},
25487 {"close", (PyCFunction) __Pyx_Coroutine_Close_Method, METH_NOARGS,
25488 (char*) PyDoc_STR("close() -> raise GeneratorExit inside generator.")},
25489 {0, 0, 0, 0}
25490 };
25491 static PyMemberDef __pyx_Generator_memberlist[] = {
25492 {(char *) "gi_running", T_BOOL, offsetof(__pyx_CoroutineObject, is_running), READONLY, NULL},
25493 {(char *) "gi_yieldfrom", T_OBJECT, offsetof(__pyx_CoroutineObject, yieldfrom), READONLY,
25494 (char*) PyDoc_STR("object being iterated by 'yield from', or None")},
25495 {(char *) "gi_code", T_OBJECT, offsetof(__pyx_CoroutineObject, gi_code), READONLY, NULL},
25496 {0, 0, 0, 0, 0}
25497 };
25498 static PyGetSetDef __pyx_Generator_getsets[] = {
25499 {(char *) "__name__", (getter) __Pyx_Coroutine_get_name, (setter) __Pyx_Coroutine_set_name,
25500 (char*) PyDoc_STR("name of the generator"), 0},
25501 {(char *) "__qualname__", (getter) __Pyx_Coroutine_get_qualname,
25502 (setter) __Pyx_Coroutine_set_qualname,
25503 (char*) PyDoc_STR("qualified name of the generator"), 0},
25504 {(char *) "gi_frame", (getter) __Pyx_Coroutine_get_frame, NULL,
25505 (char*) PyDoc_STR("Frame of the generator"), 0},
25506 {0, 0, 0, 0, 0}
25507 };
25508 static PyTypeObject __pyx_GeneratorType_type = {
25509 PyVarObject_HEAD_INIT(0, 0)
25510 "generator",
25511 sizeof(__pyx_CoroutineObject),
25512 0,
25513 (destructor) __Pyx_Coroutine_dealloc,
25514 0,
25515 0,
25516 0,
25517 0,
25518 0,
25519 0,
25520 0,
25521 0,
25522 0,
25523 0,
25524 0,
25525 Py_TPFLAGS_DEFAULT | Py_TPFLAGS_HAVE_GC | Py_TPFLAGS_HAVE_FINALIZE,
25526 0,
25527 (traverseproc) __Pyx_Coroutine_traverse,
25528 0,
25529 0,
25530 offsetof(__pyx_CoroutineObject, gi_weakreflist),
25531 0,

```

```

25532 (iternextfunc) __Pyx_Generator_Next,
25533 __pyx_Generator_methods,
25534 __pyx_Generator_memberlist,
25535 __pyx_Generator_getsets,
25536 0,
25537 0,
25538 0,
25539 0,
25540 0,
25541 0,
25542 0,
25543 0,
25544 0,
25545 0,
25546 0,
25547 0,
25548 0,
25549 0,
25550 0,
25551 #if CYTHON_USE_TP_FINALIZE
25552 0,
25553 #else
25554 __Pyx_Coroutine_del,
25555 #endif
25556 0,
25557 #if CYTHON_USE_TP_FINALIZE
25558 __Pyx_Coroutine_del,
25559 #elif PY_VERSION_HEX >= 0x030400a1
25560 0,
25561 #endif
25562 #if PY_VERSION_HEX >= 0x030800b1 && (!CYTHON_COMPILING_IN_PYPY || PYPY_VERSION_NUM >= 0x07030800)
25563 0,
25564 #endif
25565 #if PY_VERSION_HEX >= 0x030800b4 && PY_VERSION_HEX < 0x03090000
25566 0,
25567 #endif
25568 #if CYTHON_COMPILING_IN_PYPY && PY_VERSION_HEX >= 0x03090000
25569 0,
25570 #endif
25571 };
25572 static int __pyx_Generator_init(void) {
25573 __pyx_GeneratorType_type.tp_getattro = __Pyx_PyObject_GenericGetAttrNoDict;
25574 __pyx_GeneratorType_type.tp_iter = PyObject_SelfIter;
25575 __pyx_GeneratorType = __Pyx_FetchCommonType(&__pyx_GeneratorType_type);
25576 if (unlikely(!__pyx_GeneratorType)) {
25577 return -1;
25578 }
25579 return 0;
25580 }
25581
25582 /* CheckBinaryVersion */
25583 static int __Pyx_check_binary_version(void) {
25584 char ctversion[4], rtversion[4];
25585 PyOS_snprintf(ctversion, 4, "%d.%d", PY_MAJOR_VERSION, PY_MINOR_VERSION);
25586 PyOS_snprintf(rtversion, 4, "%s", Py_GetVersion());
25587 if (ctversion[0] != rtversion[0] || ctversion[2] != rtversion[2]) {
25588 char message[200];
25589 PyOS_snprintf(message, sizeof(message),
25590 "compiletime version %s of module '%.100s' "
25591 "does not match runtime version %s",
25592 ctversion, __Pyx_MODULE_NAME, rtversion);
25593 return PyErr_WarnEx(NULL, message, 1);
25594 }
25595 return 0;
25596 }
25597
25598 /* InitStrings */
25599 static int __Pyx_InitStrings(__Pyx_StringTabEntry *t) {
25600 while (t->p) {
25601 #if PY_MAJOR_VERSION < 3
25602 if (t->is_unicode) {
25603 *t->p = PyUnicode_DecodeUTF8(t->s, t->n - 1, NULL);
25604 } else if (t->intern) {
25605 *t->p = PyString_InternFromString(t->s);
25606 } else {
25607 *t->p = PyString_FromStringAndSize(t->s, t->n - 1);
25608 }
25609 } else
25610 if (t->is_unicode | t->is_str) {
25611 if (t->intern) {
25612 *t->p = PyUnicode_InternFromString(t->s);
25613 } else if (t->encoding) {
25614 *t->p = PyUnicode_Decode(t->s, t->n - 1, t->encoding, NULL);
25615 } else {
25616 *t->p = PyUnicode_FromStringAndSize(t->s, t->n - 1);
25617 }
25618 } else {

```

```

25619 *t->p = PyBytes_FromStringAndSize(t->s, t->n - 1);
25620 }
25621 #endif
25622 if (!*t->p)
25623 return -1;
25624 if (PyObject_Hash(*t->p) == -1)
25625 return -1;
25626 ++t;
25627 }
25628 return 0;
25629 }
25630
25631 static CYTHON_INLINE PyObject* __Pyx_PyUnicode_FromString(const char* c_str) {
25632 return __Pyx_PyUnicode_FromStringAndSize(c_str, (Py_ssize_t)strlen(c_str));
25633 }
25634 static CYTHON_INLINE const char* __Pyx_PyObject_AsString(PyObject* o) {
25635 Py_ssize_t ignore;
25636 return __Pyx_PyObject_AsStringAndSize(o, &ignore);
25637 }
25638 #if __PYX_DEFAULT_STRING_ENCODING_IS_ASCII || __PYX_DEFAULT_STRING_ENCODING_IS_DEFAULT
25639 #if !CYTHON_PEP393_ENABLED
25640 static const char* __Pyx_PyUnicode_AsStringAndSize(PyObject* o, Py_ssize_t *length) {
25641 char* defenc_c;
25642 PyObject* defenc = _PyUnicode_AsDefaultEncodedString(o, NULL);
25643 if (!defenc) return NULL;
25644 defenc_c = PyBytes_AS_STRING(defenc);
25645 #if __PYX_DEFAULT_STRING_ENCODING_IS_ASCII
25646 {
25647 char* end = defenc_c + PyBytes_GET_SIZE(defenc);
25648 char* c;
25649 for (c = defenc_c; c < end; c++) {
25650 if ((unsigned char) (*c) >= 128) {
25651 PyUnicode_AsASCIIString(o);
25652 return NULL;
25653 }
25654 }
25655 }
25656 #endif
25657 *length = PyBytes_GET_SIZE(defenc);
25658 return defenc_c;
25659 }
25660 #else
25661 static CYTHON_INLINE const char* __Pyx_PyUnicode_AsStringAndSize(PyObject* o, Py_ssize_t *length) {
25662 if (unlikely(__Pyx_PyUnicode_READY(o) == -1)) return NULL;
25663 #if __PYX_DEFAULT_STRING_ENCODING_IS_ASCII
25664 if (likely(PyUnicode_IS_ASCII(o))) {
25665 *length = PyUnicode_GET_LENGTH(o);
25666 return PyUnicode_AsUTF8(o);
25667 } else {
25668 PyUnicode_AsASCIIString(o);
25669 return NULL;
25670 }
25671 #else
25672 return PyUnicode_AsUTF8AndSize(o, length);
25673 #endif
25674 }
25675 #endif
25676 #endif
25677 static CYTHON_INLINE const char* __Pyx_PyObject_AsStringAndSize(PyObject* o, Py_ssize_t *length) {
25678 #if __PYX_DEFAULT_STRING_ENCODING_IS_ASCII || __PYX_DEFAULT_STRING_ENCODING_IS_DEFAULT
25679 if (
25680 #if PY_MAJOR_VERSION < 3 && __PYX_DEFAULT_STRING_ENCODING_IS_ASCII
25681 __Pyx_sys_getdefaultencoding_not_ascii &&
25682 #endif
25683 PyUnicode_Check(o)) {
25684 return __Pyx_PyUnicode_AsStringAndSize(o, length);
25685 } else
25686 #endif
25687 #if (!CYTHON_COMPILING_IN_PYPY) || (defined(PyByteArray_AS_STRING) && defined(PyByteArray_GET_SIZE))
25688 if (PyByteArray_Check(o)) {
25689 *length = PyByteArray_GET_SIZE(o);
25690 return PyByteArray_AS_STRING(o);
25691 } else
25692 #endif
25693 {
25694 char* result;
25695 int r = PyBytes_AsStringAndSize(o, &result, length);
25696 if (unlikely(r < 0)) {
25697 return NULL;
25698 } else {
25699 return result;
25700 }
25701 }
25702 }
25703 static CYTHON_INLINE int __Pyx_PyObject_IsTrue(PyObject* x) {
25704 int is_true = x == Py_True;
25705 if (is_true | (x == Py_False) | (x == Py_None)) return is_true;

```

```

25706 else return PyObject_IsTrue(x);
25707 }
25708 static CYTHON_INLINE int __Pyx_PyObject_IsTrueAndDecref(PyObject* x) {
25709 int retval;
25710 if (unlikely(!x)) return -1;
25711 retval = __Pyx_PyObject_IsTrue(x);
25712 Py_DECREF(x);
25713 return retval;
25714 }
25715 static PyObject* __Pyx_PyNumber_IntOrLongWrongResultType(PyObject* result, const char* type_name) {
25716 #if PY_MAJOR_VERSION >= 3
25717 if (PyLong_Check(result)) {
25718 if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
25719 "__int__ returned non-int (type %.200s). "
25720 "The ability to return an instance of a strict subclass of int "
25721 "is deprecated, and may be removed in a future version of Python.",
25722 Py_TYPE(result)->tp_name)) {
25723 Py_DECREF(result);
25724 return NULL;
25725 }
25726 return result;
25727 }
25728 #endif
25729 PyErr_Format(PyExc_TypeError,
25730 "%s returned non-%.4s (type %.200s)",
25731 type_name, type_name, Py_TYPE(result)->tp_name);
25732 Py_DECREF(result);
25733 return NULL;
25734 }
25735 static CYTHON_INLINE PyObject* __Pyx_PyNumber_IntOrLong(PyObject* x) {
25736 #if CYTHON_USE_TYPE_SLOTS
25737 PyNumberMethods *m;
25738 #endif
25739 const char *name = NULL;
25740 PyObject *res = NULL;
25741 #if PY_MAJOR_VERSION < 3
25742 if (likely(PyInt_Check(x) || PyLong_Check(x)))
25743 #else
25744 if (likely(PyLong_Check(x)))
25745 #endif
25746 return __Pyx_NewRef(x);
25747 #if CYTHON_USE_TYPE_SLOTS
25748 m = Py_TYPE(x)->tp_as_number;
25749 #if PY_MAJOR_VERSION < 3
25750 if (m && m->nb_int) {
25751 name = "int";
25752 res = m->nb_int(x);
25753 }
25754 else if (m && m->nb_long) {
25755 name = "long";
25756 res = m->nb_long(x);
25757 }
25758 #else
25759 if (likely(m && m->nb_int)) {
25760 name = "int";
25761 res = m->nb_int(x);
25762 }
25763 #endif
25764 #else
25765 if (!PyBytes_CheckExact(x) && !PyUnicode_CheckExact(x)) {
25766 res = PyNumber_Int(x);
25767 }
25768 #endif
25769 if (likely(res)) {
25770 #if PY_MAJOR_VERSION < 3
25771 if (unlikely(!PyInt_Check(res) && !PyLong_Check(res))) {
25772 #else
25773 if (unlikely(!PyLong_CheckExact(res))) {
25774 #endif
25775 return __Pyx_PyNumber_IntOrLongWrongResultType(res, name);
25776 }
25777 }
25778 else if (!PyErr_Occurred()) {
25779 PyErr_SetString(PyExc_TypeError,
25780 "an integer is required");
25781 }
25782 return res;
25783 }
25784 static CYTHON_INLINE Py_ssize_t __Pyx_PyIndex_AsSsize_t(PyObject* b) {
25785 Py_ssize_t ival;
25786 PyObject *x;
25787 #if PY_MAJOR_VERSION < 3
25788 if (likely(PyInt_CheckExact(b))) {
25789 if (sizeof(Py_ssize_t) >= sizeof(long))
25790 return PyInt_AS_LONG(b);
25791 else
25792 return PyInt_AsSsize_t(b);

```

```

25793 }
25794 #endif
25795 if (likely(PyLong_CheckExact(b))) {
25796 #if CYTHON_USE_PYLONG_INTERNALS
25797 const digit* digits = ((PyLongObject*)b)->ob_digit;
25798 const Py_ssize_t size = Py_SIZE(b);
25799 if (likely(__Pyx_sst_abs(size) <= 1)) {
25800 ival = likely(size) ? digits[0] : 0;
25801 if (size == -1) ival = -ival;
25802 return ival;
25803 } else {
25804 switch (size) {
25805 case 2:
25806 if (8 * sizeof(Py_ssize_t) > 2 * PyLong_SHIFT) {
25807 return (Py_ssize_t) (((size_t)digits[1]) « PyLong_SHIFT) | (size_t)digits[0]);
25808 }
25809 break;
25810 case -2:
25811 if (8 * sizeof(Py_ssize_t) > 2 * PyLong_SHIFT) {
25812 return -(Py_ssize_t) (((size_t)digits[1]) « PyLong_SHIFT) | (size_t)digits[0]);
25813 }
25814 break;
25815 case 3:
25816 if (8 * sizeof(Py_ssize_t) > 3 * PyLong_SHIFT) {
25817 return (Py_ssize_t) (((size_t)digits[2]) « PyLong_SHIFT) | (size_t)digits[1]) «
PyLong_SHIFT) | (size_t)digits[0]);
25818 }
25819 break;
25820 case -3:
25821 if (8 * sizeof(Py_ssize_t) > 3 * PyLong_SHIFT) {
25822 return -(Py_ssize_t) (((size_t)digits[2]) « PyLong_SHIFT) | (size_t)digits[1]) «
PyLong_SHIFT) | (size_t)digits[0]);
25823 }
25824 break;
25825 case 4:
25826 if (8 * sizeof(Py_ssize_t) > 4 * PyLong_SHIFT) {
25827 return (Py_ssize_t) (((size_t)digits[3]) « PyLong_SHIFT) | (size_t)digits[2]) «
PyLong_SHIFT) | (size_t)digits[1]) « PyLong_SHIFT) | (size_t)digits[0]);
25828 }
25829 break;
25830 case -4:
25831 if (8 * sizeof(Py_ssize_t) > 4 * PyLong_SHIFT) {
25832 return -(Py_ssize_t) (((size_t)digits[3]) « PyLong_SHIFT) | (size_t)digits[2]) «
PyLong_SHIFT) | (size_t)digits[1]) « PyLong_SHIFT) | (size_t)digits[0]);
25833 }
25834 break;
25835 }
25836 }
25837 #endif
25838 return PyLong_AsSsize_t(b);
25839 }
25840 x = PyNumber_Index(b);
25841 if (!x) return -1;
25842 ival = PyInt_AsSsize_t(x);
25843 Py_DECREF(x);
25844 return ival;
25845 }
25846 static CYTHON_INLINE Py_hash_t __Pyx_PyIndex_AsHash_t(PyObject* o) {
25847 if (sizeof(Py_hash_t) == sizeof(Py_ssize_t)) {
25848 return (Py_hash_t) __Pyx_PyIndex_AsSsize_t(o);
25849 #if PY_MAJOR_VERSION < 3
25850 } else if (likely(PyInt_CheckExact(o))) {
25851 return PyInt_AS_LONG(o);
25852 #endif
25853 } else {
25854 Py_ssize_t ival;
25855 PyObject *x;
25856 x = PyNumber_Index(o);
25857 if (!x) return -1;
25858 ival = PyInt_AsLong(x);
25859 Py_DECREF(x);
25860 return ival;
25861 }
25862 }
25863 static CYTHON_INLINE PyObject * __Pyx_PyBool_FromLong(long b) {
25864 return b ? __Pyx_NewRef(Py_True) : __Pyx_NewRef(Py_False);
25865 }
25866 static CYTHON_INLINE PyObject * __Pyx_PyInt_FromSize_t(size_t ival) {
25867 return PyInt_FromSize_t(ival);
25868 }
25869
25870
25871 #endif /* Py_PYTHON_H */

```

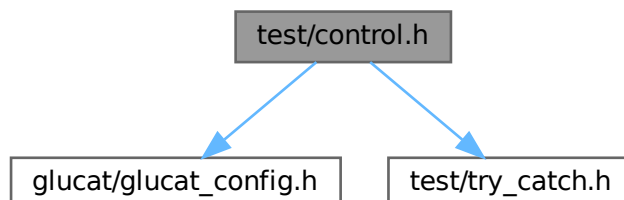


## 7.65 test/control.h File Reference

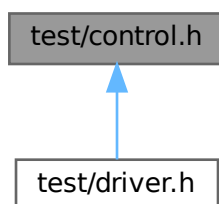
```
#include "glucat/glucat_config.h"
```

```
#include "test/try_catch.h"
```

Include dependency graph for control.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class `glucat::control_t`  
*Parameters to control tests.*

### Namespaces

- namespace `glucat`

## 7.66 control.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_CONTROL_H
00002 #define _GLUCAT_CONTROL_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 control.h : Define and set parameters to control tests
00006
00007 begin : 2010-04-21
00008 copyright : (C) 2010-2016 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033 #include "glucat/glucat_config.h"
00034 #include "test/try_catch.h"
00035
00036 namespace glucat
00037 {
00038 class control_t
00039 {
00040 private:
00041 bool m_valid;
00042 bool valid() const
00043 { return m_valid; }
00044
00045 bool m_catch_exceptions;
00046 bool catch_exceptions() const
00047 { return m_catch_exceptions; }
00048
00049 static bool m_verbose_output;
00050
00051 control_t(int argc, char ** argv);
00052 // Enforce singleton
00053 // Reference: A. Alexandrescu, "Modern C++ Design", Chapter 6
00054 control_t() = default;
00055 ~control_t() = default;
00056 control_t(const control_t&) = delete;
00057 control_t& operator= (const control_t&) = delete;
00058
00059 friend class friend_for_private_destructor;
00060 public:
00061 static const control_t& control(int argc, char ** argv)
00062 { static const control_t c(argc, argv); return c; }
00063
00064 int call(intfn f) const;
00065 int call(intintfn f, int arg) const;
00066
00067 static bool verbose()
00068 { return m_verbose_output; }
00069 };
00070
00071 bool control_t::m_verbose_output = false;
00072
00073 control_t::
00074 control_t(int argc, char ** argv)
00075 : m_valid(true), m_catch_exceptions(true)
00076 {
00077 bool print_help = false;
00078 const std::string& arg_0_str = argv[0];
00079 const std::string program_name = arg_0_str.substr(arg_0_str.find_last_of('/')+1);
00080 for (int arg_ndx = 1; arg_ndx < argc; ++arg_ndx)
00081 {
00082 const std::string& arg_str = argv[arg_ndx];

```

```

00098 bool valid = false;
00099 if (arg_str.substr(0,2) == "--")
00100 {
00101 valid = true;
00102 const std::string& arg_name = arg_str.substr(2);
00103 if (arg_name == "help")
00104 {
00105 this->m_valid = false;
00106 print_help = true;
00107 }
00108 else if (arg_name == "verbose")
00109 this->m_verbose_output = true;
00110 else if (arg_name == "no-catch")
00111 this->m_catch_exceptions = false;
00112 else
00113 valid = false;
00114 }
00115 if (!valid)
00116 {
00117 std::cout << "Invalid argument: " << arg_str << std::endl;
00118 this->m_valid = false;
00119 print_help = true;
00120 }
00121 }
00122 if (print_help)
00123 {
00124 std::cout << program_name << " for " << GLUCAT_PACKAGE_NAME << " version " << GLUCAT_VERSION << ":" <<
std::endl;
00125 std::cout << "Usage: " << program_name << " [option ...]" << std::endl;
00126 std::cout << "Options:" << std::endl;
00127 std::cout << " --help : Print this summary." << std::endl;
00128 std::cout << " --no-catch : Do not catch exceptions." << std::endl;
00129 std::cout << " --verbose : Produce more detailed test output." << std::endl;
00130 }
00131 }
00132
00133 inline
00134 int
00135 control_t::
00136 call(intfn f) const
00137 {
00138 if (valid())
00139 return (catch_exceptions())
00140 ? try_catch(f)
00141 : (*f)();
00142 else
00143 return 1;
00144 }
00145
00146 inline
00147 int
00148 control_t::
00149 call(intintfn f, int arg) const
00150 {
00151 if (valid())
00152 return (catch_exceptions())
00153 ? try_catch(f, arg)
00154 : (*f)(arg);
00155 else
00156 return 1;
00157 }
00158 }
00159 }
00160 }
00161 #endif // _GLUCAT_CONTROL_H

```

## 7.67 test/driver.h File Reference

```

#include "glucat/glucat.h"
#include "glucat/glucat_imp.h"
#include "test/tuning.h"
#include "test/try_catch.h"
#include "test/control.h"
#include <stdio>

```

Include dependency graph for driver.h:



## 7.68 driver.h

[Go to the documentation of this file.](#)

```

00001 #ifndef GLUCAT_TEST_DRIVER_H
00002 #define GLUCAT_TEST_DRIVER_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 driver.h : Header for example and timing test driver
00006 -----
00007 begin : Sun 2001-12-09
00008 copyright : (C) 2001-2021 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 #include "glucat/glucat.h"
00035 #include "glucat/glucat_imp.h"
00036 #include "test/tuning.h"
00037 #include "test/try_catch.h"
00038 #include "test/control.h"
00039 #include <cstdio>
00040
00041 #endif // GLUCAT_TEST_DRIVER_H

```

## 7.69 test/timing.h File Reference

### Namespaces

- namespace [glucat](#)
- namespace [glucat::timing](#)

### Functions

- static double [glucat::timing::elapsed](#) (clock\_t cpu\_time)  
*Elapsed time in milliseconds.*

## Variables

- const double `glucat::timing::MS_PER_SEC` = 1000.0  
*Timing constant: milliseconds per second.*
- const double `glucat::timing::MS_PER_CLOCK` = `MS_PER_SEC` / `double(CLOCKS_PER_SEC)`  
*Timing constant: milliseconds per clock.*
- const int `glucat::timing::EXTRA_TRIALS` = 2  
*Timing constant: trial expansion factor.*

## 7.70 timing.h

Go to the documentation of this file.

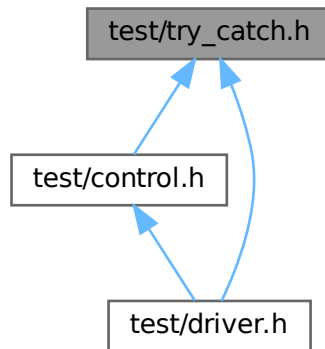
```

00001 #ifndef GLUCAT_TEST_TIMING_H
00002 #define GLUCAT_TEST_TIMING_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 timing.h : Common definitions for timing tests
00006 -----
00007 begin : Tue 2012-03-27
00008 copyright : (C) 2012 by Paul C. Leopardi
00009 *****/
00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****/
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****/
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 namespace glucat
00035 {
00036 namespace timing
00037 {
00038 {
00039 const double MS_PER_SEC = 1000.0;
00040
00042 const double MS_PER_CLOCK = MS_PER_SEC / double(CLOCKS_PER_SEC);
00043
00045 const int EXTRA_TRIALS = 2;
00046
00048 inline
00049 static
00050 double
00051 elapsed(clock_t cpu_time)
00052 { return double(clock() - cpu_time) * MS_PER_CLOCK; }
00053
00054 }
00055 }
00056 #endif // GLUCAT_TEST_TIMING_H

```

## 7.71 test/try\_catch.h File Reference

This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [glucat](#)

### Typedefs

- typedef int(\* [glucat::intfn](#)) ()  
*For exception catching: pointer to function returning int.*
- typedef int(\* [glucat::intintfn](#)) (int)  
*For exception catching: pointer to function of int returning int.*

### Functions

- int [glucat::try\\_catch](#) ([intfn](#) f)  
*Exception catching for functions returning int.*
- int [glucat::try\\_catch](#) ([intintfn](#) f, int arg)  
*Exception catching for functions of int returning int.*

## 7.72 try\_catch.h

[Go to the documentation of this file.](#)

```

00001 #ifndef _GLUCAT_TRY_CATCH_H
00002 #define _GLUCAT_TRY_CATCH_H
00003 /*****
00004 GluCat : Generic library of universal Clifford algebra templates
00005 try_catch.h : Catch exceptions
00006 -----
00007 begin : Sun 2001-12-20
00008 copyright : (C) 2001-2010 by Paul C. Leopardi
00009 *****/

```

```

00010
00011 This library is free software: you can redistribute it and/or modify
00012 it under the terms of the GNU Lesser General Public License as published
00013 by the Free Software Foundation, either version 3 of the License, or
00014 (at your option) any later version.
00015
00016 This library is distributed in the hope that it will be useful,
00017 but WITHOUT ANY WARRANTY; without even the implied warranty of
00018 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00019 GNU Lesser General Public License for more details.
00020
00021 You should have received a copy of the GNU Lesser General Public License
00022 along with this library. If not, see <http://www.gnu.org/licenses/>.
00023
00024 *****
00025 This library is based on a prototype written by Arvind Raja and was
00026 licensed under the LGPL with permission of the author. See Arvind Raja,
00027 "Object-oriented implementations of Clifford algebras in C++: a prototype",
00028 in Ablamowicz, Lounesto and Parra (eds.)
00029 "Clifford algebras with numeric and symbolic computations", Birkhauser, 1996.
00030 *****
00031 See also Arvind Raja's original header comments in glucat.h
00032 *****/
00033
00034 namespace glucat
00035 {
00037 typedef int (*intfn)();
00038
00040 typedef int (*intintfn)(int);
00041
00043 int try_catch(intfn f);
00044
00046 int try_catch(intintfn f, int arg);
00047
00049 int try_catch(intfn f)
00050 {
00051 int result = 0;
00052 try
00053 { result = (*f)(); }
00054 catch (const glucat_error& e)
00055 { e.print_error_msg(); }
00056 catch (const std::bad_alloc& e)
00057 { std::cerr << "bad_alloc" << std::endl; }
00058 catch (...)
00059 { std::cerr << "unexpected exception" << std::endl; }
00060 return result;
00061 }
00062
00064 int try_catch(intintfn f, int arg)
00065 {
00066 int result = 0;
00067 try
00068 { result = (*f)(arg); }
00069 catch (const glucat_error& e)
00070 { e.print_error_msg(); }
00071 catch (const std::bad_alloc& e)
00072 { std::cerr << "bad_alloc" << std::endl; }
00073 catch (...)
00074 { std::cerr << "unexpected exception" << std::endl; }
00075 return result;
00076 }
00077 }
00078 #endif // _GLUCAT_TRY_CATCH_H

```





# Index

`_GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS`  
    `clifford_algebra.h`, [245](#)

`_GLUCAT_CTAssert`  
    `global.h`, [314](#)  
    `glucat`, [24](#)  
    `tuning.h`, [411](#)

`_GLUCAT_HASH_N`  
    `framed_multi_imp.h`, [286](#)

`_GLUCAT_HASH_SIZE_T`  
    `framed_multi_imp.h`, [286](#)

`_GLUCAT_ISINF`  
    `portability.h`, [391](#)

`_GLUCAT_ISNAN`  
    `portability.h`, [391](#)

`__add__`  
    `PyClical.clifford`, [82](#)

`__and__`  
    `PyClical.clifford`, [82](#)  
    `PyClical.index_set`, [161](#)

`__call__`  
    `PyClical.clifford`, [82](#)

`__cinit__`  
    `PyClical.clifford`, [83](#)  
    `PyClical.index_set`, [161](#)

`__contains__`  
    `PyClical.clifford`, [83](#)  
    `PyClical.index_set`, [161](#)

`__dealloc__`  
    `PyClical.clifford`, [84](#)  
    `PyClical.index_set`, [162](#)

`__getitem__`  
    `PyClical.clifford`, [84](#)  
    `PyClical.index_set`, [162](#)

`__iadd__`  
    `PyClical.clifford`, [84](#)

`__iand__`  
    `PyClical.clifford`, [85](#)  
    `PyClical.index_set`, [162](#)

`__idiv__`  
    `PyClical.clifford`, [85](#)

`__imod__`  
    `PyClical.clifford`, [85](#)

`__imul__`  
    `PyClical.clifford`, [86](#)

`__invert__`  
    `PyClical.index_set`, [163](#)

`__ior__`  
    `PyClical.clifford`, [86](#)  
    `PyClical.index_set`, [163](#)

`__isub__`  
    `PyClical.clifford`, [86](#)

`__iter__`  
    `PyClical.clifford`, [87](#)  
    `PyClical.index_set`, [163](#)

`__ixor__`  
    `PyClical.clifford`, [87](#)  
    `PyClical.index_set`, [163](#)

`__mod__`  
    `PyClical.clifford`, [87](#)

`__mul__`  
    `PyClical.clifford`, [88](#)

`__neg__`  
    `PyClical.clifford`, [88](#)

`__or__`  
    `PyClical.clifford`, [88](#)  
    `PyClical.index_set`, [164](#)

`__pos__`  
    `PyClical.clifford`, [89](#)

`__pow__`  
    `PyClical.clifford`, [89](#)

`__repr__`  
    `PyClical.clifford`, [89](#)  
    `PyClical.index_set`, [164](#)

`__richcmp__`  
    `PyClical.clifford`, [90](#)  
    `PyClical.index_set`, [164](#)

`__setitem__`  
    `PyClical.index_set`, [165](#)

`__str__`  
    `PyClical.clifford`, [90](#)  
    `PyClical.index_set`, [165](#)

`__sub__`  
    `PyClical.clifford`, [90](#)

`__truediv__`  
    `PyClical.clifford`, [91](#)

`__version__`  
    `PyClical`, [74](#)

`__xor__`  
    `PyClical.clifford`, [91](#)  
    `PyClical.index_set`, [165](#)

`_test`  
    `PyClical`, [70](#)

`~basis_table`  
    `glucat::basis_table< Scalar_T, LO, HI, Matrix_T >`,  
        [78](#)

`~clifford_algebra`  
    `glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >`, [102](#)

- ~control\_t
  - glucat::control\_t, [112](#)
- ~framed\_multi
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [129](#)
- ~generator\_table
  - glucat::gen::generator\_table< Matrix\_T >, [140](#)
- ~glucat\_error
  - glucat::glucat\_error, [144](#)
- ~matrix\_multi
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [178](#)
- ~random\_generator
  - glucat::random\_generator< Scalar\_T >, [224](#)
- ~reference
  - glucat::index\_set< LO, HI >::reference, [227](#)
- ~var\_term
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::var\_term, [234](#)
- abs
  - glucat, [24](#)
  - glucat::numeric\_traits< Scalar\_T >, [192](#)
  - PyClical.clifford, [91](#)
- acos
  - glucat, [25](#)
  - glucat::numeric\_traits< Scalar\_T >, [192](#)
- acosh
  - glucat, [25](#)
- agc3
  - cga3, [9](#)
- approx\_equal
  - glucat, [26](#)
- are\_same
  - glucat::compare\_types< LHS\_T, RHS\_T >, [110](#)
  - glucat::compare\_types< T, T >, [111](#)
- array
  - pade::pade\_log\_denom< dd\_real >, [203](#)
  - pade::pade\_log\_denom< float >, [204](#)
  - pade::pade\_log\_denom< long double >, [205](#)
  - pade::pade\_log\_denom< qd\_real >, [206](#)
  - pade::pade\_log\_denom< Scalar\_T >, [202](#)
  - pade::pade\_log\_numer< dd\_real >, [208](#)
  - pade::pade\_log\_numer< float >, [209](#)
  - pade::pade\_log\_numer< long double >, [210](#)
  - pade::pade\_log\_numer< qd\_real >, [211](#)
  - pade::pade\_log\_numer< Scalar\_T >, [207](#)
  - pade::pade\_sqrt\_denom< dd\_real >, [213](#)
  - pade::pade\_sqrt\_denom< float >, [214](#)
  - pade::pade\_sqrt\_denom< long double >, [215](#)
  - pade::pade\_sqrt\_denom< qd\_real >, [216](#)
  - pade::pade\_sqrt\_denom< Scalar\_T >, [212](#)
  - pade::pade\_sqrt\_numer< dd\_real >, [218](#)
  - pade::pade\_sqrt\_numer< float >, [219](#)
  - pade::pade\_sqrt\_numer< long double >, [220](#)
  - pade::pade\_sqrt\_numer< qd\_real >, [221](#)
  - pade::pade\_sqrt\_numer< Scalar\_T >, [217](#)
- asin
  - glucat, [26](#), [27](#)
  - glucat::numeric\_traits< Scalar\_T >, [192](#)
- asinh
  - glucat, [27](#)
- atan
  - glucat, [27](#), [28](#)
  - glucat::numeric\_traits< Scalar\_T >, [192](#)
- atanh
  - glucat, [28](#)
- basis
  - glucat::basis\_table< Scalar\_T, LO, HI, Matrix\_T >, [79](#)
- basis\_element
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [182](#)
- basis\_matrix\_t
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [176](#)
- basis\_table
  - glucat::basis\_table< Scalar\_T, LO, HI, Matrix\_T >, [78](#)
- BITS\_PER\_SET\_VALUE
  - glucat, [57](#)
- bitset\_t
  - glucat::index\_set< LO, HI >, [149](#)
- BOOST\_STATIC\_ASSERT
  - glucat::index\_set< LO, HI >, [151](#)
- call
  - glucat::control\_t, [113](#)
- cascade\_log
  - glucat, [28](#)
- catch\_exceptions
  - glucat::control\_t, [113](#)
- centre\_pm4\_qp4
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [133](#)
- centre\_pp4\_qm4
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [133](#)
- centre\_qp1\_pm1
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [133](#)
- cga3, [9](#)
  - agc3, [9](#)
  - cga3, [9](#)
  - cga3std, [10](#)
- cga3std
  - cga3, [10](#)
- check\_complex
  - glucat, [29](#)
- cl
  - PyClical, [74](#)
- classify\_eigenvalues
  - glucat::matrix, [63](#)
- classname
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [102](#)
  - glucat::error< Class\_T >, [120](#)

- glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 133
- glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::var\_term, 235
- glucat::glucat\_error, 144
- glucat::index\_set< LO, HI >, 151
- glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 182
- Clifford
  - PyClical.h, 419
- clifford\_algebra.h
  - \_GLUCAT\_CLIFFORD\_ALGEBRA\_OPERATIONS, 245
- clifford\_exp
  - glucat, 29
- clifford\_hidden\_doctests
  - PyClical, 70
- clifford\_to\_repr
  - PyClical.h, 420
- clifford\_to\_str
  - PyClical.h, 420
- compare
  - glucat, 29
  - glucat::index\_set< LO, HI >, 158
- complexifier
  - glucat, 30
- conj
  - glucat, 30
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 102
  - glucat::numeric\_traits< Scalar\_T >, 193
  - PyClical.clifford, 92
- const\_iterator
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 126
- control
  - glucat::control\_t, 113
- control\_t
  - glucat::control\_t, 112
- cos
  - glucat, 30
  - glucat::numeric\_traits< Scalar\_T >, 193
- cosh
  - glucat, 31
  - glucat::numeric\_traits< Scalar\_T >, 193
- count
  - glucat::index\_set< LO, HI >, 151
  - PyClical.index\_set, 166
- count\_neg
  - glucat::index\_set< LO, HI >, 152
  - PyClical.index\_set, 166
- count\_pos
  - glucat::index\_set< LO, HI >, 152
  - PyClical.index\_set, 166
- cr\_sqrt
  - glucat, 31
- crd\_of\_mult
  - glucat, 31, 32
- db\_sqrt
  - glucat, 32
- db\_step
  - glucat, 32
- DEFAULT\_HI
  - glucat, 57
- default\_truncation
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 109
- denom
  - pade::pade\_log\_denom< dd\_real >, 203
  - pade::pade\_log\_denom< float >, 204
  - pade::pade\_log\_denom< long double >, 205
  - pade::pade\_log\_denom< qd\_real >, 206
  - pade::pade\_log\_denom< Scalar\_T >, 202
  - pade::pade\_sqrt\_denom< dd\_real >, 213
  - pade::pade\_sqrt\_denom< float >, 214
  - pade::pade\_sqrt\_denom< long double >, 215
  - pade::pade\_sqrt\_denom< qd\_real >, 216
  - pade::pade\_sqrt\_denom< Scalar\_T >, 212
- divide
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 133
- e
  - PyClical, 72
- eig\_case\_t
  - glucat::matrix, 63
- eigenvalues
  - glucat::matrix, 63
- elapsed
  - glucat::timing, 68
- elliptic
  - glucat, 32
- epsilon
  - PyClical.h, 421
- error
  - glucat::error< Class\_T >, 120
- error\_squared
  - glucat, 33
- error\_squared\_tol
  - glucat, 33
- error\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 126
  - glucat::index\_set< LO, HI >, 149
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 176
- even
  - glucat, 33
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 102
  - PyClical.clifford, 92
- exp
  - glucat, 34
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 136
  - glucat::numeric\_traits< Scalar\_T >, 193
- EXTRA\_TRIALS

- glucat::timing, 68
- fast
  - glucat, 34
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 134
- fast\_framed\_multi
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 134
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 182
- fast\_matrix\_multi
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 134
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 183
- fill
  - PyClical, 74
- flip
  - glucat::index\_set< LO, HI >, 152
  - glucat::index\_set< LO, HI >::reference, 227
- fmod
  - glucat::numeric\_traits< Scalar\_T >, 193
- fold
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 134
  - glucat::index\_set< LO, HI >, 152, 153
- folded\_dim
  - glucat, 34
- frame
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 102
  - PyClical.clifford, 92
- framed\_multi
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 129–132, 136
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 184
- framed\_multi\_imp.h
  - \_GLUCAT\_HASH\_N, 286
  - \_GLUCAT\_HASH\_SIZE\_T, 286
- framed\_multi\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 126
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 176
- framed\_pair\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 126
- friend\_for\_private\_destructor
  - glucat::basis\_table< Scalar\_T, LO, HI, Matrix\_T >, 79
  - glucat::control\_t, 114
  - glucat::gen::generator\_table< Matrix\_T >, 142
  - glucat::random\_generator< Scalar\_T >, 225
- gen\_from\_pm1\_qm1
  - glucat::gen::generator\_table< Matrix\_T >, 140
- gen\_from\_pm4\_qp4
  - glucat::gen::generator\_table< Matrix\_T >, 140
- gen\_from\_pp4\_qm4
  - glucat::gen::generator\_table< Matrix\_T >, 141
- gen\_from\_qp1\_pm1
  - glucat::gen::generator\_table< Matrix\_T >, 141
- gen\_vector
  - glucat::gen::generator\_table< Matrix\_T >, 141
- generator
  - glucat::gen::generator\_table< Matrix\_T >, 141
  - glucat::random\_generator< Scalar\_T >, 224
- generator\_table
  - glucat::gen::generator\_table< Matrix\_T >, 140
- global.h
  - \_GLUCAT\_CTAssert, 314
- glucat, 10
  - \_GLUCAT\_CTAssert, 24
  - abs, 24
  - acos, 25
  - acosh, 25
  - approx\_equal, 26
  - asin, 26, 27
  - asinh, 27
  - atan, 27, 28
  - atanh, 28
  - BITS\_PER\_SET\_VALUE, 57
  - cascade\_log, 28
  - check\_complex, 29
  - clifford\_exp, 29
  - compare, 29
  - complexifier, 30
  - conj, 30
  - cos, 30
  - cosh, 31
  - cr\_sqrt, 31
  - crd\_of\_mult, 31, 32
  - db\_sqrt, 32
  - db\_step, 32
  - DEFAULT\_HI, 57
  - elliptic, 32
  - error\_squared, 33
  - error\_squared\_tol, 33
  - even, 33
  - exp, 34
  - fast, 34
  - folded\_dim, 34
  - imag, 35
  - index\_t, 22
  - intfn, 22
  - intintfn, 22
  - inv, 35
  - inverse\_gray, 35
  - inverse\_reversed\_gray, 35
  - involute, 36
  - I\_ln2, 57
  - I\_pi, 58
  - log, 36, 37
  - log2, 37
  - matrix\_log, 37

- matrix\_sqrt, 37
- max\_abs, 38
- max\_pos, 38
- min\_neg, 38
- MS\_PER\_S, 58
- norm, 38
- odd, 39
- offset\_level, 39
- operator!=, 39, 40
- operator<<, 46, 47
- operator>>, 47
- operator+, 43, 44
- operator-, 44
- operator/, 45, 46
- operator%, 40
- operator&, 41
- operator\*, 42, 43
- operator^, 47, 48
- operator|, 48, 49
- outer\_pow, 49
- pade\_approx, 50
- pade\_log, 50
- pos\_mod, 50
- pow, 50, 51
- pure, 51
- quad, 51
- real, 51
- reframe, 52
- reverse, 52
- scalar, 52
- set\_value\_t, 22
- sign\_of\_square, 52
- sin, 53
- sinh, 53
- sqrt, 53, 54
- star, 54, 55
- tan, 55
- tanh, 56
- to\_demote, 56
- to\_promote, 56
- try\_catch, 56, 57
- tuning\_fast, 23
- Tuning\_Fast\_Basis\_Max\_Count, 58
- Tuning\_Fast\_CR\_Sqrt\_Max\_Steps, 58
- Tuning\_Fast\_DB\_Sqrt\_Max\_Steps, 58
- Tuning\_Fast\_Div\_Max\_Steps, 58
- Tuning\_Fast\_Fast\_Size\_Threshold, 58
- Tuning\_Fast\_Inv\_Fast\_Dim\_Threshold, 59
- Tuning\_Fast\_Log\_Max\_Inner\_Steps, 59
- Tuning\_Fast\_Log\_Max\_Outer\_Steps, 59
- Tuning\_Fast\_Mult\_Matrix\_Threshold, 59
- Tuning\_Fast\_Products\_Size\_Threshold, 59
- Tuning\_Int\_Digits, 59
- Tuning\_Max\_Threshold, 59
- tuning\_naive, 23
- Tuning\_Naive\_Basis\_Max\_Count, 59
- Tuning\_Naive\_Fast\_Size\_Threshold, 60
- Tuning\_Naive\_Inv\_Fast\_Dim\_Threshold, 60
- Tuning\_Naive\_Mult\_Matrix\_Threshold, 60
- tuning\_slow, 23
- Tuning\_Slow\_Basis\_Max\_Count, 60
- Tuning\_Slow\_Fast\_Size\_Threshold, 60
- Tuning\_Slow\_Inv\_Fast\_Dim\_Threshold, 60
- Tuning\_Slow\_Mult\_Matrix\_Threshold, 60
- Tuning\_Slow\_Products\_Size\_Threshold, 60
- vector\_part, 57
- glucat/clifford\_algebra.h, 237, 245
- glucat/clifford\_algebra\_imp.h, 254, 261
- glucat/errors.h, 273, 275
- glucat/errors\_imp.h, 275, 276
- glucat/framed\_multi.h, 277, 280
- glucat/framed\_multi\_imp.h, 284, 286
- glucat/generation.h, 307, 308
- glucat/generation\_imp.h, 309, 310
- glucat/global.h, 313, 315
- glucat/glucat.h, 316, 317
- glucat/glucat\_config.h, 318, 322
- glucat/glucat\_imp.h, 323, 324
- glucat/index\_set.h, 325, 326
- glucat/index\_set\_imp.h, 329, 330
- glucat/long\_double.h, 342, 343
- glucat/matrix.h, 344, 346
- glucat/matrix\_imp.h, 347, 349
- glucat/matrix\_multi.h, 357, 359
- glucat/matrix\_multi\_imp.h, 362, 366
- glucat/portability.h, 390, 392
- glucat/promotion.h, 392, 394
- glucat/qd.h, 396, 397
- glucat/random.h, 401, 402
- glucat/scalar.h, 403, 404
- glucat/scalar\_imp.h, 407, 408
- glucat/tuning.h, 410, 412
- glucat::basis\_table< Scalar\_T, LO, HI, Matrix\_T >, 77
  - ~basis\_table, 78
  - basis, 79
  - basis\_table, 78
  - friend\_for\_private\_destructor, 79
  - operator=, 79
- glucat::bool\_to\_type< truth\_value >, 79
  - value, 80
- glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multi-
  - vector\_T >, 99
  - ~clifford\_algebra, 102
  - classname, 102
  - conj, 102
  - default\_truncation, 109
  - even, 102
  - frame, 102
  - grade, 102
  - index\_set\_t, 101
  - inv, 103
  - involute, 103
  - isinf, 103
  - isnan, 103
  - max\_abs, 103
  - multivector\_t, 101

- norm, 103
- odd, 104
- operator(), 104
- operator+=, 105
- operator-, 105
- operator-=, 105
- operator/=: 105, 106
- operator==, 106
- operator%=: 104
- operator&=: 104
- operator[], 106
- operator\*=, 104
- operator^=: 106
- operator|=: 106
- outer\_pow, 107
- pair\_t, 101
- pow, 107
- pure, 107
- quad, 107
- reverse, 107
- scalar, 107
- scalar\_t, 101
- truncated, 108
- v\_hi, 109
- v\_lo, 109
- vector\_part, 108
- vector\_t, 101
- write, 108
- glucat::compare\_types< LHS\_T, RHS\_T >, 109
  - are\_same, 110
- glucat::compare\_types< T, T >, 110
  - are\_same, 111
- glucat::control\_t, 111
  - ~control\_t, 112
  - call, 113
  - catch\_exceptions, 113
  - control, 113
  - control\_t, 112
  - friend\_for\_private\_destructor, 114
  - m\_catch\_exceptions, 114
  - m\_valid, 114
  - m\_verbose\_output, 114
  - operator=, 113
  - valid, 113
  - verbose, 114
- glucat::CTAssertion< bool >, 115
- glucat::CTAssertion< true >, 115
- glucat::error< Class\_T >, 118
  - classname, 120
  - error, 120
  - heading, 120
  - print\_error\_msg, 120
- glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 121
  - ~framed\_multi, 129
  - centre\_pm4\_qp4, 133
  - centre\_pp4\_qm4, 133
  - centre\_qp1\_pm1, 133
  - classname, 133
  - const\_iterator, 126
  - divide, 133
  - error\_t, 126
  - exp, 136
  - fast, 134
  - fast\_framed\_multi, 134
  - fast\_matrix\_multi, 134
  - fold, 134
  - framed\_multi, 129–132, 136
  - framed\_multi\_t, 126
  - framed\_pair\_t, 126
  - index\_set\_t, 126
  - iterator, 127
  - map\_t, 127
  - matrix\_multi, 136
  - matrix\_multi\_t, 127
  - matrix\_t, 127
  - multivector\_t, 127
  - nbr\_terms, 135
  - operator<<, 137
  - operator>>, 137
  - operator+=, 135
  - operator/, 137
  - operator%, 136
  - operator&, 136
  - operator\*, 137
  - operator^, 137
  - operator|, 137
  - random, 135
  - scalar\_t, 127
  - size\_type, 128
  - sorted\_map\_t, 128
  - star, 138
  - term\_t, 128
  - tune\_p, 128
  - unfold, 135
  - var\_term\_t, 128
  - vector\_t, 128
- glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::hash\_size\_t, 145
  - hash\_size\_t, 145
  - n, 146
  - operator(), 146
- glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::var\_term, 233
  - ~var\_term, 234
  - classname, 235
  - operator\*=, 235
  - var\_pair\_t, 234
  - var\_term, 234, 235
- glucat::gen, 61
  - offset\_to\_super, 61
  - signature\_t, 61
- glucat::gen::generator\_table< Matrix\_T >, 138
  - ~generator\_table, 140
  - friend\_for\_private\_destructor, 142
  - gen\_from\_pm1\_qm1, 140
  - gen\_from\_pm4\_qp4, 140

- gen\_from\_pp4\_qm4, 141
- gen\_from\_qp1\_pm1, 141
- gen\_vector, 141
- generator, 141
- generator\_table, 140
- operator(), 142
- operator=, 142
- glucat::glucat\_error, 143
  - ~glucat\_error, 144
  - classname, 144
  - glucat\_error, 144
  - heading, 144
  - name, 145
  - print\_error\_msg, 144
- glucat::index\_set< LO, HI >, 146
  - bitset\_t, 149
  - BOOST\_STATIC\_ASSERT, 151
  - classname, 151
  - compare, 158
  - count, 151
  - count\_neg, 152
  - count\_pos, 152
  - error\_t, 149
  - flip, 152
  - fold, 152, 153
  - hash\_fn, 153
  - index\_pair\_t, 150
  - index\_set, 150, 151
  - index\_set\_t, 150
  - is\_contiguous, 153
  - lex\_less\_than, 153
  - max, 153
  - min, 154
  - operator!=, 154
  - operator<, 154
  - operator==, 155
  - operator&, 158
  - operator&=, 154
  - operator[], 155
  - operator~, 156
  - operator^, 158
  - operator^=, 155
  - operator|, 158
  - operator|=, 155
  - reference, 159
  - reset, 156
  - set, 156, 157
  - sign\_of\_mult, 157
  - sign\_of\_square, 157
  - test, 157
  - unfold, 157
  - v\_hi, 159
  - v\_lo, 159
  - value\_of\_fold, 158
- glucat::index\_set< LO, HI >::reference, 226
  - ~reference, 227
  - flip, 227
  - index\_set, 229
  - m\_idx, 229
  - m\_pst, 229
  - operator bool, 227
  - operator=, 228
  - operator==, 228
  - operator~, 228
  - reference, 227
- glucat::index\_set\_hash< LO, HI >, 169
  - index\_set\_t, 170
  - operator(), 170
- glucat::matrix, 62
  - classify\_eigenvalues, 63
  - eig\_case\_t, 63
  - eigenvalues, 63
  - inner, 64
  - isinf, 64
  - isnan, 64
  - kron, 64
  - mono\_kron, 65
  - mono\_prod, 65
  - nnz, 65
  - nork, 65
  - nork\_range, 66
  - norm\_frob2, 66
  - prod, 66
  - signed\_perm\_nork, 66
  - sparse\_prod, 67
  - to\_lapack, 67
  - trace, 67
  - unit, 67
- glucat::matrix::eig\_genus< Matrix\_T >, 116
  - m\_eig\_case, 117
  - m\_is\_singular, 117
  - m\_safe\_arg, 117
  - Scalar\_T, 117
- glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 170
  - ~matrix\_multi, 178
  - basis\_element, 182
  - basis\_matrix\_t, 176
  - classname, 182
  - error\_t, 176
  - fast\_framed\_multi, 182
  - fast\_matrix\_multi, 183
  - framed\_multi, 184
  - framed\_multi\_t, 176
  - index\_set\_t, 176
  - m\_frame, 187
  - m\_matrix, 187
  - matrix\_index\_t, 176
  - matrix\_log, 184
  - matrix\_multi, 178–182, 184
  - matrix\_multi\_t, 176
  - matrix\_sqrt, 184
  - matrix\_t, 176
  - multivector\_t, 177
  - operator<<, 185
  - operator>>, 185
  - operator+=, 183

- operator/, 185
- operator=, 183
- operator%, 185
- operator&, 185
- operator\*, 185
- operator^, 186
- operator|, 186
- orientation\_t, 177
- random, 183
- reframe, 186
- scalar\_t, 177
- star, 186
- term\_t, 177
- tune\_p, 177
- vector\_t, 177
- glucat::numeric\_traits< Scalar\_T >, 190
  - abs, 192
  - acos, 192
  - asin, 192
  - atan, 192
  - conj, 193
  - cos, 193
  - cosh, 193
  - exp, 193
  - fmod, 193
  - imag, 194
  - isInf, 194
  - isNaN, 195
  - isNaN\_or\_isInf, 195
  - ln\_2, 196
  - log, 196
  - log2, 196
  - NaN, 197
  - pi, 197
  - pow, 197
  - real, 198
  - sin, 198
  - sinh, 198
  - sqrt, 198
  - tan, 198
  - tanh, 199
  - to\_double, 199
  - to\_int, 199
  - to\_scalar\_t, 199–201
- glucat::numeric\_traits< Scalar\_T >::demoted, 116
  - type, 116
- glucat::numeric\_traits< Scalar\_T >::promoted, 221
  - type, 222
- glucat::random\_generator< Scalar\_T >, 222
  - ~random\_generator, 224
  - friend\_for\_private\_destructor, 225
  - generator, 224
  - normal, 224
  - normal\_dist, 225
  - operator=, 224
  - random\_generator, 223
  - seed, 225
  - uint\_gen, 225
  - uniform, 224
  - uniform\_dist, 225
- glucat::sorted\_range< Map\_T, Sorted\_Map\_T >, 229
  - map\_t, 230
  - sorted\_begin, 231
  - sorted\_end, 231
  - sorted\_iterator, 230
  - sorted\_map\_t, 230
  - sorted\_range, 230
- glucat::sorted\_range< Sorted\_Map\_T, Sorted\_Map\_T >, 231
  - map\_t, 232
  - sorted\_begin, 232
  - sorted\_end, 232
  - sorted\_iterator, 232
  - sorted\_map\_t, 232
  - sorted\_range, 232
- glucat::timing, 68
  - elapsed, 68
  - EXTRA\_TRIALS, 68
  - MS\_PER\_CLOCK, 68
  - MS\_PER\_SEC, 69
- glucat\_config.h
  - GLUCAT\_HAVE\_CXX11, 319
  - GLUCAT\_HAVE\_INTTYPES\_H, 319
  - GLUCAT\_HAVE\_STDINT\_H, 319
  - GLUCAT\_HAVE\_STDIO\_H, 319
  - GLUCAT\_HAVE\_STDLIB\_H, 319
  - GLUCAT\_HAVE\_STRING\_H, 319
  - GLUCAT\_HAVE\_STRINGS\_H, 319
  - GLUCAT\_HAVE\_SYS\_STAT\_H, 320
  - GLUCAT\_HAVE\_SYS\_TYPES\_H, 320
  - GLUCAT\_HAVE\_UNISTD\_H, 320
  - GLUCAT\_PACKAGE, 320
  - GLUCAT\_PACKAGE\_BUGREPORT, 320
  - GLUCAT\_PACKAGE\_NAME, 320
  - GLUCAT\_PACKAGE\_STRING, 320
  - GLUCAT\_PACKAGE\_TARNAME, 321
  - GLUCAT\_PACKAGE\_URL, 321
  - GLUCAT\_PACKAGE\_VERSION, 321
  - GLUCAT\_STDC\_HEADERS, 321
  - GLUCAT\_VERSION, 321
- glucat\_error
  - glucat::glucat\_error, 144
- GLUCAT\_HAVE\_CXX11
  - glucat\_config.h, 319
- GLUCAT\_HAVE\_INTTYPES\_H
  - glucat\_config.h, 319
- GLUCAT\_HAVE\_STDINT\_H
  - glucat\_config.h, 319
- GLUCAT\_HAVE\_STDIO\_H
  - glucat\_config.h, 319
- GLUCAT\_HAVE\_STDLIB\_H
  - glucat\_config.h, 319
- GLUCAT\_HAVE\_STRING\_H
  - glucat\_config.h, 319
- GLUCAT\_HAVE\_STRINGS\_H
  - glucat\_config.h, 319



GLUCAT\_HAVE\_SYS\_STAT\_H  
     glucat\_config.h, 320  
 GLUCAT\_HAVE\_SYS\_TYPES\_H  
     glucat\_config.h, 320  
 GLUCAT\_HAVE\_UNISTD\_H  
     glucat\_config.h, 320  
 GLUCAT\_PACKAGE  
     glucat\_config.h, 320  
 GLUCAT\_PACKAGE\_BUGREPORT  
     glucat\_config.h, 320  
 GLUCAT\_PACKAGE\_NAME  
     glucat\_config.h, 320  
 GLUCAT\_PACKAGE\_STRING  
     glucat\_config.h, 320  
 GLUCAT\_PACKAGE\_TARNAME  
     glucat\_config.h, 321  
 GLUCAT\_PACKAGE\_URL  
     glucat\_config.h, 321  
 GLUCAT\_PACKAGE\_VERSION  
     glucat\_config.h, 321  
 glucat\_package\_version  
     PyClical.h, 421  
 GLUCAT\_STDC\_HEADERS  
     glucat\_config.h, 321  
 GLUCAT\_VERSION  
     glucat\_config.h, 321  
 grade  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T,  
         Multivector\_T >, 102  
 hash\_fn  
     glucat::index\_set< LO, HI >, 153  
     PyClical.index\_set, 167  
 hash\_size\_t  
     glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P  
         >::hash\_size\_t, 145  
 heading  
     glucat::error< Class\_T >, 120  
     glucat::glucat\_error, 144  
 hi\_ndx  
     PyClical.h, 421  
 i  
     PyClical, 74  
 imag  
     glucat, 35  
     glucat::numeric\_traits< Scalar\_T >, 194  
 index\_pair\_t  
     glucat::index\_set< LO, HI >, 150  
 index\_set  
     glucat::index\_set< LO, HI >, 150, 151  
     glucat::index\_set< LO, HI >::reference, 229  
 index\_set\_hidden\_doctests  
     PyClical, 72  
 index\_set\_t  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T,  
         Multivector\_T >, 101  
     glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >,  
         126  
     glucat::index\_set< LO, HI >, 150  
     glucat::index\_set\_hash< LO, HI >, 170  
     glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >,  
         176  
 index\_set\_to\_repr  
     PyClical.h, 420  
 index\_set\_to\_str  
     PyClical.h, 420  
 index\_t  
     glucat, 22  
 IndexSet  
     PyClical.h, 419  
 inner  
     glucat::matrix, 64  
 instance  
     PyClical.clifford, 98  
     PyClical.index\_set, 169  
 intfn  
     glucat, 22  
 intintfn  
     glucat, 22  
 inv  
     glucat, 35  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T,  
         Multivector\_T >, 103  
     PyClical.clifford, 93  
 inverse\_gray  
     glucat, 35  
 inverse\_reversed\_gray  
     glucat, 35  
 involute  
     glucat, 36  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T,  
         Multivector\_T >, 103  
     PyClical.clifford, 93  
 is\_contiguous  
     glucat::index\_set< LO, HI >, 153  
 isInf  
     glucat::numeric\_traits< Scalar\_T >, 194  
 isinf  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T,  
         Multivector\_T >, 103  
     glucat::matrix, 64  
     PyClical.clifford, 93  
 isNaN  
     glucat::numeric\_traits< Scalar\_T >, 195  
 isnan  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T,  
         Multivector\_T >, 103  
     glucat::matrix, 64  
     PyClical.clifford, 94  
 isNaN\_or\_isInf  
     glucat::numeric\_traits< Scalar\_T >, 195  
 ist  
     PyClical, 74  
 istpq  
     PyClical, 73  
 iterator

- glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 127
- ixt
  - PyClical, 74
- kron
  - glucat::matrix, 64
- l\_ln2
  - glucat, 57
- l\_pi
  - glucat, 58
- lex\_less\_than
  - glucat::index\_set< LO, HI >, 153
- lhs
  - PyClical, 74
- ln\_2
  - glucat::numeric\_traits< Scalar\_T >, 196
- lo\_ndx
  - PyClical.h, 421
- log
  - glucat, 36, 37
  - glucat::numeric\_traits< Scalar\_T >, 196
- log2
  - glucat, 37
  - glucat::numeric\_traits< Scalar\_T >, 196
- m\_catch\_exceptions
  - glucat::control\_t, 114
- m\_eig\_case
  - glucat::matrix::eig\_genus< Matrix\_T >, 117
- m\_frame
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 187
- m\_idx
  - glucat::index\_set< LO, HI >::reference, 229
- m\_is\_singular
  - glucat::matrix::eig\_genus< Matrix\_T >, 117
- m\_matrix
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 187
- m\_pst
  - glucat::index\_set< LO, HI >::reference, 229
- m\_safe\_arg
  - glucat::matrix::eig\_genus< Matrix\_T >, 117
- m\_valid
  - glucat::control\_t, 114
- m\_verbose\_output
  - glucat::control\_t, 114
- map\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 127
  - glucat::sorted\_range< Map\_T, Sorted\_Map\_T >, 230
  - glucat::sorted\_range< Sorted\_Map\_T, Sorted\_Map\_T >, 232
- matrix\_index\_t
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 176
- matrix\_log
  - glucat, 37
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 184
- matrix\_multi
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 136
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 178–182, 184
- matrix\_multi\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 127
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 176
- matrix\_sqrt
  - glucat, 37
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 184
- matrix\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 127
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 176
- max
  - glucat::index\_set< LO, HI >, 153
  - PyClical.index\_set, 167
- max\_abs
  - glucat, 38
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 103
  - PyClical.clifford, 94
- max\_pos
  - glucat, 38
- min
  - glucat::index\_set< LO, HI >, 154
  - PyClical.index\_set, 167
- min\_neg
  - glucat, 38
- mono\_kron
  - glucat::matrix, 65
- mono\_prod
  - glucat::matrix, 65
- MS\_PER\_CLOCK
  - glucat::timing, 68
- MS\_PER\_S
  - glucat, 58
- MS\_PER\_SEC
  - glucat::timing, 69
- multivector\_t
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 101
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 127
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 177
- n
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::hash\_size\_t, 146

name  
     glucat::glucat\_error, 145  
 NaN  
     glucat::numeric\_traits< Scalar\_T >, 197  
 nbar3  
     PyClical, 74  
 nbr\_terms  
     glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 135  
 ninf3  
     PyClical, 75  
 nnz  
     glucat::matrix, 65  
 None  
     PyClical, 75  
 nork  
     glucat::matrix, 65  
 nork\_range  
     glucat::matrix, 66  
 norm  
     glucat, 38  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 103  
     PyClical.clifford, 94  
 norm\_frob2  
     glucat::matrix, 66  
 normal  
     glucat::random\_generator< Scalar\_T >, 224  
 normal\_dist  
     glucat::random\_generator< Scalar\_T >, 225  
 numer  
     pade::pade\_log\_numer< dd\_real >, 208  
     pade::pade\_log\_numer< float >, 209  
     pade::pade\_log\_numer< long double >, 210  
     pade::pade\_log\_numer< qd\_real >, 211  
     pade::pade\_log\_numer< Scalar\_T >, 207  
     pade::pade\_sqrt\_numer< dd\_real >, 218  
     pade::pade\_sqrt\_numer< float >, 219  
     pade::pade\_sqrt\_numer< long double >, 220  
     pade::pade\_sqrt\_numer< qd\_real >, 221  
     pade::pade\_sqrt\_numer< Scalar\_T >, 217  
 obj  
     PyClical, 75  
 odd  
     glucat, 39  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 104  
     PyClical.clifford, 95  
 offset\_level  
     glucat, 39  
 offset\_to\_super  
     glucat::gen, 61  
 operator bool  
     glucat::index\_set< LO, HI >::reference, 227  
 operator!=  
     glucat, 39, 40  
     glucat::index\_set< LO, HI >, 154  
 operator<  
     glucat::index\_set< LO, HI >, 154  
 operator<<  
     glucat, 46, 47  
     glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 137  
     glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 185  
 operator>>  
     glucat, 47  
     glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 137  
     glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 185  
 operator()  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 104  
     glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::hash\_size\_t, 146  
     glucat::gen::generator\_table< Matrix\_T >, 142  
     glucat::index\_set\_hash< LO, HI >, 170  
 operator+  
     glucat, 43, 44  
 operator+=  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 105  
     glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 135  
     glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 183  
 operator-  
     glucat, 44  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 105  
 operator-=  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 105  
 operator/  
     glucat, 45, 46  
     glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 137  
     glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 185  
 operator/=  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 105, 106  
 operator=  
     glucat::basis\_table< Scalar\_T, LO, HI, Matrix\_T >, 79  
     glucat::control\_t, 113  
     glucat::gen::generator\_table< Matrix\_T >, 142  
     glucat::index\_set< LO, HI >::reference, 228  
     glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 183  
     glucat::random\_generator< Scalar\_T >, 224  
 operator==  
     glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 106  
     glucat::index\_set< LO, HI >, 155

- glucat::index\_set< LO, HI >::reference, [228](#)
- operator%
  - glucat, [40](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [136](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [185](#)
- operator%=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [104](#)
- operator&
  - glucat, [41](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [136](#)
  - glucat::index\_set< LO, HI >, [158](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [185](#)
- operator&=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [104](#)
  - glucat::index\_set< LO, HI >, [154](#)
- operator[]
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [106](#)
  - glucat::index\_set< LO, HI >, [155](#)
- operator\*
  - glucat, [42](#), [43](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [137](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [185](#)
- operator\*=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [104](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::var\_term, [235](#)
- operator~
  - glucat::index\_set< LO, HI >, [156](#)
  - glucat::index\_set< LO, HI >::reference, [228](#)
- operator^
  - glucat, [47](#), [48](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [137](#)
  - glucat::index\_set< LO, HI >, [158](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [186](#)
- operator^=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [106](#)
  - glucat::index\_set< LO, HI >, [155](#)
- operator|
  - glucat, [48](#), [49](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [137](#)
  - glucat::index\_set< LO, HI >, [158](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [186](#)
- operator|=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [106](#)
  - glucat::index\_set< LO, HI >, [155](#)
- orientation\_t
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [177](#)
- outer\_pow
  - glucat, [49](#)
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [107](#)
  - PyClical.clifford, [95](#)
- pade, [69](#)
- pade::pade\_log\_denom< dd\_real >, [202](#)
  - array, [203](#)
  - denom, [203](#)
- pade::pade\_log\_denom< float >, [203](#)
  - array, [204](#)
  - denom, [204](#)
- pade::pade\_log\_denom< long double >, [204](#)
  - array, [205](#)
  - denom, [205](#)
- pade::pade\_log\_denom< qd\_real >, [205](#)
  - array, [206](#)
  - denom, [206](#)
- pade::pade\_log\_denom< Scalar\_T >, [201](#)
  - array, [202](#)
  - denom, [202](#)
- pade::pade\_log\_numer< dd\_real >, [208](#)
  - array, [208](#)
  - numer, [208](#)
- pade::pade\_log\_numer< float >, [209](#)
  - array, [209](#)
  - numer, [209](#)
- pade::pade\_log\_numer< long double >, [209](#)
  - array, [210](#)
  - numer, [210](#)
- pade::pade\_log\_numer< qd\_real >, [210](#)
  - array, [211](#)
  - numer, [211](#)
- pade::pade\_log\_numer< Scalar\_T >, [206](#)
  - array, [207](#)
  - numer, [207](#)
- pade::pade\_sqrt\_denom< dd\_real >, [213](#)
  - array, [213](#)
  - denom, [213](#)
- pade::pade\_sqrt\_denom< float >, [214](#)
  - array, [214](#)
  - denom, [214](#)
- pade::pade\_sqrt\_denom< long double >, [214](#)
  - array, [215](#)
  - denom, [215](#)
- pade::pade\_sqrt\_denom< qd\_real >, [215](#)
  - array, [216](#)
  - denom, [216](#)
- pade::pade\_sqrt\_denom< Scalar\_T >, [212](#)
  - array, [212](#)
  - denom, [212](#)
- pade::pade\_sqrt\_numer< dd\_real >, [218](#)

- array, 218
- numer, 218
- pade::pade\_sqrt\_numer< float >, 219
  - array, 219
  - numer, 219
- pade::pade\_sqrt\_numer< long double >, 219
  - array, 220
  - numer, 220
- pade::pade\_sqrt\_numer< qd\_real >, 220
  - array, 221
  - numer, 221
- pade::pade\_sqrt\_numer< Scalar\_T >, 216
  - array, 217
  - numer, 217
- pade\_approx
  - glucat, 50
- pade\_log
  - glucat, 50
- pair\_t
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 101
- pi
  - glucat::numeric\_traits< Scalar\_T >, 197
  - PyClical, 75
- portability.h
  - \_GLUCAT\_ISINF, 391
  - \_GLUCAT\_ISNAN, 391
  - UBLAS\_ABS, 391
  - UBLAS\_SQRT, 391
- pos\_mod
  - glucat, 50
- pow
  - glucat, 50, 51
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 107
  - glucat::numeric\_traits< Scalar\_T >, 197
  - PyClical.clifford, 95
- print\_error\_msg
  - glucat::error< Class\_T >, 120
  - glucat::glucat\_error, 144
- prod
  - glucat::matrix, 66
- pure
  - glucat, 51
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 107
  - PyClical.clifford, 96
- PY\_SSIZE\_T\_CLEAN
  - PyClical\_nocython.cpp, 448
- PyClical, 70
  - \_\_version\_\_, 74
  - \_test, 70
  - cl, 74
  - clifford\_hidden\_doctests, 70
  - e, 72
  - fill, 74
  - i, 74
  - index\_set\_hidden\_doctests, 72
  - ist, 74
  - istpq, 73
  - ixt, 74
  - lhs, 74
  - nbar3, 74
  - ninf3, 75
  - None, 75
  - obj, 75
  - pi, 75
  - rhs, 75
  - scalar\_epsilon, 75
  - tau, 75
  - threshold, 75
  - tol, 76
- PyClical.clifford, 80
  - \_\_add\_\_, 82
  - \_\_and\_\_, 82
  - \_\_call\_\_, 82
  - \_\_cinit\_\_, 83
  - \_\_contains\_\_, 83
  - \_\_dealloc\_\_, 84
  - \_\_getitem\_\_, 84
  - \_\_iadd\_\_, 84
  - \_\_iand\_\_, 85
  - \_\_idiv\_\_, 85
  - \_\_imod\_\_, 85
  - \_\_imul\_\_, 86
  - \_\_ior\_\_, 86
  - \_\_isub\_\_, 86
  - \_\_iter\_\_, 87
  - \_\_ixor\_\_, 87
  - \_\_mod\_\_, 87
  - \_\_mul\_\_, 88
  - \_\_neg\_\_, 88
  - \_\_or\_\_, 88
  - \_\_pos\_\_, 89
  - \_\_pow\_\_, 89
  - \_\_repr\_\_, 89
  - \_\_richcmp\_\_, 90
  - \_\_str\_\_, 90
  - \_\_sub\_\_, 90
  - \_\_truediv\_\_, 91
  - \_\_xor\_\_, 91
- abs, 91
- conj, 92
- even, 92
- frame, 92
- instance, 98
- inv, 93
- involute, 93
- isinf, 93
- isnan, 94
- max\_abs, 94
- norm, 94
- odd, 95
- outer\_pow, 95
- pow, 95
- pure, 96

- quad, 96
- reframe, 96
- reverse, 97
- scalar, 97
- truncated, 97
- vector\_part, 98
- PyClical.h
  - Clifford, 419
  - clifford\_to\_repr, 420
  - clifford\_to\_str, 420
  - epsilon, 421
  - glucat\_package\_version, 421
  - hi\_ndx, 421
  - index\_set\_to\_repr, 420
  - index\_set\_to\_str, 420
  - IndexSet, 419
  - lo\_ndx, 421
  - PyFloat\_FromDouble, 421
  - scalar\_t, 419
  - String, 420
- PyClical.index\_set, 160
  - \_\_and\_\_, 161
  - \_\_cinit\_\_, 161
  - \_\_contains\_\_, 161
  - \_\_dealloc\_\_, 162
  - \_\_getitem\_\_, 162
  - \_\_iand\_\_, 162
  - \_\_invert\_\_, 163
  - \_\_ior\_\_, 163
  - \_\_iter\_\_, 163
  - \_\_ixor\_\_, 163
  - \_\_or\_\_, 164
  - \_\_repr\_\_, 164
  - \_\_richcmp\_\_, 164
  - \_\_setitem\_\_, 165
  - \_\_str\_\_, 165
  - \_\_xor\_\_, 165
  - count, 166
  - count\_neg, 166
  - count\_pos, 166
  - hash\_fn, 167
  - instance, 169
  - max, 167
  - min, 167
  - sign\_of\_mult, 168
  - sign\_of\_square, 168
- pyclical/glucat.pxd, 416
- pyclical/PyClical.h, 418, 422
- pyclical/PyClical.pxd, 423, 424
- pyclical/PyClical.pyx, 424, 425
- pyclical/PyClical\_nocython.cpp, 448
- PyClical\_nocython.cpp
  - PY\_SSIZE\_T\_CLEAN, 448
- PyFloat\_FromDouble
  - PyClical.h, 421
- quad
  - glucat, 51
- glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 107
- PyClical.clifford, 96
- random
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 135
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 183
- random\_generator
  - glucat::random\_generator< Scalar\_T >, 223
- real
  - glucat, 51
  - glucat::numeric\_traits< Scalar\_T >, 198
- reference
  - glucat::index\_set< LO, HI >, 159
  - glucat::index\_set< LO, HI >::reference, 227
- reframe
  - glucat, 52
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 186
  - PyClical.clifford, 96
- reset
  - glucat::index\_set< LO, HI >, 156
- reverse
  - glucat, 52
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 107
  - PyClical.clifford, 97
- rhs
  - PyClical, 75
- scalar
  - glucat, 52
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 107
  - PyClical.clifford, 97
- scalar\_epsilon
  - PyClical, 75
- Scalar\_T
  - glucat::matrix::eig\_genus< Matrix\_T >, 117
- scalar\_t
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 101
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 127
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 177
  - PyClical.h, 419
- seed
  - glucat::random\_generator< Scalar\_T >, 225
- set
  - glucat::index\_set< LO, HI >, 156, 157
- set\_value\_t
  - glucat, 22
- sign\_of\_mult
  - glucat::index\_set< LO, HI >, 157
  - PyClical.index\_set, 168
- sign\_of\_square

- glucat, [52](#)
- glucat::index\_set< LO, HI >, [157](#)
- PyClical.index\_set, [168](#)
- signature\_t
  - glucat::gen, [61](#)
- signed\_perm\_nork
  - glucat::matrix, [66](#)
- sin
  - glucat, [53](#)
  - glucat::numeric\_traits< Scalar\_T >, [198](#)
- sinh
  - glucat, [53](#)
  - glucat::numeric\_traits< Scalar\_T >, [198](#)
- size\_type
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [128](#)
- sorted\_begin
  - glucat::sorted\_range< Map\_T, Sorted\_Map\_T >, [231](#)
  - glucat::sorted\_range< Sorted\_Map\_T, Sorted\_Map\_T >, [232](#)
- sorted\_end
  - glucat::sorted\_range< Map\_T, Sorted\_Map\_T >, [231](#)
  - glucat::sorted\_range< Sorted\_Map\_T, Sorted\_Map\_T >, [232](#)
- sorted\_iterator
  - glucat::sorted\_range< Map\_T, Sorted\_Map\_T >, [230](#)
  - glucat::sorted\_range< Sorted\_Map\_T, Sorted\_Map\_T >, [232](#)
- sorted\_map\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [128](#)
  - glucat::sorted\_range< Map\_T, Sorted\_Map\_T >, [230](#)
  - glucat::sorted\_range< Sorted\_Map\_T, Sorted\_Map\_T >, [232](#)
- sorted\_range
  - glucat::sorted\_range< Map\_T, Sorted\_Map\_T >, [230](#)
  - glucat::sorted\_range< Sorted\_Map\_T, Sorted\_Map\_T >, [232](#)
- sparse\_prod
  - glucat::matrix, [67](#)
- sqrt
  - glucat, [53](#), [54](#)
  - glucat::numeric\_traits< Scalar\_T >, [198](#)
- star
  - glucat, [54](#), [55](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [138](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [186](#)
- std, [76](#)
- std::numeric\_limits< glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P > >, [187](#)
- std::numeric\_limits< glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P > >, [189](#)
- String
  - PyClical.h, [420](#)
- tan
  - glucat, [55](#)
  - glucat::numeric\_traits< Scalar\_T >, [198](#)
- tanh
  - glucat, [56](#)
  - glucat::numeric\_traits< Scalar\_T >, [199](#)
- tau
  - PyClical, [75](#)
- term\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [128](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [177](#)
- test
  - glucat::index\_set< LO, HI >, [157](#)
  - test/control.h, [781](#), [782](#)
  - test/driver.h, [783](#), [784](#)
  - test/timing.h, [784](#), [785](#)
  - test/try\_catch.h, [786](#)
  - test/tuning.h, [414](#), [415](#)
- threshold
  - PyClical, [75](#)
- to\_demote
  - glucat, [56](#)
- to\_double
  - glucat::numeric\_traits< Scalar\_T >, [199](#)
- to\_int
  - glucat::numeric\_traits< Scalar\_T >, [199](#)
- to\_lapack
  - glucat::matrix, [67](#)
- to\_promote
  - glucat, [56](#)
- to\_scalar\_t
  - glucat::numeric\_traits< Scalar\_T >, [199–201](#)
- tol
  - PyClical, [76](#)
- trace
  - glucat::matrix, [67](#)
- truncated
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [108](#)
  - PyClical.clifford, [97](#)
- try\_catch
  - glucat, [56](#), [57](#)
- tune\_p
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [128](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [177](#)
- tuning.h
  - \_GLUCAT\_CTAssert, [411](#)
- tuning\_fast
  - glucat, [23](#)
- Tuning\_Fast\_Basis\_Max\_Count
  - glucat, [58](#)

- Tuning\_Fast\_CR\_Sqrt\_Max\_Steps
  - glucat, [58](#)
- Tuning\_Fast\_DB\_Sqrt\_Max\_Steps
  - glucat, [58](#)
- Tuning\_Fast\_Div\_Max\_Steps
  - glucat, [58](#)
- Tuning\_Fast\_Fast\_Size\_Threshold
  - glucat, [58](#)
- Tuning\_Fast\_Inv\_Fast\_Dim\_Threshold
  - glucat, [59](#)
- Tuning\_Fast\_Log\_Max\_Inner\_Steps
  - glucat, [59](#)
- Tuning\_Fast\_Log\_Max\_Outer\_Steps
  - glucat, [59](#)
- Tuning\_Fast\_Mult\_Matrix\_Threshold
  - glucat, [59](#)
- Tuning\_Fast\_Products\_Size\_Threshold
  - glucat, [59](#)
- Tuning\_Int\_Digits
  - glucat, [59](#)
- Tuning\_Max\_Threshold
  - glucat, [59](#)
- tuning\_naive
  - glucat, [23](#)
- Tuning\_Naive\_Basis\_Max\_Count
  - glucat, [59](#)
- Tuning\_Naive\_Fast\_Size\_Threshold
  - glucat, [60](#)
- Tuning\_Naive\_Inv\_Fast\_Dim\_Threshold
  - glucat, [60](#)
- Tuning\_Naive\_Mult\_Matrix\_Threshold
  - glucat, [60](#)
- tuning\_slow
  - glucat, [23](#)
- Tuning\_Slow\_Basis\_Max\_Count
  - glucat, [60](#)
- Tuning\_Slow\_Fast\_Size\_Threshold
  - glucat, [60](#)
- Tuning\_Slow\_Inv\_Fast\_Dim\_Threshold
  - glucat, [60](#)
- Tuning\_Slow\_Mult\_Matrix\_Threshold
  - glucat, [60](#)
- Tuning\_Slow\_Products\_Size\_Threshold
  - glucat, [60](#)
- type
  - glucat::numeric\_traits< Scalar\_T >::demoted, [116](#)
  - glucat::numeric\_traits< Scalar\_T >::promoted, [222](#)
- UBLAS\_ABS
  - portability.h, [391](#)
- UBLAS\_SQRT
  - portability.h, [391](#)
- uint\_gen
  - glucat::random\_generator< Scalar\_T >, [225](#)
- unfold
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [135](#)
  - glucat::index\_set< LO, HI >, [157](#)
- uniform
  - glucat::random\_generator< Scalar\_T >, [224](#)
- uniform\_dist
  - glucat::random\_generator< Scalar\_T >, [225](#)
- unit
  - glucat::matrix, [67](#)
- v\_hi
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [109](#)
  - glucat::index\_set< LO, HI >, [159](#)
- v\_lo
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [109](#)
  - glucat::index\_set< LO, HI >, [159](#)
- valid
  - glucat::control\_t, [113](#)
- value
  - glucat::bool\_to\_type< truth\_value >, [80](#)
- value\_of\_fold
  - glucat::index\_set< LO, HI >, [158](#)
- var\_pair\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::var\_term, [234](#)
- var\_term
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::var\_term, [234](#), [235](#)
- var\_term\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [128](#)
- vector\_part
  - glucat, [57](#)
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [108](#)
  - PyClical.clifford, [98](#)
- vector\_t
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [101](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [128](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [177](#)
- verbose
  - glucat::control\_t, [114](#)
- write
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [108](#)