

glucat

0.13.0

Generated by Doxygen 1.14.0



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Namespace Documentation</b>	<b>9</b>
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	23
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&amp;()</code> [1/4] . . . . .	41
5.2.2.66 <code>operator&amp;()</code> [2/4] . . . . .	41
5.2.2.67 <code>operator&amp;()</code> [3/4] . . . . .	41
5.2.2.68 <code>operator&amp;()</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&lt;&lt;()</code> [1/5] . . . . .	46
5.2.2.87 <code>operator&lt;&lt;()</code> [2/5] . . . . .	46
5.2.2.88 <code>operator&lt;&lt;()</code> [3/5] . . . . .	46
5.2.2.89 <code>operator&lt;&lt;()</code> [4/5] . . . . .	47
5.2.2.90 <code>operator&lt;&lt;()</code> [5/5] . . . . .	47
5.2.2.91 <code>operator&gt;&gt;()</code> [1/3] . . . . .	47
5.2.2.92 <code>operator&gt;&gt;()</code> [2/3] . . . . .	47
5.2.2.93 <code>operator&gt;&gt;()</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 clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::default_truncation	57
5.2.3.3 DEFAULT_HI	58
5.2.3.4 l_ln2	58
5.2.3.5 l_pi	58
5.2.3.6 MS_PER_S	58
5.2.3.7 Tuning_Fast_Basis_Max_Count	58
5.2.3.8 Tuning_Fast_CR_Sqrt_Max_Steps	58

5.2.3.9 Tuning_Fast_DB_Sqrt_Max_Steps . . . . .	59
5.2.3.10 Tuning_Fast_Div_Max_Steps . . . . .	59
5.2.3.11 Tuning_Fast_Fast_Size_Threshold . . . . .	59
5.2.3.12 Tuning_Fast_Inv_Fast_Dim_Threshold . . . . .	59
5.2.3.13 Tuning_Fast_Log_Max_Inner_Steps . . . . .	59
5.2.3.14 Tuning_Fast_Log_Max_Outer_Steps . . . . .	59
5.2.3.15 Tuning_Fast_Mult_Matrix_Threshold . . . . .	59
5.2.3.16 Tuning_Fast_Products_Size_Threshold . . . . .	59
5.2.3.17 Tuning_Int_Digits . . . . .	60
5.2.3.18 Tuning_Max_Threshold . . . . .	60
5.2.3.19 Tuning_Naive_Basis_Max_Count . . . . .	60
5.2.3.20 Tuning_Naive_Fast_Size_Threshold . . . . .	60
5.2.3.21 Tuning_Naive_Inv_Fast_Dim_Threshold . . . . .	60
5.2.3.22 Tuning_Naive_Mult_Matrix_Threshold . . . . .	60
5.2.3.23 Tuning_Slow_Basis_Max_Count . . . . .	60
5.2.3.24 Tuning_Slow_Fast_Size_Threshold . . . . .	60
5.2.3.25 Tuning_Slow_Inv_Fast_Dim_Threshold . . . . .	61
5.2.3.26 Tuning_Slow_Mult_Matrix_Threshold . . . . .	61
5.2.3.27 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 . . . . .	62
5.3.2.1 offset_to_super . . . . .	62
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 . . . . .	64
5.4.2.1 classify_eigenvalues() . . . . .	64
5.4.2.2 eigenvalues() . . . . .	64
5.4.2.3 inner() . . . . .	64
5.4.2.4 isinf() . . . . .	64
5.4.2.5 isnan() . . . . .	65
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() . . . . .	66
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() . . . . .	67
5.4.2.14 signed_perm_nork() . . . . .	67

5.4.2.15 <code>sparse_prod()</code>	67
5.4.2.16 <code>to_blaze()</code>	67
5.4.2.17 <code>trace()</code>	68
5.4.2.18 <code>unit()</code>	68
5.5 <code>glucat::timing</code> Namespace Reference	68
5.5.1 Function Documentation	68
5.5.1.1 <code>elapsed()</code>	68
5.5.2 Variable Documentation	69
5.5.2.1 <code>EXTRA_TRIALS</code>	69
5.5.2.2 <code>MS_PER_CLOCK</code>	69
5.5.2.3 <code>MS_PER_SEC</code>	69
5.6 <code>pade</code> Namespace Reference	69
5.6.1 Variable Documentation	70
5.6.1.1 <code>pade_log_denom&lt; dd_real &gt;::denom</code>	70
5.6.1.2 <code>pade_log_denom&lt; float &gt;::denom</code>	70
5.6.1.3 <code>pade_log_denom&lt; longdouble &gt;::denom</code>	71
5.6.1.4 <code>pade_log_denom&lt; qd_real &gt;::denom</code>	71
5.6.1.5 <code>pade_log_denom&lt; Scalar_T &gt;::denom</code>	71
5.6.1.6 <code>pade_log_numer&lt; dd_real &gt;::numer</code>	72
5.6.1.7 <code>pade_log_numer&lt; float &gt;::numer</code>	72
5.6.1.8 <code>pade_log_numer&lt; longdouble &gt;::numer</code>	72
5.6.1.9 <code>pade_log_numer&lt; qd_real &gt;::numer</code>	73
5.6.1.10 <code>pade_log_numer&lt; Scalar_T &gt;::numer</code>	73
5.6.1.11 <code>pade_sqrt_denom&lt; dd_real &gt;::denom</code>	73
5.6.1.12 <code>pade_sqrt_denom&lt; float &gt;::denom</code>	74
5.6.1.13 <code>pade_sqrt_denom&lt; longdouble &gt;::denom</code>	74
5.6.1.14 <code>pade_sqrt_denom&lt; qd_real &gt;::denom</code>	74
5.6.1.15 <code>pade_sqrt_denom&lt; Scalar_T &gt;::denom</code>	75
5.6.1.16 <code>pade_sqrt_numer&lt; dd_real &gt;::numer</code>	75
5.6.1.17 <code>pade_sqrt_numer&lt; float &gt;::numer</code>	75
5.6.1.18 <code>pade_sqrt_numer&lt; longdouble &gt;::numer</code>	76
5.6.1.19 <code>pade_sqrt_numer&lt; qd_real &gt;::numer</code>	76
5.6.1.20 <code>pade_sqrt_numer&lt; Scalar_T &gt;::numer</code>	76
5.7 <code>PyClical</code> Namespace Reference	77
5.7.1 Function Documentation	77
5.7.1.1 <code>_test()</code>	77
5.7.1.2 <code>clifford_hidden_doctests()</code>	78
5.7.1.3 <code>e()</code>	79
5.7.1.4 <code>index_set_hidden_doctests()</code>	79
5.7.1.5 <code>istpq()</code>	80
5.7.2 Variable Documentation	81
5.7.2.1 <code>__version__</code>	81



5.7.2.2 cl	81
5.7.2.3 fill	81
5.7.2.4 i	81
5.7.2.5 ist	81
5.7.2.6 ixt	81
5.7.2.7 lhs	81
5.7.2.8 nbar3	82
5.7.2.9 ninf3	82
5.7.2.10 None	82
5.7.2.11 obj	82
5.7.2.12 pi	82
5.7.2.13 rhs	82
5.7.2.14 scalar_epsilon	82
5.7.2.15 tau	82
5.7.2.16 threshold	83
5.7.2.17 tol	83
5.8 std Namespace Reference	83
<b>6 Class Documentation</b>	<b>85</b>
6.1 glucat::basis_table< Scalar_T, LO, HI, Matrix_T > Class Template Reference	85
6.1.1 Detailed Description	86
6.1.2 Constructor & Destructor Documentation	86
6.1.2.1 basis_table() [1/2]	86
6.1.2.2 ~basis_table()	86
6.1.2.3 basis_table() [2/2]	86
6.1.3 Member Function Documentation	87
6.1.3.1 basis()	87
6.1.3.2 operator=()	87
6.1.4 Friends And Related Symbol Documentation	87
6.1.4.1 friend_for_private_destructor	87
6.2 glucat::bool_to_type< truth_value > Class Template Reference	87
6.2.1 Detailed Description	88
6.2.2 Member Enumeration Documentation	88
6.2.2.1 anonymous enum	88
6.3 PyClical.clifford Class Reference	88
6.3.1 Detailed Description	90
6.3.2 Member Function Documentation	90
6.3.2.1 __add__()	90
6.3.2.2 __and__()	91
6.3.2.3 __call__()	91
6.3.2.4 __cinit__()	91
6.3.2.5 __contains__()	92

6.3.2.6 <code>__dealloc__()</code>	92
6.3.2.7 <code>__getitem__()</code>	93
6.3.2.8 <code>__iadd__()</code>	93
6.3.2.9 <code>__iand__()</code>	93
6.3.2.10 <code>__idiv__()</code>	94
6.3.2.11 <code>__imod__()</code>	94
6.3.2.12 <code>__imul__()</code>	94
6.3.2.13 <code>__ior__()</code>	95
6.3.2.14 <code>__isub__()</code>	95
6.3.2.15 <code>__iter__()</code>	95
6.3.2.16 <code>__ixor__()</code>	95
6.3.2.17 <code>__mod__()</code>	96
6.3.2.18 <code>__mul__()</code>	96
6.3.2.19 <code>__neg__()</code>	96
6.3.2.20 <code>__or__()</code>	97
6.3.2.21 <code>__pos__()</code>	97
6.3.2.22 <code>__pow__()</code>	97
6.3.2.23 <code>__radd__()</code>	98
6.3.2.24 <code>__rand__()</code>	98
6.3.2.25 <code>__repr__()</code>	98
6.3.2.26 <code>__richcmp__()</code>	99
6.3.2.27 <code>__rmod__()</code>	99
6.3.2.28 <code>__rmul__()</code>	99
6.3.2.29 <code>__rsub__()</code>	100
6.3.2.30 <code>__rtruediv__()</code>	100
6.3.2.31 <code>__rxor__()</code>	100
6.3.2.32 <code>__str__()</code>	100
6.3.2.33 <code>__sub__()</code>	101
6.3.2.34 <code>__truediv__()</code>	101
6.3.2.35 <code>__xor__()</code>	101
6.3.2.36 <code>abs()</code>	102
6.3.2.37 <code>conj()</code>	102
6.3.2.38 <code>even()</code>	102
6.3.2.39 <code>frame()</code>	103
6.3.2.40 <code>inv()</code>	103
6.3.2.41 <code>involute()</code>	103
6.3.2.42 <code>isinf()</code>	104
6.3.2.43 <code>isnan()</code>	104
6.3.2.44 <code>max_abs()</code>	104
6.3.2.45 <code>norm()</code>	105
6.3.2.46 <code>odd()</code>	105
6.3.2.47 <code>outer_pow()</code>	105

6.3.2.48 pow()	106
6.3.2.49 pure()	106
6.3.2.50 quad()	106
6.3.2.51 reframe()	107
6.3.2.52 reverse()	107
6.3.2.53 scalar()	107
6.3.2.54 truncated()	108
6.3.2.55 vector_part()	108
6.3.3 Member Data Documentation	108
6.3.3.1 instance	108
6.4 glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T > Class Template Reference	109
6.4.1 Detailed Description	111
6.4.2 Member Typedef Documentation	111
6.4.2.1 index_set_t	111
6.4.2.2 multivector_t	111
6.4.2.3 pair_t	112
6.4.2.4 scalar_t	112
6.4.2.5 vector_t	112
6.4.3 Constructor & Destructor Documentation	112
6.4.3.1 ~clifford_algebra()	112
6.4.4 Member Function Documentation	112
6.4.4.1 classname()	112
6.4.4.2 conj()	112
6.4.4.3 even()	113
6.4.4.4 frame()	113
6.4.4.5 grade()	113
6.4.4.6 inv()	113
6.4.4.7 involute()	114
6.4.4.8 isinf()	114
6.4.4.9 isnan()	114
6.4.4.10 max_abs()	114
6.4.4.11 norm()	115
6.4.4.12 odd()	115
6.4.4.13 operator%=( )	115
6.4.4.14 operator&=( )	115
6.4.4.15 operator()( )	115
6.4.4.16 operator*=( ) [1/2]	116
6.4.4.17 operator*=( ) [2/2]	116
6.4.4.18 operator+=( ) [1/2]	116
6.4.4.19 operator+=( ) [2/2]	116
6.4.4.20 operator-( )	116
6.4.4.21 operator-=( ) [1/2]	116

6.4.4.22 operator-=( ) [ 2/2 ] . . . . .	117
6.4.4.23 operator/=( ) [ 1/2 ] . . . . .	117
6.4.4.24 operator/=( ) [ 2/2 ] . . . . .	117
6.4.4.25 operator==( ) [ 1/2 ] . . . . .	117
6.4.4.26 operator==( ) [ 2/2 ] . . . . .	117
6.4.4.27 operator[]() . . . . .	117
6.4.4.28 operator^=( ) . . . . .	118
6.4.4.29 operator"  =() . . . . .	118
6.4.4.30 outer_pow() . . . . .	118
6.4.4.31 pow() . . . . .	118
6.4.4.32 pure() . . . . .	118
6.4.4.33 quad() . . . . .	119
6.4.4.34 reverse() . . . . .	119
6.4.4.35 scalar() . . . . .	119
6.4.4.36 truncated() . . . . .	119
6.4.4.37 vector_part() [ 1/2 ] . . . . .	120
6.4.4.38 vector_part() [ 2/2 ] . . . . .	120
6.4.4.39 write() [ 1/2 ] . . . . .	120
6.4.4.40 write() [ 2/2 ] . . . . .	120
6.4.5 Member Data Documentation . . . . .	121
6.4.5.1 default_truncation . . . . .	121
6.4.5.2 v_hi . . . . .	121
6.4.5.3 v_lo . . . . .	121
6.5 glucat::compare_types< LHS_T, RHS_T > Class Template Reference . . . . .	121
6.5.1 Detailed Description . . . . .	122
6.5.2 Member Enumeration Documentation . . . . .	122
6.5.2.1 anonymous enum . . . . .	122
6.6 glucat::compare_types< T, T > Class Template Reference . . . . .	123
6.6.1 Detailed Description . . . . .	123
6.6.2 Member Enumeration Documentation . . . . .	123
6.6.2.1 anonymous enum . . . . .	123
6.6.2.2 anonymous enum . . . . .	123
6.7 glucat::control_t Class Reference . . . . .	124
6.7.1 Detailed Description . . . . .	125
6.7.2 Constructor & Destructor Documentation . . . . .	125
6.7.2.1 control_t() [ 1/3 ] . . . . .	125
6.7.2.2 control_t() [ 2/3 ] . . . . .	125
6.7.2.3 ~control_t() . . . . .	125
6.7.2.4 control_t() [ 3/3 ] . . . . .	125
6.7.3 Member Function Documentation . . . . .	125
6.7.3.1 call() [ 1/2 ] . . . . .	125
6.7.3.2 call() [ 2/2 ] . . . . .	126

6.7.3.3 <code>catch_exceptions()</code> . . . . .	126
6.7.3.4 <code>control()</code> . . . . .	126
6.7.3.5 <code>operator=()</code> . . . . .	126
6.7.3.6 <code>valid()</code> . . . . .	126
6.7.3.7 <code>verbose()</code> . . . . .	127
6.7.4 Friends And Related Symbol Documentation . . . . .	127
6.7.4.1 <code>friend_for_private_destructor</code> . . . . .	127
6.7.5 Member Data Documentation . . . . .	127
6.7.5.1 <code>m_catch_exceptions</code> . . . . .	127
6.7.5.2 <code>m_valid</code> . . . . .	127
6.7.5.3 <code>m_verbose_output</code> . . . . .	128
6.8 <code>glucat::CTAssertion&lt; bool &gt;</code> Struct Template Reference . . . . .	128
6.8.1 Detailed Description . . . . .	128
6.9 <code>glucat::CTAssertion&lt; true &gt;</code> Struct Reference . . . . .	128
6.9.1 Detailed Description . . . . .	128
6.10 <code>glucat::numeric_traits&lt; Scalar_T &gt;::demoted</code> Struct Reference . . . . .	129
6.10.1 Detailed Description . . . . .	129
6.10.2 Member Typedef Documentation . . . . .	129
6.10.2.1 <code>type</code> [1/2] . . . . .	129
6.10.2.2 <code>type</code> [2/2] . . . . .	129
6.11 <code>glucat::matrix::eig_genus&lt; Matrix_T &gt;</code> Struct Template Reference . . . . .	129
6.11.1 Detailed Description . . . . .	130
6.11.2 Member Typedef Documentation . . . . .	130
6.11.2.1 <code>Scalar_T</code> . . . . .	130
6.11.3 Member Data Documentation . . . . .	130
6.11.3.1 <code>m_eig_case</code> . . . . .	130
6.11.3.2 <code>m_is_singular</code> . . . . .	131
6.11.3.3 <code>m_safe_arg</code> . . . . .	131
6.12 <code>glucat::error&lt; Class_T &gt;</code> Class Template Reference . . . . .	131
6.12.1 Detailed Description . . . . .	132
6.12.2 Constructor & Destructor Documentation . . . . .	133
6.12.2.1 <code>error()</code> [1/2] . . . . .	133
6.12.2.2 <code>error()</code> [2/2] . . . . .	133
6.12.3 Member Function Documentation . . . . .	133
6.12.3.1 <code>classname()</code> . . . . .	133
6.12.3.2 <code>heading()</code> . . . . .	133
6.12.3.3 <code>print_error_msg()</code> . . . . .	134
6.13 <code>glucat::framed_multi&lt; Scalar_T, LO, HI, Tune_P &gt;</code> Class Template Reference . . . . .	134
6.13.1 Detailed Description . . . . .	139
6.13.2 Member Typedef Documentation . . . . .	139
6.13.2.1 <code>const_iterator</code> . . . . .	139
6.13.2.2 <code>error_t</code> . . . . .	139

6.13.2.3 framed_multi_t . . . . .	139
6.13.2.4 framed_pair_t . . . . .	140
6.13.2.5 index_set_t . . . . .	140
6.13.2.6 iterator . . . . .	140
6.13.2.7 map_t . . . . .	140
6.13.2.8 matrix_multi_t . . . . .	140
6.13.2.9 matrix_t . . . . .	140
6.13.2.10 multivector_t . . . . .	141
6.13.2.11 scalar_t . . . . .	141
6.13.2.12 size_type . . . . .	141
6.13.2.13 sorted_map_t . . . . .	141
6.13.2.14 term_t . . . . .	141
6.13.2.15 tune_p . . . . .	141
6.13.2.16 var_term_t . . . . .	142
6.13.2.17 vector_t . . . . .	142
6.13.3 Constructor & Destructor Documentation . . . . .	142
6.13.3.1 ~framed_multi() . . . . .	142
6.13.3.2 framed_multi() [1/15] . . . . .	142
6.13.3.3 framed_multi() [2/15] . . . . .	142
6.13.3.4 framed_multi() [3/15] . . . . .	143
6.13.3.5 framed_multi() [4/15] . . . . .	143
6.13.3.6 framed_multi() [5/15] . . . . .	143
6.13.3.7 framed_multi() [6/15] . . . . .	143
6.13.3.8 framed_multi() [7/15] . . . . .	144
6.13.3.9 framed_multi() [8/15] . . . . .	144
6.13.3.10 framed_multi() [9/15] . . . . .	144
6.13.3.11 framed_multi() [10/15] . . . . .	144
6.13.3.12 framed_multi() [11/15] . . . . .	145
6.13.3.13 framed_multi() [12/15] . . . . .	145
6.13.3.14 framed_multi() [13/15] . . . . .	145
6.13.3.15 framed_multi() [14/15] . . . . .	145
6.13.3.16 framed_multi() [15/15] . . . . .	146
6.13.4 Member Function Documentation . . . . .	146
6.13.4.1 centre_pm4_qp4() . . . . .	146
6.13.4.2 centre_pp4_qm4() . . . . .	146
6.13.4.3 centre_qp1_pm1() . . . . .	147
6.13.4.4 classname() . . . . .	147
6.13.4.5 divide() . . . . .	147
6.13.4.6 fast() . . . . .	147
6.13.4.7 fast_framed_multi() . . . . .	148
6.13.4.8 fast_matrix_multi() . . . . .	148
6.13.4.9 fold() . . . . .	148

6.13.4.10 <code>nbr_terms()</code>	148
6.13.4.11 <code>operator+=()</code>	149
6.13.4.12 <code>random()</code>	149
6.13.4.13 <code>unfold()</code>	149
6.13.5 Friends And Related Symbol Documentation	149
6.13.5.1 <code>exp</code>	149
6.13.5.2 <code>framed_multi</code>	150
6.13.5.3 <code>matrix_multi</code>	150
6.13.5.4 <code>operator%</code>	150
6.13.5.5 <code>operator&amp;</code>	150
6.13.5.6 <code>operator*</code>	150
6.13.5.7 <code>operator/</code>	151
6.13.5.8 <code>operator&lt;&lt;</code> [1/2]	151
6.13.5.9 <code>operator&lt;&lt;</code> [2/2]	151
6.13.5.10 <code>operator&gt;&gt;</code>	151
6.13.5.11 <code>operator^</code>	151
6.13.5.12 <code>operator"  </code>	151
6.13.5.13 <code>star</code>	152
6.14 <code>glucat::gen::generator_table&lt; Matrix_T &gt;</code> Class Template Reference	152
6.14.1 Detailed Description	153
6.14.2 Constructor & Destructor Documentation	153
6.14.2.1 <code>generator_table()</code> [1/2]	153
6.14.2.2 <code>~generator_table()</code>	154
6.14.2.3 <code>generator_table()</code> [2/2]	154
6.14.3 Member Function Documentation	154
6.14.3.1 <code>gen_from_pm1_qm1()</code>	154
6.14.3.2 <code>gen_from_pm4_qp4()</code>	154
6.14.3.3 <code>gen_from_pp4_qm4()</code>	155
6.14.3.4 <code>gen_from_qp1_pm1()</code>	155
6.14.3.5 <code>gen_vector()</code>	155
6.14.3.6 <code>generator()</code>	155
6.14.3.7 <code>operator()()</code>	156
6.14.3.8 <code>operator=()</code>	156
6.14.4 Friends And Related Symbol Documentation	156
6.14.4.1 <code>friend_for_private_destructor</code>	156
6.15 <code>glucat::glucat_error</code> Class Reference	157
6.15.1 Detailed Description	158
6.15.2 Constructor & Destructor Documentation	158
6.15.2.1 <code>glucat_error()</code>	158
6.15.2.2 <code>~glucat_error()</code>	158
6.15.3 Member Function Documentation	158
6.15.3.1 <code>classname()</code>	158

6.15.3.2 heading()	158
6.15.3.3 print_error_msg()	159
6.15.4 Member Data Documentation	159
6.15.4.1 name	159
6.16 glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t Class Reference	159
6.16.1 Detailed Description	159
6.16.2 Constructor & Destructor Documentation	160
6.16.2.1 hash_size_t()	160
6.16.3 Member Function Documentation	160
6.16.3.1 operator()()	160
6.16.4 Member Data Documentation	160
6.16.4.1 n	160
6.17 glucat::index_set< LO, HI > Class Template Reference	161
6.17.1 Detailed Description	164
6.17.2 Member Typedef Documentation	164
6.17.2.1 bitset_t	164
6.17.2.2 error_t	164
6.17.2.3 index_pair_t	164
6.17.2.4 index_set_t	164
6.17.3 Constructor & Destructor Documentation	165
6.17.3.1 index_set() [1/6]	165
6.17.3.2 index_set() [2/6]	165
6.17.3.3 index_set() [3/6]	165
6.17.3.4 index_set() [4/6]	165
6.17.3.5 index_set() [5/6]	166
6.17.3.6 index_set() [6/6]	166
6.17.4 Member Function Documentation	166
6.17.4.1 BOOST_STATIC_ASSERT()	166
6.17.4.2 classname()	166
6.17.4.3 count()	166
6.17.4.4 count_neg()	167
6.17.4.5 count_pos()	167
6.17.4.6 flip() [1/2]	167
6.17.4.7 flip() [2/2]	167
6.17.4.8 fold() [1/2]	167
6.17.4.9 fold() [2/2]	168
6.17.4.10 hash_fn()	168
6.17.4.11 is_contiguous()	168
6.17.4.12 lex_less_than()	168
6.17.4.13 max()	169
6.17.4.14 min()	169
6.17.4.15 operator"!=()	169



6.17.4.16 operator&=()	169
6.17.4.17 operator<()	170
6.17.4.18 operator==()	170
6.17.4.19 operator[]() [1/2]	170
6.17.4.20 operator[]() [2/2]	170
6.17.4.21 operator^=()	170
6.17.4.22 operator"  =()	171
6.17.4.23 operator~()	171
6.17.4.24 reset() [1/2]	171
6.17.4.25 reset() [2/2]	171
6.17.4.26 set() [1/3]	171
6.17.4.27 set() [2/3]	172
6.17.4.28 set() [3/3]	172
6.17.4.29 sign_of_mult()	172
6.17.4.30 sign_of_square()	172
6.17.4.31 test()	172
6.17.4.32 unfold()	173
6.17.4.33 value_of_fold()	173
6.17.5 Friends And Related Symbol Documentation	173
6.17.5.1 compare	173
6.17.5.2 operator&	173
6.17.5.3 operator^	173
6.17.5.4 operator"	174
6.17.5.5 reference	174
6.17.6 Member Data Documentation	174
6.17.6.1 v_hi	174
6.17.6.2 v_lo	174
6.18 PyClical.index_set Class Reference	175
6.18.1 Detailed Description	176
6.18.2 Member Function Documentation	176
6.18.2.1 __and__()	176
6.18.2.2 __cinit__()	176
6.18.2.3 __contains__()	177
6.18.2.4 __dealloc__()	177
6.18.2.5 __getitem__()	177
6.18.2.6 __iand__()	178
6.18.2.7 __invert__()	178
6.18.2.8 __ior__()	178
6.18.2.9 __iter__()	178
6.18.2.10 __ixor__()	179
6.18.2.11 __or__()	179
6.18.2.12 __repr__()	179

6.18.2.13 <code>__richcmp__()</code>	180
6.18.2.14 <code>__setitem__()</code>	180
6.18.2.15 <code>__str__()</code>	180
6.18.2.16 <code>__xor__()</code>	181
6.18.2.17 <code>count()</code>	181
6.18.2.18 <code>count_neg()</code>	181
6.18.2.19 <code>count_pos()</code>	182
6.18.2.20 <code>hash_fn()</code>	182
6.18.2.21 <code>max()</code>	182
6.18.2.22 <code>min()</code>	183
6.18.2.23 <code>sign_of_mult()</code>	183
6.18.2.24 <code>sign_of_square()</code>	183
6.18.3 Member Data Documentation	184
6.18.3.1 instance	184
6.19 <code>glucat::index_set_hash&lt; LO, HI &gt;</code> Class Template Reference	184
6.19.1 Detailed Description	184
6.19.2 Member Typedef Documentation	184
6.19.2.1 <code>index_set_t</code>	184
6.19.3 Member Function Documentation	185
6.19.3.1 <code>operator&gt;()()</code>	185
6.20 <code>glucat::matrix_multi&lt; Scalar_T, LO, HI, Tune_P &gt;</code> Class Template Reference	185
6.20.1 Detailed Description	190
6.20.2 Member Typedef Documentation	190
6.20.2.1 <code>basis_matrix_t</code>	190
6.20.2.2 <code>error_t</code>	190
6.20.2.3 <code>framed_multi_t</code>	190
6.20.2.4 <code>index_set_t</code>	191
6.20.2.5 <code>matrix_index_t</code>	191
6.20.2.6 <code>matrix_multi_t</code>	191
6.20.2.7 <code>matrix_t</code>	191
6.20.2.8 <code>multivector_t</code>	191
6.20.2.9 <code>orientation_t</code>	191
6.20.2.10 <code>scalar_t</code>	192
6.20.2.11 <code>term_t</code>	192
6.20.2.12 <code>tune_p</code>	192
6.20.2.13 <code>vector_t</code>	192
6.20.3 Constructor & Destructor Documentation	192
6.20.3.1 <code>~matrix_multi()</code>	192
6.20.3.2 <code>matrix_multi()</code> [1/17]	192
6.20.3.3 <code>matrix_multi()</code> [2/17]	193
6.20.3.4 <code>matrix_multi()</code> [3/17]	193
6.20.3.5 <code>matrix_multi()</code> [4/17]	193

6.20.3.6 <a href="#">matrix_multi()</a> [5/17]	193
6.20.3.7 <a href="#">matrix_multi()</a> [6/17]	194
6.20.3.8 <a href="#">matrix_multi()</a> [7/17]	194
6.20.3.9 <a href="#">matrix_multi()</a> [8/17]	194
6.20.3.10 <a href="#">matrix_multi()</a> [9/17]	194
6.20.3.11 <a href="#">matrix_multi()</a> [10/17]	195
6.20.3.12 <a href="#">matrix_multi()</a> [11/17]	195
6.20.3.13 <a href="#">matrix_multi()</a> [12/17]	195
6.20.3.14 <a href="#">matrix_multi()</a> [13/17]	195
6.20.3.15 <a href="#">matrix_multi()</a> [14/17]	196
6.20.3.16 <a href="#">matrix_multi()</a> [15/17]	196
6.20.3.17 <a href="#">matrix_multi()</a> [16/17]	196
6.20.3.18 <a href="#">matrix_multi()</a> [17/17]	196
6.20.4 Member Function Documentation	197
6.20.4.1 <a href="#">basis_element()</a>	197
6.20.4.2 <a href="#">classname()</a>	197
6.20.4.3 <a href="#">fast_framed_multi()</a>	197
6.20.4.4 <a href="#">fast_matrix_multi()</a>	197
6.20.4.5 <a href="#">operator+=()</a>	198
6.20.4.6 <a href="#">operator=()</a>	198
6.20.4.7 <a href="#">random()</a>	198
6.20.5 Friends And Related Symbol Documentation	198
6.20.5.1 <a href="#">framed_multi</a>	198
6.20.5.2 <a href="#">matrix_log</a>	199
6.20.5.3 <a href="#">matrix_multi</a>	199
6.20.5.4 <a href="#">matrix_sqrt</a>	199
6.20.5.5 <a href="#">operator%</a>	199
6.20.5.6 <a href="#">operator&amp;</a>	199
6.20.5.7 <a href="#">operator*</a>	200
6.20.5.8 <a href="#">operator/</a>	200
6.20.5.9 <a href="#">operator&lt;&lt;</a>	200
6.20.5.10 <a href="#">operator&gt;&gt;</a>	200
6.20.5.11 <a href="#">operator^</a>	200
6.20.5.12 <a href="#">operator"  </a>	200
6.20.5.13 <a href="#">reframe</a>	201
6.20.5.14 <a href="#">star</a>	201
6.20.6 Member Data Documentation	201
6.20.6.1 <a href="#">m_frame</a>	201
6.20.6.2 <a href="#">m_matrix</a>	201
6.21 <a href="#">std::numeric_limits&lt; glucat::framed_multi&lt; Scalar_T, LO, HI, Tune_P &gt; &gt; Struct Template Reference</a>	202
6.21.1 Detailed Description	202
6.22 <a href="#">std::numeric_limits&lt; glucat::matrix_multi&lt; Scalar_T, LO, HI, Tune_P &gt; &gt; Struct Template Reference</a>	203

6.22.1 Detailed Description	203
6.23 <code>glucat::numeric_traits&lt; Scalar_T &gt;</code> Class Template Reference	204
6.23.1 Detailed Description	206
6.23.2 Member Function Documentation	206
6.23.2.1 <code>abs()</code>	206
6.23.2.2 <code>acos()</code>	206
6.23.2.3 <code>asin()</code>	206
6.23.2.4 <code>atan()</code>	207
6.23.2.5 <code>conj()</code>	207
6.23.2.6 <code>cos()</code>	207
6.23.2.7 <code>cosh()</code>	207
6.23.2.8 <code>exp()</code>	207
6.23.2.9 <code>fmod()</code>	208
6.23.2.10 <code>imag()</code>	208
6.23.2.11 <code>isInf()</code> [1/3]	208
6.23.2.12 <code>isInf()</code> [2/3]	208
6.23.2.13 <code>isInf()</code> [3/3]	209
6.23.2.14 <code>isNaN()</code> [1/3]	209
6.23.2.15 <code>isNaN()</code> [2/3]	209
6.23.2.16 <code>isNaN()</code> [3/3]	209
6.23.2.17 <code>isNaN_or_isInf()</code>	210
6.23.2.18 <code>ln_2()</code> [1/2]	210
6.23.2.19 <code>ln_2()</code> [2/2]	210
6.23.2.20 <code>log()</code>	210
6.23.2.21 <code>log2()</code>	211
6.23.2.22 <code>NaN()</code>	211
6.23.2.23 <code>pi()</code> [1/2]	211
6.23.2.24 <code>pi()</code> [2/2]	211
6.23.2.25 <code>pow()</code>	212
6.23.2.26 <code>real()</code>	212
6.23.2.27 <code>sin()</code>	212
6.23.2.28 <code>sinh()</code>	212
6.23.2.29 <code>sqrt()</code>	212
6.23.2.30 <code>tan()</code>	213
6.23.2.31 <code>tanh()</code>	213
6.23.2.32 <code>to_double()</code>	213
6.23.2.33 <code>to_int()</code>	213
6.23.2.34 <code>to_scalar_t()</code> [1/9]	213
6.23.2.35 <code>to_scalar_t()</code> [2/9]	214
6.23.2.36 <code>to_scalar_t()</code> [3/9]	214
6.23.2.37 <code>to_scalar_t()</code> [4/9]	214
6.23.2.38 <code>to_scalar_t()</code> [5/9]	214

6.23.2.39 to_scalar_t() [ 6/9] . . . . .	214
6.23.2.40 to_scalar_t() [ 7/9] . . . . .	215
6.23.2.41 to_scalar_t() [ 8/9] . . . . .	215
6.23.2.42 to_scalar_t() [ 9/9] . . . . .	215
6.24 pade::pade_log_denom< Scalar_T > Struct Template Reference . . . . .	215
6.24.1 Detailed Description . . . . .	216
6.24.2 Member Typedef Documentation . . . . .	216
6.24.2.1 array . . . . .	216
6.24.3 Member Data Documentation . . . . .	216
6.24.3.1 denom . . . . .	216
6.25 pade::pade_log_denom< dd_real > Struct Reference . . . . .	216
6.25.1 Detailed Description . . . . .	217
6.25.2 Member Typedef Documentation . . . . .	217
6.25.2.1 array . . . . .	217
6.25.3 Member Data Documentation . . . . .	217
6.25.3.1 denom . . . . .	217
6.26 pade::pade_log_denom< float > Struct Reference . . . . .	217
6.26.1 Detailed Description . . . . .	217
6.26.2 Member Typedef Documentation . . . . .	218
6.26.2.1 array . . . . .	218
6.26.3 Member Data Documentation . . . . .	218
6.26.3.1 denom . . . . .	218
6.27 pade::pade_log_denom< long double > Struct Reference . . . . .	218
6.27.1 Detailed Description . . . . .	218
6.27.2 Member Typedef Documentation . . . . .	218
6.27.2.1 array . . . . .	218
6.27.3 Member Data Documentation . . . . .	219
6.27.3.1 denom . . . . .	219
6.28 pade::pade_log_denom< qd_real > Struct Reference . . . . .	219
6.28.1 Detailed Description . . . . .	219
6.28.2 Member Typedef Documentation . . . . .	219
6.28.2.1 array . . . . .	219
6.28.3 Member Data Documentation . . . . .	219
6.28.3.1 denom . . . . .	219
6.29 pade::pade_log_numer< Scalar_T > Struct Template Reference . . . . .	220
6.29.1 Detailed Description . . . . .	220
6.29.2 Member Typedef Documentation . . . . .	220
6.29.2.1 array . . . . .	220
6.29.3 Member Data Documentation . . . . .	220
6.29.3.1 numer . . . . .	220
6.30 pade::pade_log_numer< dd_real > Struct Reference . . . . .	221
6.30.1 Detailed Description . . . . .	221

6.30.2 Member Typedef Documentation	221
6.30.2.1 array	221
6.30.3 Member Data Documentation	221
6.30.3.1 numer	221
6.31 pade::pade_log_numer< float > Struct Reference	221
6.31.1 Detailed Description	222
6.31.2 Member Typedef Documentation	222
6.31.2.1 array	222
6.31.3 Member Data Documentation	222
6.31.3.1 numer	222
6.32 pade::pade_log_numer< long double > Struct Reference	222
6.32.1 Detailed Description	222
6.32.2 Member Typedef Documentation	223
6.32.2.1 array	223
6.32.3 Member Data Documentation	223
6.32.3.1 numer	223
6.33 pade::pade_log_numer< qd_real > Struct Reference	223
6.33.1 Detailed Description	223
6.33.2 Member Typedef Documentation	223
6.33.2.1 array	223
6.33.3 Member Data Documentation	224
6.33.3.1 numer	224
6.34 pade::pade_sqrt_denom< Scalar_T > Struct Template Reference	224
6.34.1 Detailed Description	224
6.34.2 Member Typedef Documentation	224
6.34.2.1 array	224
6.34.3 Member Data Documentation	225
6.34.3.1 denom	225
6.35 pade::pade_sqrt_denom< dd_real > Struct Reference	225
6.35.1 Detailed Description	225
6.35.2 Member Typedef Documentation	225
6.35.2.1 array	225
6.35.3 Member Data Documentation	225
6.35.3.1 denom	225
6.36 pade::pade_sqrt_denom< float > Struct Reference	226
6.36.1 Detailed Description	226
6.36.2 Member Typedef Documentation	226
6.36.2.1 array	226
6.36.3 Member Data Documentation	226
6.36.3.1 denom	226
6.37 pade::pade_sqrt_denom< long double > Struct Reference	226
6.37.1 Detailed Description	227

6.37.2 Member Typedef Documentation	227
6.37.2.1 array	227
6.37.3 Member Data Documentation	227
6.37.3.1 denom	227
6.38 pade::pade_sqrt_denom< qd_real > Struct Reference	227
6.38.1 Detailed Description	227
6.38.2 Member Typedef Documentation	228
6.38.2.1 array	228
6.38.3 Member Data Documentation	228
6.38.3.1 denom	228
6.39 pade::pade_sqrt_numer< Scalar_T > Struct Template Reference	228
6.39.1 Detailed Description	228
6.39.2 Member Typedef Documentation	229
6.39.2.1 array	229
6.39.3 Member Data Documentation	229
6.39.3.1 numer	229
6.40 pade::pade_sqrt_numer< dd_real > Struct Reference	229
6.40.1 Detailed Description	229
6.40.2 Member Typedef Documentation	229
6.40.2.1 array	229
6.40.3 Member Data Documentation	230
6.40.3.1 numer	230
6.41 pade::pade_sqrt_numer< float > Struct Reference	230
6.41.1 Detailed Description	230
6.41.2 Member Typedef Documentation	230
6.41.2.1 array	230
6.41.3 Member Data Documentation	230
6.41.3.1 numer	230
6.42 pade::pade_sqrt_numer< long double > Struct Reference	231
6.42.1 Detailed Description	231
6.42.2 Member Typedef Documentation	231
6.42.2.1 array	231
6.42.3 Member Data Documentation	231
6.42.3.1 numer	231
6.43 pade::pade_sqrt_numer< qd_real > Struct Reference	231
6.43.1 Detailed Description	232
6.43.2 Member Typedef Documentation	232
6.43.2.1 array	232
6.43.3 Member Data Documentation	232
6.43.3.1 numer	232
6.44 glucat::numeric_traits< Scalar_T >::promoted Struct Reference	232
6.44.1 Detailed Description	233

6.44.2 Member Typedef Documentation	233
6.44.2.1 type [1/3]	233
6.44.2.2 type [2/3]	233
6.44.2.3 type [3/3]	233
6.45 glucat::random_generator< Scalar_T > Class Template Reference	233
6.45.1 Detailed Description	234
6.45.2 Constructor & Destructor Documentation	234
6.45.2.1 random_generator() [1/2]	234
6.45.2.2 random_generator() [2/2]	235
6.45.2.3 ~random_generator()	235
6.45.3 Member Function Documentation	235
6.45.3.1 generator()	235
6.45.3.2 normal()	235
6.45.3.3 operator=()	235
6.45.3.4 uniform()	236
6.45.4 Friends And Related Symbol Documentation	236
6.45.4.1 friend_for_private_destructor	236
6.45.5 Member Data Documentation	236
6.45.5.1 normal_dist	236
6.45.5.2 seed	236
6.45.5.3 uint_gen	236
6.45.5.4 uniform_dist	237
6.46 glucat::index_set< LO, HI >::reference Class Reference	237
6.46.1 Detailed Description	238
6.46.2 Constructor & Destructor Documentation	238
6.46.2.1 reference() [1/2]	238
6.46.2.2 reference() [2/2]	238
6.46.2.3 ~reference()	239
6.46.3 Member Function Documentation	239
6.46.3.1 flip()	239
6.46.3.2 operator bool()	239
6.46.3.3 operator=() [1/2]	239
6.46.3.4 operator=() [2/2]	240
6.46.3.5 operator==()	240
6.46.3.6 operator~()	240
6.46.4 Friends And Related Symbol Documentation	240
6.46.4.1 index_set	240
6.46.5 Member Data Documentation	241
6.46.5.1 m_idx	241
6.46.5.2 m_pst	241
6.47 glucat::sorted_range< Map_T, Sorted_Map_T > Class Template Reference	241
6.47.1 Detailed Description	242



6.47.2 Member Typedef Documentation	242
6.47.2.1 map_t	242
6.47.2.2 sorted_iterator	242
6.47.2.3 sorted_map_t	242
6.47.3 Constructor & Destructor Documentation	242
6.47.3.1 sorted_range()	242
6.47.4 Member Data Documentation	243
6.47.4.1 sorted_begin	243
6.47.4.2 sorted_end	243
6.48 glucat::sorted_range< Sorted_Map_T, Sorted_Map_T > Class Template Reference	243
6.48.1 Detailed Description	244
6.48.2 Member Typedef Documentation	244
6.48.2.1 map_t	244
6.48.2.2 sorted_iterator	244
6.48.2.3 sorted_map_t	244
6.48.3 Constructor & Destructor Documentation	244
6.48.3.1 sorted_range()	244
6.48.4 Member Data Documentation	245
6.48.4.1 sorted_begin	245
6.48.4.2 sorted_end	245
6.49 glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term Class Reference	245
6.49.1 Detailed Description	246
6.49.2 Member Typedef Documentation	247
6.49.2.1 var_pair_t	247
6.49.3 Constructor & Destructor Documentation	247
6.49.3.1 ~var_term()	247
6.49.3.2 var_term() [1/2]	247
6.49.3.3 var_term() [2/2]	247
6.49.4 Member Function Documentation	247
6.49.4.1 classname()	247
6.49.4.2 operator*=( )	248
<b>7 File Documentation</b>	<b>249</b>
7.1 glucat/clifford_algebra.h File Reference	249
7.1.1 Macro Definition Documentation	257
7.1.1.1 _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS	257
7.2 clifford_algebra.h	257
7.3 glucat/clifford_algebra_imp.h File Reference	266
7.4 clifford_algebra_imp.h	273
7.5 glucat/errors.h File Reference	286
7.6 errors.h	287
7.7 glucat/errors_imp.h File Reference	287

7.8 errors_imp.h . . . . .	288
7.9 glucat/framed_multi.h File Reference . . . . .	289
7.10 framed_multi.h . . . . .	292
7.11 glucat/framed_multi_imp.h File Reference . . . . .	296
7.11.1 Macro Definition Documentation . . . . .	298
7.11.1.1 _GLUCAT_HASH_N . . . . .	298
7.11.1.2 _GLUCAT_HASH_SIZE_T . . . . .	298
7.12 framed_multi_imp.h . . . . .	299
7.13 glucat/generation.h File Reference . . . . .	319
7.14 generation.h . . . . .	320
7.15 glucat/generation_imp.h File Reference . . . . .	321
7.16 generation_imp.h . . . . .	322
7.17 glucat/global.h File Reference . . . . .	325
7.17.1 Macro Definition Documentation . . . . .	327
7.17.1.1 _GLUCAT_CTAssert . . . . .	327
7.18 global.h . . . . .	327
7.19 glucat/glucat.h File Reference . . . . .	328
7.20 glucat.h . . . . .	329
7.21 glucat/glucat_config.h File Reference . . . . .	330
7.21.1 Macro Definition Documentation . . . . .	331
7.21.1.1 GLUCAT_HAVE_CXX11 . . . . .	331
7.21.1.2 GLUCAT_HAVE_INTTYPES_H . . . . .	331
7.21.1.3 GLUCAT_HAVE_STDINT_H . . . . .	331
7.21.1.4 GLUCAT_HAVE_STDIO_H . . . . .	331
7.21.1.5 GLUCAT_HAVE_STDLIB_H . . . . .	332
7.21.1.6 GLUCAT_HAVE_STRING_H . . . . .	332
7.21.1.7 GLUCAT_HAVE_STRINGS_H . . . . .	332
7.21.1.8 GLUCAT_HAVE_SYS_STAT_H . . . . .	332
7.21.1.9 GLUCAT_HAVE_SYS_TYPES_H . . . . .	332
7.21.1.10 GLUCAT_HAVE_UNISTD_H . . . . .	332
7.21.1.11 GLUCAT_PACKAGE . . . . .	332
7.21.1.12 GLUCAT_PACKAGE_BUGREPORT . . . . .	332
7.21.1.13 GLUCAT_PACKAGE_NAME . . . . .	333
7.21.1.14 GLUCAT_PACKAGE_STRING . . . . .	333
7.21.1.15 GLUCAT_PACKAGE_TARNAME . . . . .	333
7.21.1.16 GLUCAT_PACKAGE_URL . . . . .	333
7.21.1.17 GLUCAT_PACKAGE_VERSION . . . . .	333
7.21.1.18 GLUCAT_STDC_HEADERS . . . . .	333
7.21.1.19 GLUCAT_VERSION . . . . .	333
7.22 glucat_config.h . . . . .	334
7.23 glucat/glucat_imp.h File Reference . . . . .	335
7.24 glucat_imp.h . . . . .	336

7.25 glucat/index_set.h File Reference . . . . .	337
7.26 index_set.h . . . . .	338
7.27 glucat/index_set_imp.h File Reference . . . . .	341
7.28 index_set_imp.h . . . . .	342
7.29 glucat/long_double.h File Reference . . . . .	354
7.30 long_double.h . . . . .	355
7.31 glucat/matrix.h File Reference . . . . .	356
7.32 matrix.h . . . . .	358
7.33 glucat/matrix_imp.h File Reference . . . . .	359
7.34 matrix_imp.h . . . . .	361
7.35 glucat/matrix_multi.h File Reference . . . . .	368
7.36 matrix_multi.h . . . . .	370
7.37 glucat/matrix_multi_imp.h File Reference . . . . .	374
7.38 matrix_multi_imp.h . . . . .	378
7.39 glucat/portability.h File Reference . . . . .	402
7.39.1 Macro Definition Documentation . . . . .	403
7.39.1.1 _GLUCAT_ISINF . . . . .	403
7.39.1.2 _GLUCAT_ISNAN . . . . .	403
7.39.1.3 UBLAS_ABS . . . . .	403
7.39.1.4 UBLAS_SQRT . . . . .	404
7.40 portability.h . . . . .	404
7.41 glucat/promotion.h File Reference . . . . .	404
7.42 promotion.h . . . . .	406
7.43 glucat/qd.h File Reference . . . . .	408
7.44 qd.h . . . . .	409
7.45 glucat/random.h File Reference . . . . .	413
7.46 random.h . . . . .	414
7.47 glucat/scalar.h File Reference . . . . .	415
7.48 scalar.h . . . . .	416
7.49 glucat/scalar_imp.h File Reference . . . . .	419
7.50 scalar_imp.h . . . . .	420
7.51 glucat/tuning.h File Reference . . . . .	422
7.51.1 Function Documentation . . . . .	423
7.51.1.1 _GLUCAT_CTAssert() . . . . .	423
7.52 tuning.h . . . . .	424
7.53 test/tuning.h File Reference . . . . .	426
7.54 tuning.h . . . . .	427
7.55 pyclical/glucat.pxd File Reference . . . . .	428
7.56 glucat.pxd . . . . .	428
7.57 pyclical/PyClical.h File Reference . . . . .	430
7.57.1 Typedef Documentation . . . . .	431
7.57.1.1 Clifford . . . . .	431

7.57.1.2 IndexSet . . . . .	431
7.57.1.3 scalar_t . . . . .	432
7.57.1.4 String . . . . .	432
7.57.2 Function Documentation . . . . .	432
7.57.2.1 clifford_to_repr() . . . . .	432
7.57.2.2 clifford_to_str() . . . . .	432
7.57.2.3 index_set_to_repr() . . . . .	432
7.57.2.4 index_set_to_str() . . . . .	433
7.57.2.5 PyFloat_FromDouble() . . . . .	433
7.57.3 Variable Documentation . . . . .	433
7.57.3.1 epsilon . . . . .	433
7.57.3.2 glucat_package_version . . . . .	433
7.57.3.3 hi_ndx . . . . .	433
7.57.3.4 lo_ndx . . . . .	434
7.58 PyClical.h . . . . .	434
7.59 pyclical/PyClical.pxd File Reference . . . . .	436
7.60 PyClical.pxd . . . . .	436
7.61 pyclical/PyClical.pyx File Reference . . . . .	436
7.62 PyClical.pyx . . . . .	437
7.63 test/control.h File Reference . . . . .	461
7.64 control.h . . . . .	462
7.65 test/driver.h File Reference . . . . .	463
7.66 driver.h . . . . .	464
7.67 test/timing.h File Reference . . . . .	464
7.68 timing.h . . . . .	465
7.69 test/try_catch.h File Reference . . . . .	466
7.70 try_catch.h . . . . .	466
7.71 test/undefine.h File Reference . . . . .	467
7.72 undefine.h . . . . .	467

<b>Index</b>	<b>469</b>
--------------	------------

# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">cga3</a>	Definitions for 3D Conformal Geometric Algebra [DL]	9
<a href="#">glucat</a>		10
<a href="#">glucat::gen</a>		61
<a href="#">glucat::matrix</a>		62
<a href="#">glucat::timing</a>		68
<a href="#">pade</a>		69
<a href="#">PyClical</a>		77
<a href="#">std</a>		83



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

std::bitset	
glucat::index_set< lo_ndx, hi_ndx > . . . . .	161
glucat::index_set< DEFAULT_LO, DEFAULT_HI > . . . . .	161
glucat::index_set< LO, HI > . . . . .	161
glucat::bool_to_type< truth_value > . . . . .	87
cdef	
PyClical.clifford . . . . .	88
PyClical.index_set . . . . .	175
Clifford	
PyClical.clifford . . . . .	88
glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T > . . . . .	109
glucat::framed_multi< Scalar_T, LO, HI, Tune_P > . . . . .	134
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > . . . . .	185
glucat::matrix_multi< scalar_t, lo_ndx, hi_ndx, tuning_promoted > . . . . .	185
glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, framed_multi< double, DEFAULT_LO, DEFAULT_HI, tuning<>> > . . . . .	109
glucat::framed_multi< Scalar_T, LO, HI, Tune_P > . . . . .	134
glucat::clifford_algebra< double, index_set< DEFAULT_LO, DEFAULT_HI >, matrix_multi< double, DEFAULT_LO, DEFAULT_HI, tuning<>> > . . . . .	109
glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > . . . . .	185
glucat::compare_types< LHS_T, RHS_T > . . . . .	121
glucat::compare_types< T, T > . . . . .	123
glucat::control_t . . . . .	124
glucat::CTAssertion< bool > . . . . .	128
glucat::CTAssertion< true > . . . . .	128
glucat::numeric_traits< Scalar_T >::demoted . . . . .	129
glucat::matrix::eig_genus< Matrix_T > . . . . .	129
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::hash_size_t . . . . .	159
glucat::index_set_hash< LO, HI > . . . . .	184
IndexSet	
PyClical.index_set . . . . .	175
inline	
PyClical.clifford . . . . .	88
PyClical.index_set . . . . .	175

std::logic_error	
glucat::glucat_error . . . . .	157
glucat::error< multivector_t > . . . . .	131
glucat::error< index_set > . . . . .	131
glucat::error< Class_T > . . . . .	131
std::map	
glucat::basis_table< Scalar_T, LO, HI, Matrix_T > . . . . .	85
glucat::gen::generator_table< Matrix_T > . . . . .	152
numeric_limits	
std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > > . . . . .	202
std::numeric_limits< glucat::matrix_multi< Scalar_T, LO, HI, Tune_P > > . . . . .	203
glucat::numeric_traits< Scalar_T > . . . . .	204
obj	
PyClical.clifford . . . . .	88
PyClical.index_set . . . . .	175
pade::pade_log_denom< Scalar_T > . . . . .	215
pade::pade_log_denom< dd_real > . . . . .	216
pade::pade_log_denom< float > . . . . .	217
pade::pade_log_denom< long double > . . . . .	218
pade::pade_log_denom< qd_real > . . . . .	219
pade::pade_log_numer< Scalar_T > . . . . .	220
pade::pade_log_numer< dd_real > . . . . .	221
pade::pade_log_numer< float > . . . . .	221
pade::pade_log_numer< long double > . . . . .	222
pade::pade_log_numer< qd_real > . . . . .	223
pade::pade_sqrt_denom< Scalar_T > . . . . .	224
pade::pade_sqrt_denom< dd_real > . . . . .	225
pade::pade_sqrt_denom< float > . . . . .	226
pade::pade_sqrt_denom< long double > . . . . .	226
pade::pade_sqrt_denom< qd_real > . . . . .	227
pade::pade_sqrt_numer< Scalar_T > . . . . .	228
pade::pade_sqrt_numer< dd_real > . . . . .	229
pade::pade_sqrt_numer< float > . . . . .	230
pade::pade_sqrt_numer< long double > . . . . .	231
pade::pade_sqrt_numer< qd_real > . . . . .	231
std::pair	
glucat::framed_multi< Scalar_T, LO, HI, Tune_P >::var_term . . . . .	245
glucat::numeric_traits< Scalar_T >::promoted . . . . .	232
glucat::random_generator< Scalar_T > . . . . .	233
glucat::index_set< LO, HI >::reference . . . . .	237
glucat::sorted_range< Map_T, Sorted_Map_T > . . . . .	241
glucat::sorted_range< Sorted_Map_T, Sorted_Map_T > . . . . .	243
toClifford	
PyClical.clifford . . . . .	88
toIndexSet	
PyClical.index_set . . . . .	175
std::unordered_map	
glucat::framed_multi< Scalar_T, LO, HI, Tune_P > . . . . .	134
glucat::framed_multi< Scalar_T, LO, HI, Tune_P > . . . . .	134



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">glucat::basis_table&lt; Scalar_T, LO, HI, Matrix_T &gt;</a>	85
Table of basis elements used as a cache by basis_element()	
<a href="#">glucat::bool_to_type&lt; truth_value &gt;</a>	87
Bool to type	
<a href="#">PyClical.clifford</a>	88
<a href="#">glucat::clifford_algebra&lt; Scalar_T, Index_Set_T, Multivector_T &gt;</a>	
Clifford_algebra<> declares the operations of a <a href="#">Clifford</a> algebra	109
<a href="#">glucat::compare_types&lt; LHS_T, RHS_T &gt;</a>	
Type comparison	121
<a href="#">glucat::compare_types&lt; T, T &gt;</a>	123
<a href="#">glucat::control_t</a>	
Parameters to control tests	124
<a href="#">glucat::CTAssertion&lt; bool &gt;</a>	
Compile time assertion	128
<a href="#">glucat::CTAssertion&lt; true &gt;</a>	128
<a href="#">glucat::numeric_traits&lt; Scalar_T &gt;::demoted</a>	
Demoted type for long double	129
<a href="#">glucat::matrix::eig_genus&lt; Matrix_T &gt;</a>	
Structure containing classification of eigenvalues	129
<a href="#">glucat::error&lt; Class_T &gt;</a>	
Specific exception class	131
<a href="#">glucat::framed_multi&lt; Scalar_T, LO, HI, Tune_P &gt;</a>	
A <a href="#">framed_multi&lt;Scalar_T,LO,HI,Tune_P&gt;</a> is a framed approximation to a multivector	134
<a href="#">glucat::gen::generator_table&lt; Matrix_T &gt;</a>	
Table of generators for specific signatures	152
<a href="#">glucat::glucat_error</a>	
Abstract exception class	157
<a href="#">glucat::framed_multi&lt; Scalar_T, LO, HI, Tune_P &gt;::hash_size_t</a>	159
<a href="#">glucat::index_set&lt; LO, HI &gt;</a>	
Index set class based on <a href="#">std::bitset&lt;&gt;</a> in Gnu standard C++ library	161
<a href="#">PyClical.index_set</a>	175
<a href="#">glucat::index_set_hash&lt; LO, HI &gt;</a>	184
<a href="#">glucat::matrix_multi&lt; Scalar_T, LO, HI, Tune_P &gt;</a>	
A <a href="#">matrix_multi&lt;Scalar_T,LO,HI,Tune_P&gt;</a> is a matrix approximation to a multivector	185
<a href="#">std::numeric_limits&lt; glucat::framed_multi&lt; Scalar_T, LO, HI, Tune_P &gt; &gt;</a>	
Numeric limits for <a href="#">framed_multi</a> inherit limits for the corresponding scalar type	202

<a href="#">std::numeric_limits&lt; glucat::matrix_multi&lt; Scalar_T, LO, HI, Tune_P &gt; &gt;</a>	
Numeric limits for matrix_multi inherit limits for the corresponding scalar type . . . . .	203
<a href="#">glucat::numeric_traits&lt; Scalar_T &gt;</a>	
Extra traits which extend numeric limits . . . . .	204
<a href="#">pade::pade_log_denom&lt; Scalar_T &gt;</a>	
Coefficients of denominator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)	
215	
<a href="#">pade::pade_log_denom&lt; dd_real &gt;</a> . . . . .	216
<a href="#">pade::pade_log_denom&lt; float &gt;</a> . . . . .	217
<a href="#">pade::pade_log_denom&lt; long double &gt;</a> . . . . .	218
<a href="#">pade::pade_log_denom&lt; qd_real &gt;</a> . . . . .	219
<a href="#">pade::pade_log_numer&lt; Scalar_T &gt;</a>	
Coefficients of numerator polynomials of Pade approximations produced by Pade1(log(1+x),x,n,n)	
220	
<a href="#">pade::pade_log_numer&lt; dd_real &gt;</a> . . . . .	221
<a href="#">pade::pade_log_numer&lt; float &gt;</a> . . . . .	221
<a href="#">pade::pade_log_numer&lt; long double &gt;</a> . . . . .	222
<a href="#">pade::pade_log_numer&lt; qd_real &gt;</a> . . . . .	223
<a href="#">pade::pade_sqrt_denom&lt; Scalar_T &gt;</a>	
Coefficients of denominator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)	
224	
<a href="#">pade::pade_sqrt_denom&lt; dd_real &gt;</a> . . . . .	225
<a href="#">pade::pade_sqrt_denom&lt; float &gt;</a> . . . . .	226
<a href="#">pade::pade_sqrt_denom&lt; long double &gt;</a> . . . . .	226
<a href="#">pade::pade_sqrt_denom&lt; qd_real &gt;</a> . . . . .	227
<a href="#">pade::pade_sqrt_numer&lt; Scalar_T &gt;</a>	
Coefficients of numerator polynomials of Pade approximations produced by Pade1(sqrt(1+x),x,n,n)	
228	
<a href="#">pade::pade_sqrt_numer&lt; dd_real &gt;</a> . . . . .	229
<a href="#">pade::pade_sqrt_numer&lt; float &gt;</a> . . . . .	230
<a href="#">pade::pade_sqrt_numer&lt; long double &gt;</a> . . . . .	231
<a href="#">pade::pade_sqrt_numer&lt; qd_real &gt;</a> . . . . .	231
<a href="#">glucat::numeric_traits&lt; Scalar_T &gt;::promoted</a>	
Extra traits which extend numeric limits . . . . .	232
<a href="#">glucat::random_generator&lt; Scalar_T &gt;</a>	
Random number generator with single instance per Scalar_T . . . . .	233
<a href="#">glucat::index_set&lt; LO, HI &gt;::reference</a>	
Index set member reference . . . . .	237
<a href="#">glucat::sorted_range&lt; Map_T, Sorted_Map_T &gt;</a>	
Sorted range for use with output . . . . .	241
<a href="#">glucat::sorted_range&lt; Sorted_Map_T, Sorted_Map_T &gt;</a> . . . . .	243
<a href="#">glucat::framed_multi&lt; Scalar_T, LO, HI, Tune_P &gt;::var_term</a>	
Variable term . . . . .	245

## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

glucat/clifford_algebra.h	249
glucat/clifford_algebra_imp.h	266
glucat/errors.h	286
glucat/errors_imp.h	287
glucat/framed_multi.h	289
glucat/framed_multi_imp.h	296
glucat/generation.h	319
glucat/generation_imp.h	321
glucat/global.h	325
glucat/glucat.h	328
glucat/glucat_config.h	330
glucat/glucat_imp.h	335
glucat/index_set.h	337
glucat/index_set_imp.h	341
glucat/long_double.h	354
glucat/matrix.h	356
glucat/matrix_imp.h	359
glucat/matrix_multi.h	368
glucat/matrix_multi_imp.h	374
glucat/portability.h	402
glucat/promotion.h	404
glucat/qd.h	408
glucat/random.h	413
glucat/scalar.h	415
glucat/scalar_imp.h	419
glucat/tuning.h	422
pyclical/glucat.pxd	428
pyclical/PyClical.h	430
pyclical/PyClical.pxd	436
pyclical/PyClical.pyx	436
test/control.h	461
test/driver.h	463
test/timing.h	464
test/try_catch.h	466
test/tuning.h	426
test/undefine.h	467



## 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`  
`<_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 subalgebra.*

- 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

- template<typename Scalar\_T, typename Index\_Set\_T, typename Multivector\_T>  
const Scalar\_T [clifford\\_algebra](#)< Scalar\_T, Index\_Set\_T, Multivector\_T >::default\_truncation = std::numeric\_limits<Scalar\_T>::epsilon()

*Default for truncation.*

- 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.

References [BITS\\_PER\\_SET\\_VALUE](#).

### 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

References [\\_GLUCAT\\_CTAssert\(\)](#), and [DEFAULT\\_HI](#).

### 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](#)

Referenced by [\\_GLUCAT\\_CTAssert\(\)](#).

#### 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 902 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 882 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 843 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 824 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 1007 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 987 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 948 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 929 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 1107 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 1087 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 1051 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 1034 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>
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 1910 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>
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 689 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,  $rev \circ inv = inv \circ rev$ .

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 873 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 850 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 806 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>
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 1344 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>
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>
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 1704 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>
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 1317 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>
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 1305 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 1745 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 2074 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>
auto glucat::fast (
    const Matrix_T & X,
    index_t level) -> Multivector_T [static]
```

Inverse generalized Fast Fourier Transform.

Definition at line 1024 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>
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()

```
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()`

```
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 1795 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 2033 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 798 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 790 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 1957 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 1563 of file `matrix_multi_imp.h`.

References `abs()`, `approx_equal()`, `glucat::matrix::classify_eigenvalues()`, `cr_sqrt()`, `db_sqrt()`, `pade::pade_sqrt_denom< Scalar_T >::exp()`, `glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >::isnan()`, `matrix_sqrt()`, `norm()`, `pade::pade_sqrt_numer< Scalar_T >::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`.

Referenced by `operator<<()`.

### 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`.

Referenced by `operator<<()`.

**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](#).

**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 579 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 560 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 500 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 1712 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 731 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 612 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 1144 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 &
```

References [max\\_pos\(\)](#), [min\\_neg\(\)](#), and [sign\\_of\\_square\(\)](#).

**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 952 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 1204 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 1243 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 963 of file [matrix\\_multi\\_imp.h](#).

**5.2.2.94 operator<sup>^</sup>() [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<sup>^</sup>() [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<sup>^</sup>() [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 541 of file [matrix\\_multi\\_imp.h](#).

**5.2.2.97 operator<sup>^</sup>() [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 756 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 714 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>
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 1242 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>
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 1890 of file [matrix\\_multi\\_imp.h](#).

References [pade::pade\\_log\\_denom< Scalar\\_T >::denom](#), [glucat::clifford\\_algebra< Scalar\\_T, Index\\_Set\\_T, Multivector\\_T >::isnan\(\)](#), [pade::pade\\_log\\_numer< Scalar\\_T >::numer](#), 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 343 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](#).

Referenced by [operator<<\(\)](#).

**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 978 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 955 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 910 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 1722 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 1657 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 682 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 674 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 683 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 598 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 1078 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 1059 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 1015 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](#).

Referenced by [\\_GLUCAT\\_CTAssert\(\)](#).

**5.2.3.2 clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >::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()
```

Default for truncation.

Definition at line 74 of file [clifford\\_algebra\\_imp.h](#).

### 5.2.3.3 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](#).

Referenced by [\\_GLUCAT\\_CTAssert\(\)](#).

### 5.2.3.4 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\(\)](#).

### 5.2.3.5 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\(\)](#).

### 5.2.3.6 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](#).

### 5.2.3.7 Tuning\_Fast\_Basis\_Max\_Count

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

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

### 5.2.3.8 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](#).

#### 5.2.3.9 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](#).

#### 5.2.3.10 Tuning\_Fast\_Div\_Max\_Steps

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

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

#### 5.2.3.11 Tuning\_Fast\_Fast\_Size\_Threshold

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

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

#### 5.2.3.12 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](#).

#### 5.2.3.13 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](#).

#### 5.2.3.14 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](#).

#### 5.2.3.15 Tuning\_Fast\_Mult\_Matrix\_Threshold

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

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

#### 5.2.3.16 Tuning\_Fast\_Products\_Size\_Threshold

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

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

#### 5.2.3.17 Tuning\_Int\_Digits

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

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

#### 5.2.3.18 Tuning\_Max\_Threshold

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

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

#### 5.2.3.19 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](#).

#### 5.2.3.20 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](#).

#### 5.2.3.21 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](#).

#### 5.2.3.22 Tuning\_Naive\_Mult\_Matrix\_Threshold

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

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

#### 5.2.3.23 Tuning\_Slow\_Basis\_Max\_Count

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

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

#### 5.2.3.24 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](#).



#### 5.2.3.25 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](#).

#### 5.2.3.26 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](#).

#### 5.2.3.27 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\(\)](#), [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\\_blaze](#) (const Matrix\_T &val) -> blaze::DynamicMatrix< double, blaze::rowMajor >

*Convert matrix to Blaze 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 492 of file [matrix\\_imp.h](#).

References [eigenvalues\(\)](#), [epsilon](#), [glucat::matrix::eig\\_genus< Matrix\\_T >::m\\_eig\\_case](#), [glucat::matrix::eig\\_genus< Matrix\\_T >::m\\_eig\\_genus< Matrix\\_T >::m\\_safe\\_arg](#), [glucat::numeric\\_traits< Scalar\\_T >::pi\(\)](#), and [glucat::numeric\\_traits< Scalar\\_T](#)

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 464 of file [matrix\\_imp.h](#).

References [to\\_blaze\(\)](#).

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:  $\sum(x(i,j)*y(i,j))/x.nrows()$

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

Definition at line 368 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 270 of file [matrix\\_imp.h](#).

References [glucat::numeric\\_traits< Scalar\\_T >::isInf\(\)](#).

#### 5.4.2.5 `isnan()`

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

Not a Number.

Definition at line 287 of file `matrix_imp.h`.

References `glucat::numeric_traits< Scalar_T >::isNaN()`.

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 78 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 114 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 315 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 253 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 177 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 147 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 390 of file [matrix\\_imp.h](#).

References [glucat::numeric\\_traits< Scalar\\_T >::isNaN\(\)](#), and [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 356 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 223 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 345 of file [matrix\\_imp.h](#).

#### 5.4.2.16 to\_blaze()

```
template<typename Matrix_T>
auto glucat::matrix::to_blaze (
    const Matrix_T & val) -> blaze::DynamicMatrix<double,blaze::rowMajor> [static]
```

Convert matrix to Blaze format.

Definition at line 435 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 411 of file [matrix\\_imp.h](#).

References [glucat::numeric\\_traits< Scalar\\_T >::isNaN\(\)](#), and [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 305 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()

```
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  $\text{Pade1}(\log(1+x), x, n, n)$
- 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  $\text{Pade1}(\log(1+x), x, n, n)$
- 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  $\text{Pade1}(\sqrt{1+x}, x, n, n)$
- 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  $\text{Pade1}(\sqrt{1+x}, x, n, n)$
- struct [pade\\_sqrt\\_numer< dd\\_real >](#)
- struct [pade\\_sqrt\\_numer< float >](#)
- struct [pade\\_sqrt\\_numer< long double >](#)
- struct [pade\\_sqrt\\_numer< qd\\_real >](#)

## Variables

- `template<typename Scalar_T>`  
`const pade\_sqrt\_numer< Scalar_T >::array pade\_sqrt\_numer< Scalar_T >::numer`
- `template<typename Scalar_T>`  
`const pade\_sqrt\_denom< Scalar_T >::array pade\_sqrt\_denom< Scalar_T >::denom`
- `const pade\_sqrt\_numer< float >::array pade\_sqrt\_numer< float >::numer`
- `const pade\_sqrt\_denom< float >::array pade\_sqrt\_denom< float >::denom`
- `const pade\_sqrt\_numer< longdouble >::array pade\_sqrt\_numer< longdouble >::numer`
- `const pade\_sqrt\_denom< longdouble >::array pade\_sqrt\_denom< longdouble >::denom`
- `const pade\_sqrt\_numer< dd_real >::array pade\_sqrt\_numer< dd_real >::numer`
- `const pade\_sqrt\_denom< dd_real >::array pade\_sqrt\_denom< dd_real >::denom`
- `const pade\_sqrt\_numer< qd_real >::array pade\_sqrt\_numer< qd_real >::numer`
- `const pade\_sqrt\_denom< qd_real >::array pade\_sqrt\_denom< qd_real >::denom`
- `template<typename Scalar_T>`  
`const pade\_log\_numer< Scalar_T >::array pade\_log\_numer< Scalar_T >::numer`
- `template<typename Scalar_T>`  
`const pade\_log\_denom< Scalar_T >::array pade\_log\_denom< Scalar_T >::denom`
- `const pade\_log\_numer< float >::array pade\_log\_numer< float >::numer`
- `const pade\_log\_denom< float >::array pade\_log\_denom< float >::denom`
- `const pade\_log\_numer< longdouble >::array pade\_log\_numer< longdouble >::numer`
- `const pade\_log\_denom< longdouble >::array pade\_log\_denom< longdouble >::denom`
- `const pade\_log\_numer< dd_real >::array pade\_log\_numer< dd_real >::numer`
- `const pade\_log\_denom< dd_real >::array pade\_log\_denom< dd_real >::denom`
- `const pade\_log\_numer< qd_real >::array pade\_log\_numer< qd_real >::numer`
- `const pade\_log\_denom< qd_real >::array pade\_log\_denom< qd_real >::denom`

## 5.6.1 Variable Documentation

### 5.6.1.1 `pade_log_denom< dd_real >::denom`

`const pade\_log\_denom<dd_real>::array pade::pade\_log\_denom< dd_real >::denom`

**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 1815 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.2 `pade_log_denom< float >::denom`

`const pade\_log\_denom<float>::array pade::pade\_log\_denom< float >::denom`

**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 1753 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.3 pade\_log\_denom< longdouble >::denom

```
const pade_log_denom<longdouble>::array pade::pade_log_denom< longdouble >::denom
```

**Initial value:**

```
=
{
    1.0L,                17.0L/2.0L,                1088.0L/33.0L,                850.0L/11.0L,
    41650.0L/341.0L,      140777.0L/1023.0L,          1126216.0L/9889.0L,
    63206.0L/899.0L,
    790075.0L/24273.0L,    60775.0L/5394.0L,            38896.0L/13485.0L,
    21658.0L/40455.0L,
    21658.0L/310155.0L,    4165.0L/682341.0L,            680.0L/2047023.0L,
    34.0L/3411705.0L,
    17.0L/129644790.0L,    1.0L/2333606220
}
```

Definition at line 1780 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.4 pade\_log\_denom< qd\_real >::denom

```
const pade_log_denom<qd_real>::array pade::pade_log_denom< qd_real >::denom
```

**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 1862 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.5 pade\_log\_denom< Scalar\_T >::denom

```
template<typename Scalar_T>
const pade_log_denom<Scalar_T>::array pade::pade_log_denom< Scalar_T >::denom
```

**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 1727 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.6 pade\_log\_numer< dd\_real >::number

```
const pade_log_numer<dd_real>::array pade::pade_log_numer< dd_real >::number
```

Initial value:

```
=
{
    dd_real("0"),                dd_real("1"),
    dd_real("10"),               dd_real("22781")/dd_real("492"),
    dd_real("21603")/dd_real("164"), dd_real("5492649")/dd_real("21320"),
    dd_real("978724")/dd_real("2665"), dd_real("4191605")/dd_real("10619"),
    dd_real("12874933")/dd_real("39442"), dd_real("11473457")/dd_real("54612"),
    dd_real("2406734")/dd_real("22755"), dd_real("166770367")/dd_real("4004880"),
    dd_real("30653165")/dd_real("2402928"), dd_real("647746389")/dd_real("215195552"),
    dd_real("25346331")/dd_real("47074027"), dd_real("278270613")/dd_real("3900419380"),
    dd_real("105689791")/dd_real("15601677520"), dd_real("606046475")/dd_real("1379188292768"),
    dd_real("969715")/dd_real("53502994116"), dd_real("11098301")/dd_real("26204577562592"),
    dd_real("118999")/dd_real("26204577562592"), dd_real("18858053")/dd_real("1392249205900512960")
}
```

Definition at line 1795 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.7 pade\_log\_numer< float >::number

```
const pade_log_numer<float>::array pade::pade_log_numer< float >::number
```

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 1741 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.8 pade\_log\_numer< longdouble >::number

```
const pade_log_numer<longdouble>::array pade::pade_log_numer< longdouble >::number
```

Initial value:

```
=
{
    0.0L,                1.0L,                8.0L,
    3835.0L/132.0L,      11363807.0L/122760.0L,          162981.0L/1705.0L,
    8365.0L/132.0L,      9036157.0L/125860.0L,
    18009875.0L/453096.0L, 44211925.0L/2718576.0L,          4149566.0L/849555.0L,
    16973929.0L/16020180.0L,
    172459.0L/1068012.0L, 116317061.0L/7025382936.0L,          19679783.0L/18441630207.0L,
    23763863.0L/614721006900.0L,
    50747.0L/79318839600.0L, 42142223.0L/14295951736466400.0L
}
```

Definition at line 1766 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.9 pade\_log\_numer< qd\_real >::numer

```
const pade_log_numer<qd_real>::array pade::pade_log_numer< qd_real >::numer
```

Initial value:

```
=
{
    qd_real("0"),
    qd_real("16"),
    qd_real("1"),
    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 1836 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.10 pade\_log\_numer< Scalar\_T >::numer

```
template<typename Scalar_T>
const pade_log_numer<Scalar_T>::array pade::pade_log_numer< Scalar_T >::numer
```

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 1710 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.11 pade\_sqrt\_denom< dd\_real >::denom

```
const pade_sqrt_denom<dd_real>::array pade::pade_sqrt_denom< dd_real >::denom
```

Initial value:

```
=
{
    dd_real("1"),
    dd_real("41")/dd_real("4"),
    dd_real("195")/dd_real("4"),
    dd_real("9139")/dd_real("64"),
}
```

```

        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("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 1488 of file [matrix\\_multi\\_imp.h](#).

#### 5.6.1.12 `pade_sqrt_denom< float >::denom`

```
const pade_sqrt_denom<float>::array pade::pade_sqrt_denom< float >::denom
```

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 1425 of file [matrix\\_multi\\_imp.h](#).

#### 5.6.1.13 `pade_sqrt_denom< longdouble >::denom`

```
const pade_sqrt_denom<longdouble>::array pade::pade_sqrt_denom< longdouble >::denom
```

Initial value:

```

=
{
    1.0L,          33.0L/4.0L,      31.0L,      4495.0L/64.0L,
    27405.0L/256.0L,  118755.0L/1024.0L,   94185.0L/1024.0L,  444015.0L/8192.0L,
    1562275.0L/65536.0L,  2042975.0L/262144.0L,   245157.0L/131072.0L,  676039.0L/2097152.0L,
    323323.0L/8388608.0L,  101745.0L/33554432.0L,   4845.0L/33554432.0L,  969.0L/268435456.0L,
    153.0L/4294967296.0L,  1.0L/17179869184.0L
}

```

Definition at line 1452 of file [matrix\\_multi\\_imp.h](#).

#### 5.6.1.14 `pade_sqrt_denom< qd_real >::denom`

```
const pade_sqrt_denom<qd_real>::array pade::pade_sqrt_denom< qd_real >::denom
```

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("65")/qd_real("4"),
    qd_real("39711")/qd_real("64"),
    qd_real("5949147")/qd_real("1024"),
    qd_real("170574723")/qd_real("8192"),
    qd_real("8996462475")/qd_real("262144"),
    qd_real("59826782925")/qd_real("2097152"),
    qd_real("420696483235")/qd_real("33554432"),
    qd_real("797168807855")/qd_real("268435456"),
    qd_real("6499270398159")/qd_real("17179869184"),
}

```

```

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 1535 of file [matrix\\_multi\\_imp.h](#).

#### 5.6.1.15 pade\_sqrt\_denom< Scalar\_T >::denom

```

template<typename Scalar_T>
const pade_sqrt_denom<Scalar_T>::array pade::pade_sqrt_denom< Scalar_T >::denom

```

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 1399 of file [matrix\\_multi\\_imp.h](#).

#### 5.6.1.16 pade\_sqrt\_numer< dd\_real >::numer

```

const pade_sqrt_numer<dd_real>::array pade::pade_sqrt_numer< dd_real >::numer

```

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 1468 of file [matrix\\_multi\\_imp.h](#).

#### 5.6.1.17 pade\_sqrt\_numer< float >::numer

```

const pade_sqrt_numer<float>::array pade::pade_sqrt_numer< float >::numer

```

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 1413 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.18 pade\_sqrt\_numer< longdouble >::numer

```
const pade_sqrt_numer<longdouble>::array pade::pade_sqrt_numer< longdouble >::numer
```

Initial value:

```
=
{
    1.0L,                35.0L/4.0L,                35.0L,                5425.0L/64.0L,
    35525.0L/256.0L,     166257.0L/1024.0L,         143325.0L/1024.0L,         740025.0L/8192.0L,
    2877875.0L/65536.0L,  4206125.0L/262144.0L,         572033.0L/131072.0L,         1820105.0L/2097152.0L,
    1028755.0L/8388608.0L, 395675.0L/33554432.0L,         24225.0L/33554432.0L,         6783.0L/268435456.0L,
    1785.0L/4294967296.0L, 35.0L/17179869184.0L
}
```

Definition at line 1438 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.19 pade\_sqrt\_numer< qd\_real >::numer

```
const pade_sqrt_numer<qd_real>::array pade::pade_sqrt_numer< qd_real >::numer
```

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 1509 of file [matrix\\_multi\\_imp.h](#).

### 5.6.1.20 pade\_sqrt\_numer< Scalar\_T >::numer

```
template<typename Scalar_T>
const pade_sqrt_numer<Scalar_T>::array pade::pade_sqrt_numer< Scalar_T >::numer
```

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 1382 of file [matrix\\_multi\\_imp.h](#).



## 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 2025 of file [PyClical.pyx](#).

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

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

### 5.7.1.2 clifford\_hidden\_doctests()

PyClic1al.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 1316 of file [PyClical.pyx](#).

### 5.7.1.3 e()

```
PyClical.e (
    obj)
```

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

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

Definition at line 1999 of file [PyClical.pyx](#).

### 5.7.1.4 index\_set\_hidden\_doctests()

```
PyClical.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 [PyClicl.pyx](#).

#### 5.7.1.5 istpq()

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

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

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

Definition at line 2012 of file [PyClicl.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 1973 of file [PyClical.pyx](#).

### 5.7.2.3 `fill`

```
PyClical.fill
```

Definition at line 1927 of file [PyClical.pyx](#).

### 5.7.2.4 `i`

```
PyClical.i
```

Definition at line 1654 of file [PyClical.pyx](#).

### 5.7.2.5 `ist`

```
PyClical.ist = index_set
```

Definition at line 1991 of file [PyClical.pyx](#).

### 5.7.2.6 `ixt`

```
PyClical.ixt
```

Definition at line 1927 of file [PyClical.pyx](#).

### 5.7.2.7 `lhs`

```
PyClical.lhs
```

Definition at line 1422 of file [PyClical.pyx](#).

#### 5.7.2.8 nbar3

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

Definition at line 2022 of file [PyClical.pyx](#).

#### 5.7.2.9 ninf3

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

Definition at line 2021 of file [PyClical.pyx](#).

#### 5.7.2.10 None

```
PyClical.None
```

Definition at line 1422 of file [PyClical.pyx](#).

#### 5.7.2.11 obj

```
PyClical.obj
```

Definition at line 1654 of file [PyClical.pyx](#).

#### 5.7.2.12 pi

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

Definition at line 1970 of file [PyClical.pyx](#).

#### 5.7.2.13 rhs

```
PyClical.rhs
```

Definition at line 1422 of file [PyClical.pyx](#).

#### 5.7.2.14 scalar\_epsilon

```
PyClical.scalar_epsilon = epsilon
```

Definition at line 1968 of file [PyClical.pyx](#).

#### 5.7.2.15 tau

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

Definition at line 1971 of file [PyClical.pyx](#).

### 5.7.2.16 threshold

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

Definition at line 1422 of file [PyClical.pyx](#).

### 5.7.2.17 tol

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

Definition at line 1422 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 1159 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]
```

Referenced by [basis\(\)](#), [basis\\_table\(\)](#), and [operator=\(\)](#).

#### 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]
```

References [basis\\_table\(\)](#).

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

Single instance of basis table.

Definition at line 1165 of file [matrix\\_multi\\_imp.h](#).

References [basis\\_table\(\)](#).

#### 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]
```

References [basis\\_table\(\)](#).

### 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 1170 of file [matrix\\_multi\\_imp.h](#).

References [friend\\_for\\_private\\_destructor](#).

Referenced by [friend\\_for\\_private\\_destructor](#).

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	
-------	--

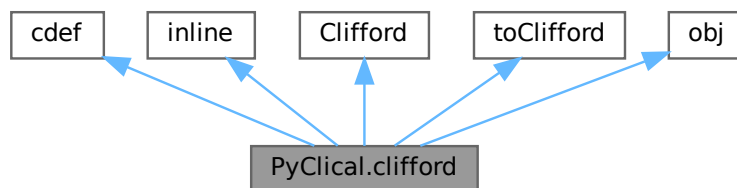
Definition at line 72 of file [global.h](#).

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)
- `__radd__` (rhs, lhs)
- `__iadd__` (self, rhs)
- `__sub__` (lhs, rhs)
- `__rsub__` (rhs, lhs)
- `__isub__` (self, rhs)
- `__mul__` (lhs, rhs)
- `__rmul__` (rhs, lhs)
- `__imul__` (self, rhs)
- `__mod__` (lhs, rhs)
- `__rmod__` (rhs, lhs)
- `__imod__` (self, rhs)
- `__and__` (lhs, rhs)
- `__rand__` (rhs, lhs)
- `__iand__` (self, rhs)
- `__xor__` (lhs, rhs)
- `__rxor__` (rhs, lhs)
- `__ixor__` (self, rhs)
- `__truediv__` (lhs, rhs)
- `__rtruediv__` (rhs, lhs)
- `__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 532 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 872 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 1083 of file [PyClical.pyx](#).

References [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 [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 [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 760 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 896 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 974 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 857 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 820 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 1013 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 789 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 935 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 833 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 798 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 [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 1002 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 1024 of file [PyClical.pyx](#).

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

### 6.3.2.23 `__radd__()`

```
PyClical.clifford.__radd__ (
    rhs,
    lhs)
```

Geometric sum.

```
>>> print(1 + clifford("{2}"))
1+{2}
```

Definition at line 751 of file [PyClical.pyx](#).

### 6.3.2.24 `__rand__()`

```
PyClical.clifford.__rand__ (
    rhs,
    lhs)
```

Inner product.

```
>>> print(2 & clifford("{2}"))
0
```

Definition at line 887 of file [PyClical.pyx](#).

### 6.3.2.25 `__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 1298 of file [PyClical.pyx](#).

References [clifford\\_to\\_repr\(\)](#).

**6.3.2.26** `__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](#).

**6.3.2.27** `__rmod__()`

```
PyClical.clifford.__rmod__ (
    rhs,
    lhs)
```

Contraction.

```
>>> print(2 % clifford("{2}"))
2{2}
```

Definition at line 848 of file [PyClical.pyx](#).

**6.3.2.28** `__rmul__()`

```
PyClical.clifford.__rmul__ (
    rhs,
    lhs)
```

Geometric product.

```
>>> print(2 * clifford("{2}"))
2{2}
```

Definition at line 811 of file [PyClical.pyx](#).

### 6.3.2.29 `__rsub__()`

```
PyClical.clifford.__rsub__ (
    rhs,
    lhs)
```

Geometric difference.

```
>>> print(1 - clifford("{2}"))
1-{2}
```

Definition at line 780 of file [PyClical.pyx](#).

### 6.3.2.30 `__rtruediv__()`

```
PyClical.clifford.__rtruediv__ (
    rhs,
    lhs)
```

Geometric quotient.

```
>>> print(2 / clifford("{2}"))
2{2}
```

Definition at line 965 of file [PyClical.pyx](#).

### 6.3.2.31 `__rxor__()`

```
PyClical.clifford.__rxor__ (
    rhs,
    lhs)
```

Outer product.

```
>>> print(2 ^ clifford("{2}"))
2{2}
```

Definition at line 926 of file [PyClical.pyx](#).

### 6.3.2.32 `__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 1307 of file [PyClical.pyx](#).

References [clifford\\_to\\_str\(\)](#).



**6.3.2.33 `__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 769 of file [PyClical.pyx](#).

**6.3.2.34 `__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 950 of file [PyClical.pyx](#).

**6.3.2.35 `__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 911 of file [PyClical.pyx](#).

**6.3.2.36 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 1238 of file [PyClical.pyx](#).

References [glucat.abs\(\)](#).

**6.3.2.37 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 1201 of file [PyClical.pyx](#).

References [conj\(\)](#), [instance](#), and [PyClical.index\\_set.instance](#).

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

**6.3.2.38 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 1124 of file [PyClical.pyx](#).

References [even\(\)](#), [instance](#), and [PyClical.index\\_set.instance](#).

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

**6.3.2.39 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 1287 of file [PyClical.pyx](#).

References [frame\(\)](#), [instance](#), and [PyClical.index\\_set.instance](#).

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

**6.3.2.40 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 989 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [inv\(\)](#).

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

**6.3.2.41 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 1170 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [involute\(\)](#).

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

#### 6.3.2.42 isinf()

```
PyClical.clifford.isinf (  
    self)
```

Check if a multivector contains any infinite values.

```
>>> clifford().isinf()  
False
```

Definition at line 1269 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [isnan\(\)](#).

#### 6.3.2.43 isnan()

```
PyClical.clifford.isnan (  
    self)
```

Check if a multivector contains any IEEE NaN values.

```
>>> clifford().isnan()  
False
```

Definition at line 1278 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [isinf\(\)](#).

Referenced by [isinf\(\)](#), and [isnan\(\)](#).

#### 6.3.2.44 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 1247 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [max\\_abs\(\)](#).

Referenced by [max\\_abs\(\)](#).

**6.3.2.45 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 1227 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [norm\(\)](#).

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

**6.3.2.46 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 1133 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [odd\(\)](#).

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

**6.3.2.47 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 1067 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [outer\\_pow\(\)](#).

Referenced by [outer\\_pow\(\)](#).

#### 6.3.2.48 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 1043 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [pow\(\)](#).

Referenced by [\\_\\_pow\\_\\_\(\)](#), and [pow\(\)](#).

#### 6.3.2.49 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 1113 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [pure\(\)](#).

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

#### 6.3.2.50 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 1216 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [quad\(\)](#).

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

**6.3.2.51 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](#).

**6.3.2.52 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 1186 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [reverse\(\)](#).

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

**6.3.2.53 scalar()**

```
PyClical.clifford.scalar (
    self)
```

Scalar part.

```
>>> clifford("1+{1}+{1,2}").scalar()
1.0
>>> clifford("{1,2}").scalar()
0.0
```

Definition at line 1102 of file [PyClical.pyx](#).

References [instance](#), [PyClical.index\\_set.instance](#), and [scalar\(\)](#).

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

### 6.3.2.54 truncated()

```
PyClicl.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 1258 of file [PyClicl.pyx](#).

References [instance](#), [PyClicl.index\\_set.instance](#), and [truncated\(\)](#).

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

### 6.3.2.55 vector\_part()

```
PyClicl.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 1142 of file [PyClicl.pyx](#).

References [instance](#), [PyClicl.index\\_set.instance](#), and [vector\\_part\(\)](#).

Referenced by [vector\\_part\(\)](#).

## 6.3.3 Member Data Documentation

### 6.3.3.1 instance

```
PyClicl.clifford.instance = new Clifford((<clifford>other).unwrap())
```

Definition at line 592 of file [PyClicl.pyx](#).

Referenced by [\\_\\_call\\_\\_\(\)](#), [\\_\\_dealloc\\_\\_\(\)](#), [\\_\\_getitem\\_\\_\(\)](#), [\\_\\_neg\\_\\_\(\)](#), [conj\(\)](#), [even\(\)](#), [frame\(\)](#), [inv\(\)](#), [involute\(\)](#), [isinf\(\)](#), [isnan\(\)](#), [max\\_abs\(\)](#), [norm\(\)](#), [odd\(\)](#), [outer\\_pow\(\)](#), [pow\(\)](#), [pure\(\)](#), [quad\(\)](#), [reverse\(\)](#), [scalar\(\)](#), [truncated\(\)](#), and [vector\\_part\(\)](#).

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

- [pyclicl/PyClicl.pyx](#)

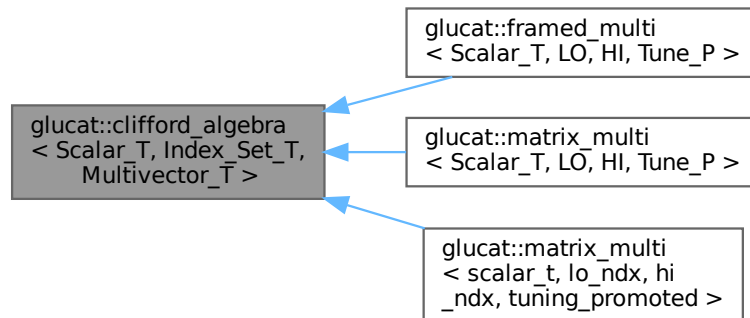


## 6.4 glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T > Class Template Reference

`clifford_algebra<>` declares the operations of a [Clifford](#) algebra

```
#include <clifford_algebra.h>
```

Inheritance diagram for `glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >`:



### 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](#)  
*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.

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

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

#### 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.

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

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

#### 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.

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

Referenced by [frame\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), 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.

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

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

#### 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.

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

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

#### 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}.

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

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

#### 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.

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

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

#### 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.

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

Referenced by [glucat::cascade\\_log\(\)](#), [glucat::exp\(\)](#), [isnan\(\)](#), [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.

References [max\\_abs\(\)](#).

Referenced by [max\\_abs\(\)](#).

**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.

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

Referenced by [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), and [norm\(\)](#).

**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.

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

Referenced by [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::fast\(\)](#), and [odd\(\)](#).

**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.

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

**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.

References [outer\\_pow\(\)](#).

Referenced by [outer\\_pow\(\)](#).

#### 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

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

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

#### 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.

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

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

#### 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)

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

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

#### 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}.

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

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

#### 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.

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

Referenced by [glucat::exp\(\)](#), [glucat::matrix\\_log\(\)](#), [glucat::matrix\\_sqrt\(\)](#), and [scalar\(\)](#).

#### 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.

References [default\\_truncation](#), and [truncated\(\)](#).

Referenced by [glucat::matrix\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::matrix\\_multi\(\)](#), [glucat::operator<<\(\)](#), and [truncated\(\)](#).

**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\(\)](#)

References [vector\\_part\(\)](#).

Referenced by [vector\\_part\(\)](#), and [vector\\_part\(\)](#).

**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*.

References [vector\\_part\(\)](#).

**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.

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

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

**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::ofstream & ofile,
    const std::string & msg = "") const [pure virtual]
```

Write formatted multivector to file.

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

## 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 [static]
```

Default for truncation.

Definition at line 59 of file [clifford\\_algebra.h](#).

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

### 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 }
- enum

### 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

```
anonymous enum
```

Definition at line 57 of file [global.h](#).

#### 6.6.2.2 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\(\)](#).

Referenced by [control\(\)](#), [control\\_t\(\)](#), and [operator=\(\)](#).

### 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]
```

References [control\\_t\(\)](#).

## 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 (
    int(intfn 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()`

```
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](#).

References [control\\_t\(\)](#).

### 6.7.3.5 `operator=()`

```
control\_t & glucat::control_t::operator= (
    const control\_t & ) [private], [delete]
```

References [control\\_t\(\)](#).

### 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()

```
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](#).

References [friend\\_for\\_private\\_destructor](#).

Referenced by [friend\\_for\\_private\\_destructor](#).

## 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
- 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 76 of file [promotion.h](#).

#### 6.10.2 Member Typedef Documentation

##### 6.10.2.1 `type` [1/2]

```
template<typename Scalar_T>
using glucat::numeric_traits< Scalar_T >::demoted::type = float
```

Definition at line 78 of file [promotion.h](#).

##### 6.10.2.2 `type` [2/2]

```
template<typename Scalar_T>
using glucat::numeric_traits< Scalar_T >::demoted::type = float
```

Definition at line 148 of file [scalar.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:

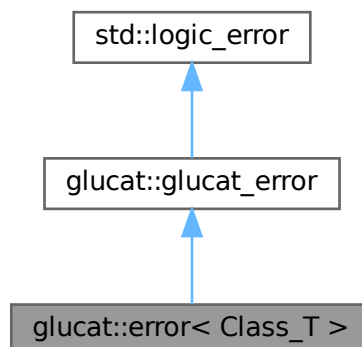
- [glucat/matrix.h](#)

## 6.12 glucat::error< Class\_T > Class Template Reference

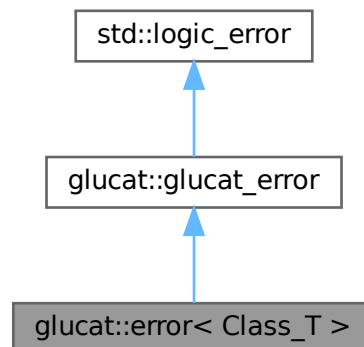
Specific exception class.

```
#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](#).

References [classname\(\)](#), and [glucat::glucat\\_error::glucat\\_error\(\)](#).

### 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](#).

References [glucat::glucat\\_error::glucat\\_error\(\)](#).

## 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](#).

References [glucat::glucat\\_error::name](#).

Referenced by [error\(\)](#), and [print\\_error\\_msg\(\)](#).

### 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](#).

Referenced by [print\\_error\\_msg\(\)](#).

### 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](#).

References [classname\(\)](#), and [heading\(\)](#).

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

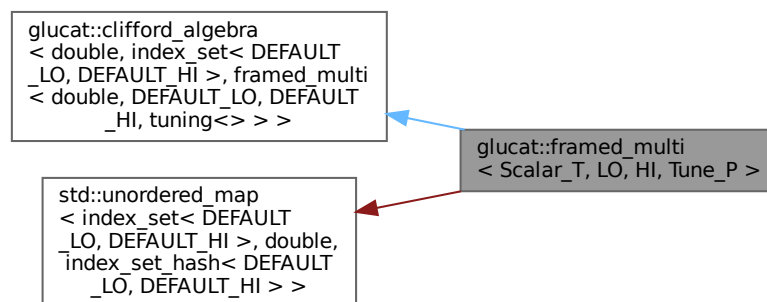
- [glucat/errors.h](#)
- [glucat/errors\\_imp.h](#)

## 6.13 glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P > Class Template Reference

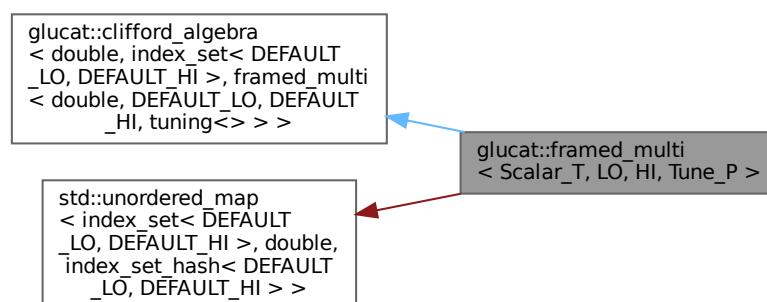
A [framed\\_multi<Scalar\\_T,LO,HI,Tune\\_P>](#) is a framed approximation to a multivector.

```
#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](#)< [double](#), [index\\_set](#)< [DEFAULT\\_LO](#), [DEFAULT\\_HI](#) >, [framed\\_multi](#)< [double](#), [DE](#)

- 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`< `double`, `index_set`< `DEFAULT_LO`, `DEFAULT_HI` >, `framed_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 `multivector_t` &rhs) -> `multivector_t` &=0
  - Geometric sum.
- virtual auto `operator-=` (const `multivector_t` &rhs) -> `multivector_t` &=0
  - Geometric difference.
- 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
  - 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
  - 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 `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.*

### 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**< **double**, **index\_set**< **DEFAULT\_LO**, **DEFAULT\_HI** >, **framed\_multi**< **double**, **DE**

- 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< double, index\_set< DEFAULT\_LO, DEFAULT\_HI >, framed\_multi< double, DE**

- static const [index\\_t v\\_lo](#)
- static const [index\\_t v\\_hi](#)
- static const double [default\\_truncation](#)

*Default for truncation.*

## 6.13.1 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::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 126 of file [framed\\_multi.h](#).

## 6.13.2 Member Typedef Documentation

### 6.13.2.1 const\_iterator

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI,
typename Tune_P = tuning<>>
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 = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI,
typename Tune_P = tuning<>>
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 = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI,
typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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](#).

References [\\_GLUCAT\\_HASH\\_N](#).

### 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](#).

References [\\_GLUCAT\\_HASH\\_N](#).

**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](#).

References [\\_GLUCAT\\_HASH\\_N](#), [framed\\_multi](#), and [glucat::numeric\\_traits< Scalar\\_T >::to\\_scalar\\_t\(\)](#).

**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\\_HASH\\_N](#), [glucat::clifford\\_algebra< Scalar\\_T, Index\\_Set\\_T, Multivector\\_T >::frame\(\)](#), [framed\\_multi](#), and [glucat::numeric\\_traits< Scalar\\_T >::to\\_scalar\\_t\(\)](#).

**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\\_HASH\\_N](#), and [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](#).

References [\\_GLUCAT\\_HASH\\_N](#).

**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](#).

References [\\_GLUCAT\\_HASH\\_N](#).

**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](#).

References [\\_GLUCAT\\_HASH\\_N](#).

**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](#).

References [\\_GLUCAT\\_HASH\\_N](#).

**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\\_HASH\\_N](#), [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](#).

References [\\_GLUCAT\\_HASH\\_N](#).

**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](#).

References [\\_GLUCAT\\_HASH\\_N](#).

**6.13.3.14 framed\_multi()** [13/15]

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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](#).

**6.13.3.15 framed\_multi()** [14/15]

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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](#).

### 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\\_N](#), [\\_GLUCAT\\_HASH\\_SIZE\\_T](#), [glucat::matrix\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::basis\\_element\(\)](#), [fast\\_framed\\_multi\(\)](#), [glucat::clifford\\_algebra< Scalar\\_T, Index\\_Set\\_T, Multivector\\_T >::frame\(\)](#), [glucat::matrix::inner\(\)](#), [glucat::matrix\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::m\\_matrix](#), [matrix\\_multi](#), [glucat::matrix::nnz\(\)](#), [glucat::clifford\\_algebra< Scalar\\_T, Index\\_Set\\_T, Multivector\\_T >::operator\\*](#), [glucat::clifford\\_algebra< double, index\\_set< DEFAULT\\_LO, DEFAULT\\_HI >, framed\\_multi< double, DEFAULT\\_LO, DEFAULT\\_HI >>](#) and [glucat::clifford\\_algebra< double, index\\_set< DEFAULT\\_LO, DEFAULT\\_HI >, framed\\_multi< double, DEFAULT\\_LO, DEFAULT\\_HI >>](#).

## 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 1464 of file [framed\\_multi\\_imp.h](#).

References [glucat::clifford\\_algebra< double, index\\_set< DEFAULT\\_LO, DEFAULT\\_HI >, framed\\_multi< double, DEFAULT\\_LO, DEFAULT\\_HI >>](#).

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 1506 of file [framed\\_multi\\_imp.h](#).

References [glucat::clifford\\_algebra< double, index\\_set< DEFAULT\\_LO, DEFAULT\\_HI >, framed\\_multi< double, DEFAULT\\_LO, DEFAULT\\_HI >>](#).

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 1548 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 1581 of file framed\_multi\_imp.h.

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

#### 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 1597 of file framed\_multi\_imp.h.

References [divide\(\)](#), [glucat::matrix::kron\(\)](#), [glucat::clifford\\_algebra< double, index\\_set< DEFAULT\\_LO, DEFAULT\\_HI >, framed\\_multiglucat::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 1695 of file [framed\\_multi\\_imp.h](#).

Referenced by [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 1663 of file [framed\\_multi\\_imp.h](#).

References [fold\(\)](#), [matrix\\_multi](#), [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 1429 of file [framed\\_multi\\_imp.h](#).

Referenced by [fast\\_matrix\\_multi\(\)](#).

#### 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 1351 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 1054 of file [framed\\_multi\\_imp.h](#).

References [framed\\_multi](#).

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 1446 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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
auto exp (
    const multivector_t & val) -> const multivector_t [friend]
```

### 6.13.5.2 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 143 of file [framed\\_multi.h](#).

Referenced by [framed\\_multi\(\)](#), [framed\\_multi\(\)](#), [framed\\_multi\(\)](#), [framed\\_multi\(\)](#), and [random\(\)](#).

### 6.13.5.3 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 141 of file [framed\\_multi.h](#).

Referenced by [fast\\_matrix\\_multi\(\)](#), and [framed\\_multi\(\)](#).

### 6.13.5.4 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 multivector_t & lhs,
    const multivector_t & rhs) -> const multivector_t [friend]
```

### 6.13.5.5 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 multivector_t & lhs,
    const multivector_t & rhs) -> const multivector_t [friend]
```

### 6.13.5.6 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 multivector_t & lhs,
    const multivector_t & rhs) -> const multivector_t [friend]
```

#### 6.13.5.7 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 multivector_t & lhs,
    const multivector_t & rhs) -> const multivector_t [friend]
```

#### 6.13.5.8 operator<< [1/2]

```
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]
```

#### 6.13.5.9 operator<< [2/2]

```
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 term_t & term) -> std::ostream & [friend]
```

#### 6.13.5.10 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]
```

#### 6.13.5.11 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 multivector_t & lhs,
    const multivector_t & rhs) -> const multivector_t [friend]
```

#### 6.13.5.12 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 multivector_t & lhs,
    const multivector_t & rhs) -> const multivector_t [friend]
```

### 6.13.5.13 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 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/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]
```

Referenced by [generator\(\)](#), [generator\\_table\(\)](#), and [operator=\(\)](#).

### 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]
```

References [generator\\_table\(\)](#).

## 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\(\)](#).

Referenced by [gen\\_vector\(\)](#).

### 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\(\)](#).

Referenced by [gen\\_vector\(\)](#).

### 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\(\)](#).

Referenced by [gen\\_vector\(\)](#).

### 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\(\)](#).

Referenced by [gen\\_vector\(\)](#).

### 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 [gen\\_from\\_pm1\\_qm1\(\)](#), [gen\\_from\\_pm4\\_qp4\(\)](#), [gen\\_from\\_pp4\\_qm4\(\)](#), [gen\\_from\\_qp1\\_pm1\(\)](#), [gen\\_vector\(\)](#), [glucat::pos\\_mod\(\)](#), and [glucat::matrix::unit\(\)](#).

Referenced by [gen\\_vector\(\)](#), and [operator>\(\)\(\)](#).

### 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](#).

References [generator\\_table\(\)](#).

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 [gen\\_vector\(\)](#), [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]
```

References [generator\\_table\(\)](#).

### 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](#).

References [friend\\_for\\_private\\_destructor](#).

Referenced by [friend\\_for\\_private\\_destructor](#).

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

- [glucat/generation.h](#)
- [glucat/generation\\_imp.h](#)

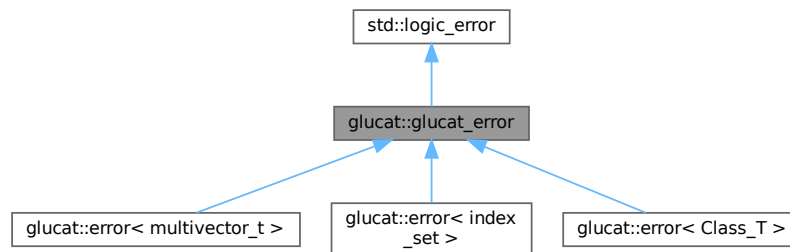


## 6.15 glucat::glucat\_error Class Reference

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](#).

References [name](#).

Referenced by [glucat::error< Class\\_T >::error\(\)](#), and [glucat::error< Class\\_T >::error\(\)](#).

#### 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 >](#), [glucat::error< index\\_set >](#), and [glucat::error< multivector\\_t >](#).

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

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

#### 6.15.3.2 `heading()`

```
virtual auto glucat::glucat_error::heading () const -> const std::string [pure virtual],
[noexcept]
```

Implemented in [glucat::error< Class\\_T >](#), [glucat::error< index\\_set >](#), and [glucat::error< multivector\\_t >](#).

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

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

### 6.15.3.3 `print_error_msg()`

```
virtual void glucat::glucat_error::print_error_msg () const [pure virtual]
```

Implemented in `glucat::error< Class_T >`, `glucat::error< index_set >`, and `glucat::error< multivector_t >`.

References `print_error_msg()`.

Referenced by `print_error_msg()`.

## 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`.

Referenced by `glucat::error< Class_T >::classname()`, and `glucat_error()`.

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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI,
typename Tune_P = tuning<>>
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 = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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](#).

References [n](#).

## 6.16.3 Member Function Documentation

### 6.16.3.1 operator()

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 [n](#).

## 6.16.4 Member Data Documentation

### 6.16.4.1 n

```
template<typename Scalar_T = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 [hash\\_size\\_t\(\)](#), and [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.

Referenced by [flip\(\)](#), [fold\(\)](#), and [fold\(\)](#).

#### 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](#).

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

#### 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 [count\(\)](#), [fold\(\)](#), [min\(\)](#), and [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_ULONG) && (HI< _GLUCAT_↵
    BITS_PER_ULONG) && (HI-LO<=_GLUCAT_BITS_PER_ULONG) ) [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 [count\\_neg\(\)](#), [count\\_pos\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [index\\_set\(\)](#), [is\\_contiguous\(\)](#), [glucat::matrix\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::matrix\\_multi\(\)](#), [operator<\(\)](#), and [sign\\_of\\_square\(\)](#).

#### 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 [count\(\)](#).

Referenced by [sign\\_of\\_square\(\)](#).

#### 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 [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](#).

References [index\\_set\(\)](#).

#### 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](#).

References [fold\(\)](#), and [index\\_set\(\)](#).

Referenced by [fold\(\)](#), [index\\_set\(\)](#), and [value\\_of\\_fold\(\)](#).

#### 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](#).

References [index\\_set\(\)](#), and [test\(\)](#).

#### 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](#).

References [count\(\)](#), [max\(\)](#), and [min\(\)](#).

#### 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](#).

Referenced by [operator<\(\)](#).

**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](#).

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

Referenced by [PyClical.index\\_set::\\_\\_iter\\_\\_\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [is\\_contiguous\(\)](#), [glucat::matrix\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::matrix\\_multi\(\)](#), and [unfold\(\)](#).

**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](#).

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

Referenced by [PyClical.index\\_set::\\_\\_iter\\_\\_\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [index\\_set\(\)](#), [is\\_contiguous\(\)](#), [glucat::matrix\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::matrix\\_multi\(\)](#), and [unfold\(\)](#).

**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](#).

References [count\(\)](#), and [lex\\_less\\_than\(\)](#).

**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](#).

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

**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](#).

Referenced by [index\\_set\(\)](#).

**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](#).

References [count\(\)](#), and [count\\_neg\(\)](#).

**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](#).

Referenced by [fold\(\)](#), [max\(\)](#), [min\(\)](#), [operator\[\]\(\)](#), and [unfold\(\)](#).



### 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](#).

References [max\(\)](#), [min\(\)](#), and [test\(\)](#).

Referenced by [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](#).

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

## 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())

## 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 38 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 [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 [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 [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 [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< lo\\_ndx, hi\\_ndx >.max\(\)](#), [max\(\)](#), [glucat::index\\_set< LO, HI >.min\(\)](#), [glucat::index\\_set< lo\\_ndx, hi\\_ndx >.min\(\)](#), and [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 [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 [count\(\)](#), and [instance](#).

Referenced by [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 [count\\_neg\(\)](#), and [instance](#).

Referenced by [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 [count\\_pos\(\)](#), and [instance](#).

Referenced by [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 [hash\\_fn\(\)](#), and [instance](#).

Referenced by [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 [instance](#), and [max\(\)](#).

Referenced by [\\_\\_iter\\_\\_\(\)](#), and [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 [instance](#), and [min\(\)](#).

Referenced by [\\_\\_iter\\_\\_\(\)](#), and [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 [instance](#), and [sign\\_of\\_mult\(\)](#).

Referenced by [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 [instance](#), and [sign\\_of\\_square\(\)](#).

Referenced by [sign\\_of\\_square\(\)](#).

### 6.18.3 Member Data Documentation

#### 6.18.3.1 instance

`PyClical.index_set.instance = new IndexSet((<index_set>other).unwrap())`

Definition at line 95 of file [PyClical.pyx](#).

Referenced by [PyClical.clifford.\\_\\_call\\_\\_\(\)](#), [\\_\\_contains\\_\\_\(\)](#), [PyClical.clifford.\\_\\_dealloc\\_\\_\(\)](#), [\\_\\_dealloc\\_\\_\(\)](#), [PyClical.clifford.\\_\\_getitem\\_\\_\(\)](#), [\\_\\_getitem\\_\\_\(\)](#), [\\_\\_invert\\_\\_\(\)](#), [PyClical.clifford.\\_\\_neg\\_\\_\(\)](#), [\\_\\_setitem\\_\\_\(\)](#), [PyClical.clifford.conj\(\)](#), [count\(\)](#), [count\\_neg\(\)](#), [count\\_pos\(\)](#), [PyClical.clifford.even\(\)](#), [PyClical.clifford.frame\(\)](#), [hash\\_fn\(\)](#), [PyClical.clifford.inv\(\)](#), [PyClical.clifford.involute\(\)](#), [PyClical.clifford.isinf\(\)](#), [PyClical.clifford.isnan\(\)](#), [max\(\)](#), [PyClical.clifford.max\\_abs\(\)](#), [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\(\)](#), [sign\\_of\\_mult\(\)](#), [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:

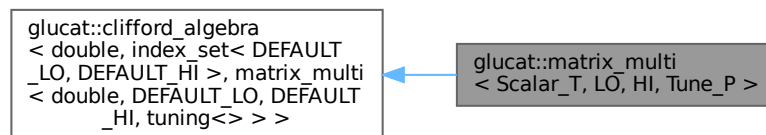
- [glucat/framed\\_multi.h](#)

## 6.20 glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P > Class Template Reference

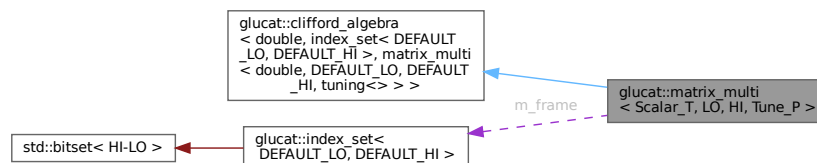
A [matrix\\_multi<Scalar\\_T,LO,HI,Tune\\_P>](#) is a matrix approximation to a multivector.

```
#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 `multivector_t` &rhs) -> `multivector_t` &=0  
Geometric sum.
- virtual auto `operator-=` (const `multivector_t` &rhs) -> `multivector_t` &=0  
Geometric difference.
- 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  
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  
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 [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.*

### 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 [m\\_frame](#), and [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 [m\\_frame](#), [m\\_matrix](#), [matrix\\_multi](#), and [glucat::numeric\\_traits< Scalar\\_T >::to\\_scalar\\_t\(\)](#).

**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\(\)](#), [m\\_frame](#), [m\\_matrix](#), [matrix\\_multi](#), 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 [m\\_frame](#), and [m\\_matrix](#).

**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\(\)](#), [m\\_frame](#), and [m\\_matrix](#).

**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\(\)](#), [m\\_frame](#), and [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\(\)](#), [m\\_frame](#), and [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\(\)](#), [m\\_frame](#), [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](#).

**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](#).

**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::clifford\\_algebra< double, index\\_set< DEFAULT\\_LO, DEFAULT\\_HI >, matrix\\_multi< double framed\\_multi, m\\_frame, and m\\_matrix](#).

**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 276 of file [matrix\\_multi\\_imp.h](#).

References [fast\\_matrix\\_multi\(\)](#), [glucat::folded\\_dim\(\)](#), [framed\\_multi](#), [m\\_frame](#), [m\\_matrix](#), and [glucat::clifford\\_algebra< Scalar\\_T, Index](#)

**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 301 of file [matrix\\_multi\\_imp.h](#).

References [m\\_frame](#), [m\\_matrix](#), and [glucat::numeric\\_traits< Scalar\\_T >::to\\_scalar\\_t\(\)](#).

**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 320 of file [matrix\\_multi\\_imp.h](#).

References [m\\_frame](#), and [m\\_matrix](#).



## 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 1183 of file [matrix\\_multi\\_imp.h](#).

References [glucat::gen::generator\\_table< Matrix\\_T >::generator\(\)](#), [m\\_frame](#), [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 1106 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\(\)](#), [framed\\_multi](#), [m\\_frame](#), [m\\_matrix](#), [glucat::gen::offset\\_to\\_super](#), [glucat::pos\\_mod\(\)](#), and [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::unfold\(\)](#).

Referenced by [fast\\_matrix\\_multi\(\)](#).

### 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 1093 of file [matrix\\_multi\\_imp.h](#).

References [fast\\_framed\\_multi\(\)](#), [fast\\_matrix\\_multi\(\)](#), and [m\\_frame](#).

Referenced by [fast\\_matrix\\_multi\(\)](#), and [matrix\\_multi\(\)](#).

#### 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 [414](#) 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 [328](#) of file [matrix\\_multi\\_imp.h](#).

References [m\\_frame](#), and [m\\_matrix](#).

#### 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 [923](#) 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](#).

Referenced by [fast\\_framed\\_multi\(\)](#), [matrix\\_multi\(\)](#), and [matrix\\_multi\(\)](#).

### 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 [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), and [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 [basis\\_element\(\)](#), [fast\\_framed\\_multi\(\)](#), [fast\\_matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), and [operator=\(\)](#).

### 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 [fast\\_framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), [matrix\\_multi\(\)](#), and [operator=\(\)](#).

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

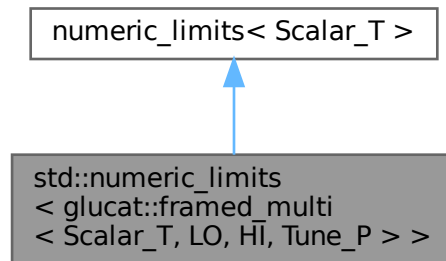
- [glucat/framed\\_multi.h](#)
- [glucat/matrix\\_multi.h](#)
- [glucat/matrix\\_multi\\_imp.h](#)

## 6.21 `std::numeric_limits< glucat::framed_multi< Scalar_T, LO, HI, Tune_P > >` Struct Template Reference

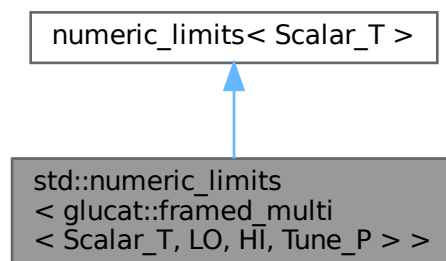
Numeric limits for `framed_multi` inherit limits for the corresponding scalar type.

```
#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:

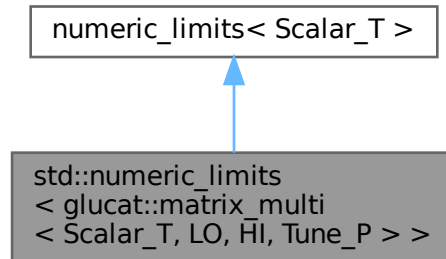
- [glucat/framed\\_multi.h](#)

## 6.22 std::numeric\_limits< glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P > > Struct Template Reference

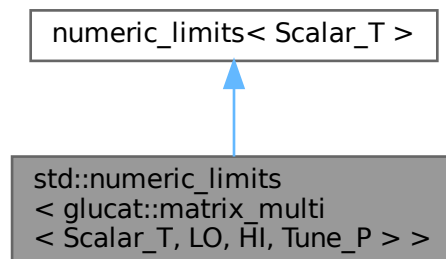
Numeric limits for matrix\_multi inherit limits for the corresponding scalar type.

```
#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>
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>
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>
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>
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>
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>
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>
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>
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>
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>
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>
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 [isInf\(\)](#).

### 6.23.2.12 isInf() [2/3]

```
template<typename Scalar_T>
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 [isInf\(\)](#), [glucat::matrix::isinf\(\)](#), and [isNaN\\_or\\_isInf\(\)](#).

### 6.23.2.13 isInf() [3/3]

```
template<typename Scalar_T>
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>
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 [isNaN\(\)](#).

### 6.23.2.15 isNaN() [2/3]

```
template<typename Scalar_T>
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 [isNaN\(\)](#), [glucat::matrix::isnan\(\)](#), [isNaN\\_or\\_isInf\(\)](#), [glucat::matrix::norm\\_frob2\(\)](#), and [glucat::matrix::trace\(\)](#).

### 6.23.2.16 isNaN() [3/3]

```
template<typename Scalar_T>
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>
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 [isInf\(\)](#), and [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::\\_ln2](#).

### 6.23.2.19 ln\_2() [2/2]

```
template<typename Scalar_T>
auto glucat::numeric_traits< Scalar_T >::ln_2 () -> Scalar_T    [inline], [static]
```

log(2)

Definition at line 196 of file [scalar.h](#).

Referenced by [log2\(\)](#).

### 6.23.2.20 log()

```
template<typename Scalar_T>
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 [log2\(\)](#).

### 6.23.2.21 log2()

```
template<typename Scalar_T>
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 [ln\\_2\(\)](#), and [log\(\)](#).

Referenced by [glucat::log2\(\)](#).

### 6.23.2.22 NaN()

```
template<typename Scalar_T>
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>
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>
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>
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>
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>
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>
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>
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>
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>
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::matrix::classify\\_eigenvalues\(\)](#), [glucat::operator<<\(\)](#), [PyFloat\\_FromDouble\(\)](#), [to\\_scalar\\_t\(\)](#), and [to\\_scalar\\_t\(\)](#).

**6.23.2.33 to\_int()**

```
template<typename Scalar_T>
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 [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 [to\\_double\(\)](#).

**6.23.2.40** `to_scalar_t()` [7/9]

```
template<typename Scalar_T>
template<typename Other_Scalar_T>
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::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_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::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 1721 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 1723 of file [matrix\\_multi\\_imp.h](#).

## 6.24.3 Member Data Documentation

### 6.24.3.1 denom

```
template<typename Scalar_T>
const array pade::pade_log_denom< Scalar_T >::denom [static]
```

Definition at line 1724 of file [matrix\\_multi\\_imp.h](#).

Referenced by [glucat::pade\\_log\(\)](#).

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 1810 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 1812 of file [matrix\\_multi\\_imp.h](#).

## 6.25.3 Member Data Documentation

### 6.25.3.1 `denom`

```
const array pade::pade\_log\_denom< dd\_real >::denom [static]
```

Definition at line 1813 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 1748 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 1750 of file [matrix\\_multi\\_imp.h](#).

## 6.26.3 Member Data Documentation

### 6.26.3.1 denom

```
const array pade::pade\_log\_denom< float >::denom [static]
```

Definition at line 1751 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 1775 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 1777 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 1778 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 1857 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 1859 of file [matrix\\_multi\\_imp.h](#).

## 6.28.3 Member Data Documentation

### 6.28.3.1 denom

```
const array pade::pade_log_denom< qd_real >::denom [static]
```

Definition at line 1860 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 1704 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 1706 of file [matrix\\_multi\\_imp.h](#).

### 6.29.3 Member Data Documentation

#### 6.29.3.1 numer

```
template<typename Scalar_T>
const array pade::pade\_log\_numer< Scalar_T >::numer [static]
```

Definition at line 1707 of file [matrix\\_multi\\_imp.h](#).

Referenced by [glucat::pade\\_log\(\)](#).

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 1790 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 1792 of file `matrix_multi_imp.h`.

### 6.30.3 Member Data Documentation

#### 6.30.3.1 `numer`

```
const array pade::pade_log_numer< dd_real >::numer [static]
```

Definition at line 1793 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](#) [number](#)

## 6.31.1 Detailed Description

Definition at line 1736 of file [matrix\\_multi\\_imp.h](#).

## 6.31.2 Member Typedef Documentation

### 6.31.2.1 array

```
using pade::pade\_log\_number< float >::array = std::array<float, 10>
```

Definition at line 1738 of file [matrix\\_multi\\_imp.h](#).

## 6.31.3 Member Data Documentation

### 6.31.3.1 number

```
const array pade::pade\_log\_number< float >::number [static]
```

Definition at line 1739 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\\_number](#)< long double > Struct Reference

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

### Public Types

- using [array](#) = std::array<long double, 18>

### Static Public Attributes

- static const [array](#) [number](#)

## 6.32.1 Detailed Description

Definition at line 1761 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 1763 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 1764 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 1831 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 1833 of file [matrix\\_multi\\_imp.h](#).

### 6.33.3 Member Data Documentation

#### 6.33.3.1 numer

```
const array pade::pade_log_numer< qd_real >::numer [static]
```

Definition at line 1834 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 1393 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 1395 of file [matrix\\_multi\\_imp.h](#).

### 6.34.3 Member Data Documentation

#### 6.34.3.1 `denom`

```
template<typename Scalar_T>
const array pade::pade_sqrt_denom< Scalar_T >::denom [static]
```

Definition at line 1396 of file `matrix_multi_imp.h`.

Referenced by `glucat::matrix_sqrt()`.

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 1483 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 1485 of file `matrix_multi_imp.h`.

### 6.35.3 Member Data Documentation

#### 6.35.3.1 `denom`

```
const array pade::pade_sqrt_denom< dd_real >::denom [static]
```

Definition at line 1486 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 1420 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 1422 of file `matrix_multi_imp.h`.

### 6.36.3 Member Data Documentation

#### 6.36.3.1 `denom`

```
const array pade::pade_sqrt_denom< float >::denom [static]
```

Definition at line 1423 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 1447 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 1449 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 1450 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 1530 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 1532 of file [matrix\\_multi\\_imp.h](#).

## 6.38.3 Member Data Documentation

### 6.38.3.1 denom

```
const array pade::pade_sqrt_denom< qd_real >::denom [static]
```

Definition at line 1533 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 numer](#)

### 6.39.1 Detailed Description

```
template<typename Scalar_T>
struct pade::pade_sqrt_numer< Scalar_T >
```

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

Definition at line 1376 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_numer< Scalar_T >::array = std::array<Scalar_T, 14>
```

Definition at line 1378 of file `matrix_multi_imp.h`.

## 6.39.3 Member Data Documentation

### 6.39.3.1 numer

```
template<typename Scalar_T>
const array pade::pade_sqrt_numer< Scalar_T >::numer [static]
```

Definition at line 1379 of file `matrix_multi_imp.h`.

Referenced by `glucat::matrix_sqrt()`.

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 1463 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 1465 of file `matrix_multi_imp.h`.

### 6.40.3 Member Data Documentation

#### 6.40.3.1 numer

```
const array pade::pade_sqrt_numer< dd_real >::numer [static]
```

Definition at line 1466 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 1408 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 1410 of file [matrix\\_multi\\_imp.h](#).

### 6.41.3 Member Data Documentation

#### 6.41.3.1 numer

```
const array pade::pade_sqrt_numer< float >::numer [static]
```

Definition at line 1411 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 1433 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 1435 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 1436 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](#) [number](#)

## 6.43.1 Detailed Description

Definition at line [1504](#) of file [matrix\\_multi\\_imp.h](#).

## 6.43.2 Member Typedef Documentation

### 6.43.2.1 array

```
using pade::pade\_sqrt\_numer< qd\_real >::array = std::array<qd\_real, 34>
```

Definition at line [1506](#) of file [matrix\\_multi\\_imp.h](#).

## 6.43.3 Member Data Documentation

### 6.43.3.1 numer

```
const array pade::pade\_sqrt\_numer< qd\_real >::numer [static]
```

Definition at line [1507](#) 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
- using [type](#) = 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 70 of file [promotion.h](#).

### 6.44.2 Member Typedef Documentation

#### 6.44.2.1 `type` [1/3]

```
template<typename Scalar_T>
using glucat::numeric_traits< Scalar_T >::promoted::type = double
```

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

#### 6.44.2.2 `type` [2/3]

```
template<typename Scalar_T>
using glucat::numeric_traits< Scalar_T >::promoted::type = long double
```

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

#### 6.44.2.3 `type` [3/3]

```
template<typename Scalar_T>
using glucat::numeric_traits< Scalar_T >::promoted::type = double
```

Definition at line 145 of file [scalar.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]
```

References [random\\_generator\(\)](#).

Referenced by [generator\(\)](#), [operator=\(\)](#), and [random\\_generator\(\)](#).

### 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 [normal\\_dist](#), [seed](#), [uint\\_gen](#), and [uniform\\_dist](#).

### 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>
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](#).

References [random\\_generator\(\)](#).

### 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 [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]
```

References [random\\_generator\(\)](#).

#### 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 [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](#).

References [friend\\_for\\_private\\_destructor](#).

Referenced by [friend\\_for\\_private\\_destructor](#).

### 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 [normal\(\)](#), and [random\\_generator\(\)](#).

#### 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 [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](#).

Referenced by [random\\_generator\(\)](#).



## 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 [random\\_generator\(\)](#), and [uniform\(\)](#).

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

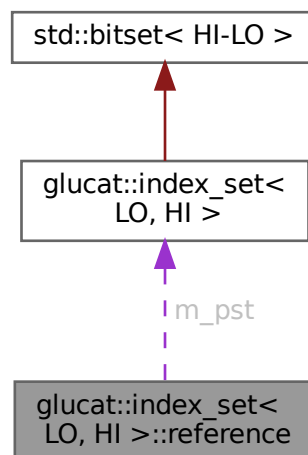
- [glucat/random.h](#)

## 6.46 glucat::index\_set&lt; LO, HI &gt;::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.

Referenced by `flip()`, `operator=()`, `operator=()`, `operator==()`, and `~reference()`.

### 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`.

References `m_idx`, and `m_pst`.

### 6.46.2.3 ~reference()

```
template<const index_t LO, const index_t HI>
glucat::index_set< LO, HI >::reference::~reference () [default]
```

References [flip\(\)](#), and [reference\(\)](#).

## 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](#).

References [m\\_idx](#), [m\\_pst](#), and [reference\(\)](#).

Referenced by [~reference\(\)](#).

### 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](#).

References [m\\_idx](#), and [m\\_pst](#).

### 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](#).

References [m\\_idx](#), [m\\_pst](#), and [reference\(\)](#).

#### 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](#).

References [m\\_idx](#), [m\\_pst](#), and [reference\(\)](#).

#### 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](#).

References [m\\_idx](#), [m\\_pst](#), and [reference\(\)](#).

#### 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](#).

References [m\\_idx](#), and [m\\_pst](#).

### 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](#).

References [index\\_set](#).

Referenced by [index\\_set](#).

## 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](#).

Referenced by [flip\(\)](#), [operator bool\(\)](#), [operator=\(\)](#), [operator=\(\)](#), [operator==\(\)](#), [operator~\(\)](#), and [reference\(\)](#).

### 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](#).

Referenced by [flip\(\)](#), [operator bool\(\)](#), [operator=\(\)](#), [operator=\(\)](#), [operator==\(\)](#), [operator~\(\)](#), and [reference\(\)](#).

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 1108 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 1111 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 1113 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 1112 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 1115 of file [framed\\_multi\\_imp.h](#).

References [sorted\\_begin](#), and [sorted\\_end](#).

## 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 1122 of file [framed\\_multi\\_imp.h](#).

Referenced by [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 1123 of file [framed\\_multi\\_imp.h](#).

Referenced by [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 1127 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 1130 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 1132 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 1131 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 1134 of file [framed\\_multi\\_imp.h](#).

References [sorted\\_begin](#), and [sorted\\_end](#).



## 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 1138 of file [framed\\_multi\\_imp.h](#).

Referenced by [sorted\\_range\(\)](#).

### 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 1139 of file [framed\\_multi\\_imp.h](#).

Referenced by [sorted\\_range\(\)](#).

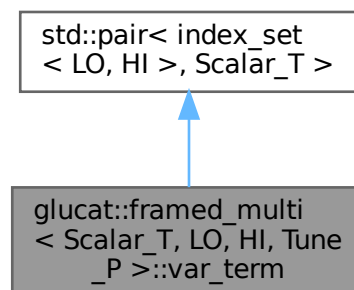
The documentation for this class was generated from the following file:

- [glucat/framed\\_multi\\_imp.h](#)

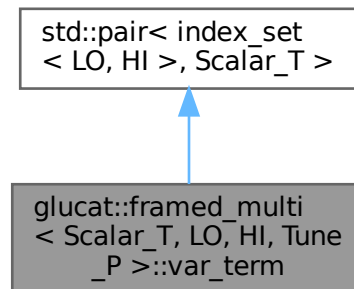
## 6.49 glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::var\_term Class Reference

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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI,
typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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 = double, const index\_t LO = DEFAULT_LO, const index\_t HI = DEFAULT_HI, typename Tune_P = tuning<>>
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>
```

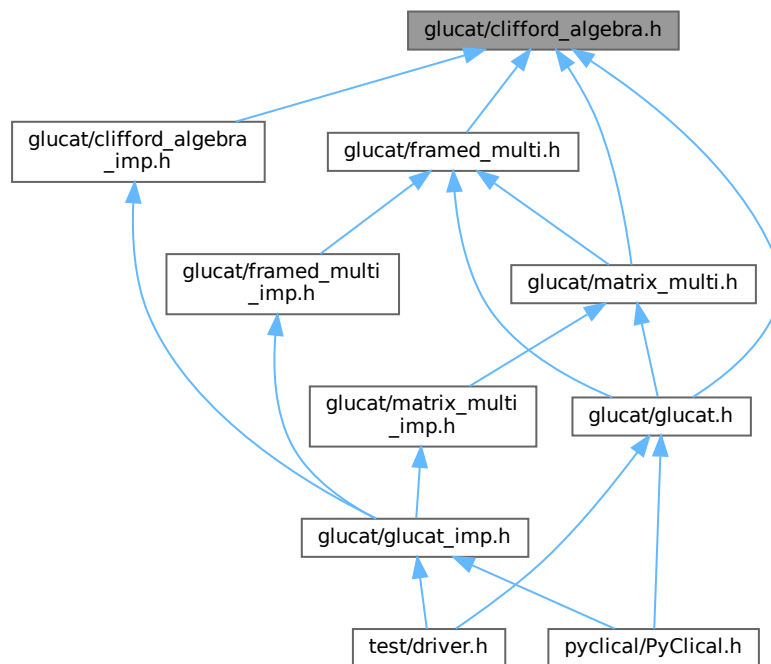
```
#include <utility>
```

```
#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 ==  $\sqrt{\text{norm}}$*

- `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 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 glucatt::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 glucatt::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.

Referenced by [glucat::matrix\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::basis\\_element\(\)](#), and [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::basis\\_element\(\)](#).

## 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;

```

```

00067     virtual auto operator== (const Scalar_T& scr) const -> bool = 0;
00069     virtual auto operator+= (const multivector_t& rhs) -> multivector_t& = 0;
00071     virtual auto operator*= (const Scalar_T& scr) -> multivector_t& = 0;
00073     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 -> const 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 -> const 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 -> const index_set_t = 0;
00103     virtual auto grade () const -> index_t = 0;
00105     virtual auto operator[] (const index_set_t ist) const -> Scalar_T = 0;
00107     virtual auto operator() (index_t grade) const -> const multivector_t = 0;
00109     virtual auto scalar () const -> Scalar_T = 0;
00111     virtual auto pure () const -> const multivector_t = 0;
00113     virtual auto even () const -> const multivector_t = 0;
00115     virtual auto odd () const -> const multivector_t = 0;
00117     virtual auto vector_part () const -> const 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 -> const multivector_t = 0;
00123     virtual auto reverse () const -> const multivector_t = 0;
00125     virtual auto conj () const -> const multivector_t = 0;
00127     virtual auto quad () const -> Scalar_T = 0;
00129     virtual auto norm () const -> Scalar_T = 0;
00131     virtual auto max_abs () const -> Scalar_T = 0;
00133     virtual auto truncated (const Scalar_T& limit = default_truncation) const -> const multivector_t
= 0;
00135     virtual auto isinf () const -> bool = 0;
00137     virtual auto isnan () const -> 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 -> 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 -> 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 -> const index_set_t override;\
00165     auto grade () const -> index_t override;\
00166     auto operator[] (const index_set_t ist) const -> Scalar_T override;\
00167     auto operator() (index_t grade) const -> const multivector_t override;\
00168     auto scalar () const -> Scalar_T override;\
00169     auto pure () const -> const multivector_t override;\
00170     auto even () const -> const multivector_t override;\
00171     auto odd () const -> const multivector_t override;\
00172     auto vector_part () const -> 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 -> const multivector_t override;\
00176     auto reverse () const -> const multivector_t override;\
00177     auto conj () const -> const multivector_t override;\
00178     auto quad () const -> Scalar_T override;\
00179     auto norm () const -> Scalar_T override;\
00180     auto max_abs () const -> Scalar_T override;\
00181     auto truncated (const Scalar_T& limit = multivector_t::default_truncation) const \
00182     -> const multivector_t override;\
00183     auto isinf () const -> bool override;\
00184     auto isnan () const -> 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

```

```

00190     template
00191     <
00192         template<typename, const index_t, const index_t, typename> class Multivector,
00193         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;

00385
00386 template
00387 <
00388     template<typename, const index_t, const index_t, typename> class Multivector,
00389     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00390 >
00391 auto
00392 operator/ (const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, const Scalar_T& scr) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;

00394
00395 template
00396 <
00397     template<typename, const index_t, const index_t, typename> class Multivector,
00398     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00399 >
00400 auto
00401 operator/ (const Scalar_T& scr, const Multivector<Scalar_T,LO,HI,Tune_P>& rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;

00403
00404 template
00405 <
00406     template<typename, const index_t, const index_t, typename> class Multivector,
00407     template<typename, const index_t, const index_t, typename> class RHS,
00408     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00409 >
00410 auto
00411 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>;

00413
00414 template
00415 <
00416     template<typename, const index_t, const index_t, typename> class Multivector,
00417     template<typename, const index_t, const index_t, typename> class RHS,
00418     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00419 >
00420 auto
00421 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>;

00423
00424 template
00425 <
00426     template<typename, const index_t, const index_t, typename> class Multivector,
00427     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00428 >
00429 auto
00430 inv(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

00432
00433 template
00434 <
00435     template<typename, const index_t, const index_t, typename> class Multivector,
00436     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00437 >
00438 auto
00439 pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, int rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;

00441
00442 template
00443 <
00444     template<typename, const index_t, const index_t, typename> class Multivector,
00445     template<typename, const index_t, const index_t, typename> class RHS,
00446     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00447 >
00448 auto
00449 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>;

00451
00452 template< template<typename, const index_t, const index_t, typename> class Multivector,
00453     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00454 auto
00455 outer_pow(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs, int rhs) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;

```

```

00457
00459     template
00460     <
00461         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
00468     template
00469     <
00470         template<typename, const index_t, const index_t, typename> class Multivector,
00471         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00472     >
00473     auto
00474     real(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00475
00477     template
00478     <
00479         template<typename, const index_t, const index_t, typename> class Multivector,
00480         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00481     >
00482     auto
00483     imag(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00484
00486     template
00487     <
00488         template<typename, const index_t, const index_t, typename> class Multivector,
00489         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00490     >
00491     auto
00492     pure(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00493
00495     template
00496     <
00497         template<typename, const index_t, const index_t, typename> class Multivector,
00498         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00499     >
00500     auto
00501     even(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00502
00504     template
00505     <
00506         template<typename, const index_t, const index_t, typename> class Multivector,
00507         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00508     >
00509     auto
00510     odd(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00511
00513     template
00514     <
00515         template<typename, const index_t, const index_t, typename> class Multivector,
00516         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00517     >
00518     auto
00519     vector_part(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const std::vector<Scalar_T>;
00520
00522     template
00523     <
00524         template<typename, const index_t, const index_t, typename> class Multivector,
00525         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00526     >
00527     auto
00528     involute(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00529
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     reverse(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00538
00540     template
00541     <
00542         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     >
00545     auto
00546     conj(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00547
00549     template
00550     <
00551         template<typename, const index_t, const index_t, typename> class Multivector,
00552         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00553     >
00554     auto

```

```

00555 quad(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00556
00558 template
00559 <
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
00567 template
00568 <
00569     template<typename, const index_t, const index_t, typename> class Multivector,
00570     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00571 >
00572 auto
00573 abs(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00574
00576 template
00577 <
00578     template<typename, const index_t, const index_t, typename> class Multivector,
00579     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00580 >
00581 auto
00582 max_abs(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T;
00583
00585 template
00586 <
00587     template<typename, const index_t, const index_t, typename> class Multivector,
00588     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00589 >
00590 auto
00591 complexifier(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00592
00595 template
00596 <
00597     template<typename, const index_t, const index_t, typename> class Multivector,
00598     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00599 >
00600 auto
00601 elliptic(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00602
00604 template
00605 <
00606     template<typename, const index_t, const index_t, typename> class Multivector,
00607     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00608 >
00609 auto
00610 sqrt(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00611       const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00612       const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00613
00615 template
00616 <
00617     template<typename, const index_t, const index_t, typename> class Multivector,
00618     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00619 >
00620 auto
00621 sqrt(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00622
00623 // Transcendental functions
00624
00626 template
00627 <
00628     template<typename, const index_t, const index_t, typename> class Multivector,
00629     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00630 >
00631 auto
00632 clifford_exp(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const
Multivector<Scalar_T,LO,HI,Tune_P>;
00633
00634 template
00635 <
00636     template<typename, const index_t, const index_t, typename> class Multivector,
00637     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00638 >
00639 auto
00640 log(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00641     const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00642     const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00643
00645 template
00646 <
00647     template<typename, const index_t, const index_t, typename> class Multivector,
00648     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00649 >
00650 auto

```

```

00651     log(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00652
00653     template
00654     <
00655         template<typename, const index_t, const index_t, typename> class Multivector,
00656         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00657     >
00658     auto
00659     cos(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00660         const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00661         const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00662
00663     template
00664     <
00665         template<typename, const index_t, const index_t, typename> class Multivector,
00666         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00667     >
00668     auto
00669     cos(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00670
00671     template
00672     <
00673         template<typename, const index_t, const index_t, typename> class Multivector,
00674         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00675     >
00676     auto
00677     acos(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00678         const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00679         const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00680
00681     template
00682     <
00683         template<typename, const index_t, const index_t, typename> class Multivector,
00684         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00685     >
00686     auto
00687     acos(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00688
00689     template
00690     <
00691         template<typename, const index_t, const index_t, typename> class Multivector,
00692         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00693     >
00694     auto
00695     acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00696
00697     template
00698     <
00699         template<typename, const index_t, const index_t, typename> class Multivector,
00700         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00701     >
00702     auto
00703     cosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00704
00705     template
00706     <
00707         template<typename, const index_t, const index_t, typename> class Multivector,
00708         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00709     >
00710     auto
00711     acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00712         const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00713         const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00714
00715     template
00716     <
00717         template<typename, const index_t, const index_t, typename> class Multivector,
00718         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00719     >
00720     auto
00721     acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00722
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
00733     template
00734     <
00735         template<typename, const index_t, const index_t, typename> class Multivector,
00736         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00737     >
00738     auto
00739     sin(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00740
00741     template
00742     <
00743         template<typename, const index_t, const index_t, typename> class Multivector,
00744         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00745     >
00746     auto
00747     sin(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;

```

```

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>;
00770
00771     template
00772     <
00773         template<typename, const index_t, const index_t, typename> class Multivector,
00774         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00775     >
00776     auto
00777     asinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00778           const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00779           const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00781
00782     template
00783     <
00784         template<typename, const index_t, const index_t, typename> class Multivector,
00785         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00786     >
00787     auto
00788     asinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00790
00791     template
00792     <
00793         template<typename, const index_t, const index_t, typename> class Multivector,
00794         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00795     >
00796     auto
00797     tan(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00798         const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00799         const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00801
00802     template
00803     <
00804         template<typename, const index_t, const index_t, typename> class Multivector,
00805         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00806     >
00807     auto
00808     tan(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00810
00811     template
00812     <
00813         template<typename, const index_t, const index_t, typename> class Multivector,
00814         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00815     >
00816     auto
00817     atan(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00818         const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00819         const bool prechecked = false) -> const Multivector<Scalar_T,LO,HI,Tune_P>;
00821
00822     template
00823     <
00824         template<typename, const index_t, const index_t, typename> class Multivector,
00825         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00826     >
00827     auto
00828     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>;
00839
00840     template
00841     <
00842         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

```

```

00845 >
00846 auto
00847 atanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00848       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

```

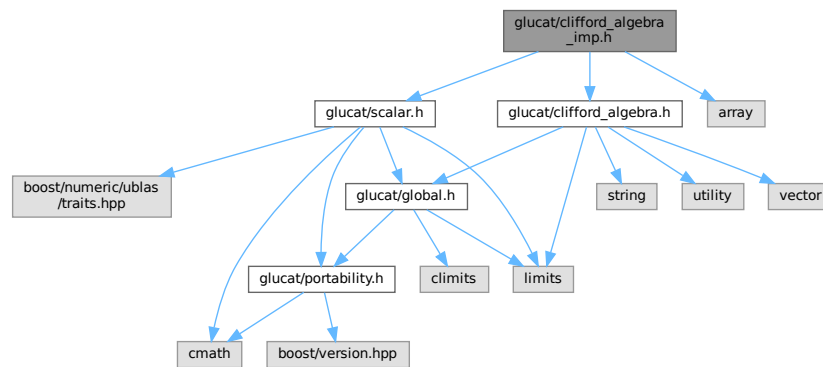
### 7.3 glucat/clifford\_algebra\_imp.h File Reference

```

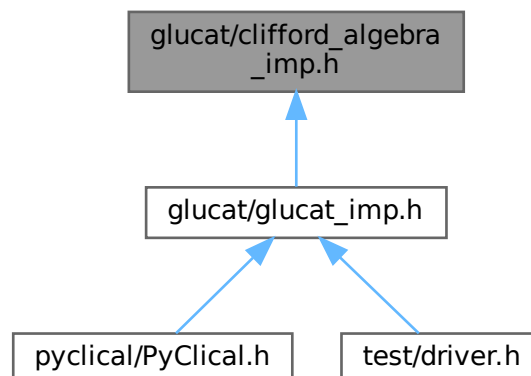
#include "glucat/clifford_algebra.h"
#include "glucat/scalar.h"
#include <array>

```

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.  $\{1\}*\{2\} \rightarrow (-\{2\})*(-\{1\})$*

- `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,  $rev \circ invo == invo \circ rev$ .*

- `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.*

## Variables

- `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()`

*Default for truncation.*

## 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
00071     template< typename Scalar_T, typename Index_Set_T, typename Multivector_T>
00072     const
00073     Scalar_T
00074     clifford_algebra<Scalar_T, Index_Set_T, Multivector_T>::
00075     default_truncation = std::numeric_limits<Scalar_T>::epsilon();
00076
00078     template
00079     <
00080         template<typename, const index_t, const index_t, typename> class Multivector,
00081         template<typename, const index_t, const index_t, typename> class RHS,
00082         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00083     >
00084     inline
00085     auto
00086     operator!= (const Multivector<Scalar_T, LO, HI, Tune_P>& lhs, const RHS<Scalar_T, LO, HI, Tune_P>& rhs) ->
bool
00087     { return !(lhs == rhs); }
00088
00090     template< template<typename, const index_t, const index_t, typename> class Multivector,
00091               typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00092     inline
00093     auto
00094     operator!= (const Multivector<Scalar_T, LO, HI, Tune_P>& lhs, const Scalar_T& scr) -> bool
00095     { return !(lhs == scr); }
00096
00098     template< template<typename, const index_t, const index_t, typename> class Multivector,
00099               typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00100     inline
00101     auto
00102     operator!= (const Scalar_T& scr, const Multivector<Scalar_T, LO, HI, Tune_P>& rhs) -> bool
00103     { return !(rhs == scr); }
00104
00106     template
00107     <
00108         template<typename, const index_t, const index_t, typename> class Multivector,
00109         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00110     >
00111     auto
00112     error_squared_tol(const Multivector<Scalar_T, LO, HI, Tune_P>& val) -> Scalar_T
00113     {
00114         using multivector_t = Multivector<Scalar_T, LO, HI, Tune_P>;
00115         static const auto scalar_eps = std::numeric_limits<Scalar_T>::epsilon();
00116         static const auto nbr_different_bits =
00117             std::numeric_limits<Scalar_T>::digits / Tune_P::denom_different_bits +
Tune_P::extra_different_bits;
00118         static const auto abs_tol = scalar_eps *
00119             numeric_traits<Scalar_T>::pow(Scalar_T(2), nbr_different_bits);
00120         using framed_multi_t = typename multivector_t::framed_multi_t;
00121         const auto nbr_terms = double(framed_multi_t(val).truncated(scalar_eps).nbr_terms());

```

```

00122     return abs_tol * abs_tol * std::max(Scalar_T(nbr_terms), Scalar_T(1));
00123 }
00124
00126 template
00127 <
00128     template<typename, const index_t, const index_t, typename> class Multivector,
00129     template<typename, const index_t, const index_t, typename> class RHS,
00130     typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00131 >
00132 inline
00133 auto
00134 error_squared(const Multivector<Scalar_T,LO,HI,Tune_P>& lhs,
00135               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
Multivector<Scalar_T,LO,HI,Tune_P>
00182 {
00183     auto result = lhs;
00184     return result += scr;
00185 }
00186
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
Multivector<Scalar_T,LO,HI,Tune_P>
00193 {
00194     return rhs + scr;
00195 }
00196
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) ->
const Multivector<Scalar_T,LO,HI,Tune_P>
00207 {
00208     auto result = lhs;
00209     return result += rhs;
00210 }
00211

```



```

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
Multivector<Scalar_T,LO,HI,Tune_P>
00218     {
00219         auto result = lhs;
00220         return result -= scr;
00221     }
00222
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
00395     template< 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     inline
00398     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
00403     template< template<typename, const index_t, const index_t, typename> class Multivector,
00404               typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00405     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
00437     template
00438     <
00439         template<typename, const index_t, const index_t, typename> class Multivector,
00440         template<typename, const index_t, const index_t, typename> class RHS,
00441         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P
00442     >
00443     inline
00444     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
00466     template< template<typename, const index_t, const index_t, typename> class Multivector,
00467               typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00468     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
00473     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
00501     template< 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     inline
00504     auto
00505     pure(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00506     { return val - val.scalar(); }
00507
00509     template< 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     inline
00512     auto
00513     even(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00514     { return val.even(); }
00515
00517     template< 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     inline
00520     auto
00521     odd(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00522     { return val.odd(); }
00523
00525     template< 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     inline
00528     auto
00529     vector_part(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const std::vector<Scalar_T>
00530     { return val.vector_part(); }
00531
00533     template< 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     inline
00536     auto
00537     involute(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00538     { return val.involute(); }
00539
00541     template< 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     inline
00544     auto
00545     reverse(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00546     { return val.reverse(); }
00547
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     conj(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00554     { return val.conj(); }
00555
00557     template< template<typename, const index_t, const index_t, typename> class Multivector,
00558               typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00559     inline
00560     auto
00561     quad(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
00562     { return val.quad(); }
00563
00565     template< template<typename, const index_t, const index_t, typename> class Multivector,
00566               typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00567     inline
00568     auto
00569     norm(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
00570     { return val.norm(); }
00571
00573     template< template<typename, const index_t, const index_t, typename> class Multivector,

```

```

00574         typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00575     inline
00576     auto
00577     abs(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
00578     { return numeric_traits<Scalar_T>::sqrt(val.norm()); }
00579
00581     template< template<typename, const index_t, const index_t, typename> class Multivector,
00582             typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00583     inline
00584     auto
00585     max_abs(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> Scalar_T
00586     { return val.max_abs(); }
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
00593     Multivector<Scalar_T,LO,HI,Tune_P>
00594     {
00595         using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00596         using traits_t = numeric_traits<Scalar_T>;
00597
00598         auto frm = val.frame();
00599         using array_t = std::array<index_t, 4>;
00600         auto incp = array_t{0, 2, 1, 0};
00601         auto incq = array_t{1, 0, 0, 0};
00602         auto bott = pos_mod((frm.count_pos() - frm.count_neg()), 4);
00603         for (auto
00604             k = index_t(0);
00605             k != incp[bott];
00606             k++)
00607             for (auto
00608                 idx = index_t(1);
00609                 idx != HI+1;
00610                 ++idx)
00611                 if (!frm[idx])
00612                 {
00613                     frm.set(idx);
00614                     break;
00615                 }
00616             for (auto
00617                 k = index_t(0);
00618                 k != incq[bott];
00619                 k++)
00620                 for (auto
00621                     idx = index_t(-1);
00622                     idx != LO-1;
00623                     --idx)
00624                     if (!frm[idx])
00625                     {
00626                         frm.set(idx);
00627                         break;
00628                     }
00629         auto new_bott = pos_mod(frm.count_pos() - frm.count_neg(), 4);
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
00639     template< template<typename, const index_t, const index_t, typename> class Multivector,
00640             typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00641     inline
00642     auto
00643     elliptic(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00644     { return complexifier(val); }
00645
00647     template< template<typename, const index_t, const index_t, typename> class Multivector,
00648             typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00649     inline
00650     static
00651     void
00652     check_complex(const Multivector<Scalar_T,LO,HI,Tune_P>& val,
00653                  const Multivector<Scalar_T,LO,HI,Tune_P>& i, const bool prechecked = false)
00654     {
00655         if (!prechecked)
00656         {
00657             using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00658             using error_t = typename multivector_t::error_t;
00659
00660             const auto i_frame = i.frame();
00661             // We need i to be a complexifier whose frame is large enough to represent val
00662             if (complexifier(i) != i ||
00663                 (val.frame() | i_frame) != i_frame ||
00664                 complexifier(val).frame().count() > i_frame.count())

```

```

00665         throw error_t("check_complex(val, i): i is not a valid complexifier for val");
00666     }
00667 }
00668
00670 template< template<typename, const index_t, const index_t, typename> class Multivector,
00671           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00672 inline
00673 auto
00674 sqrt(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
{ return sqrt(val, i, prechecked); }
00675
00676
00678 template< template<typename, const index_t, const index_t, typename> class Multivector,
00679           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00680 inline
00681 auto
00682 sqrt(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00683 { return sqrt(val, complexifier(val), true); }
00684
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
Multivector<Scalar_T,LO,HI,Tune_P>
00690 {
00691     // Scaling and squaring Pade' approximation of matrix exponential
00692     // Reference: [GL], Section 11.3, p572-576
00693     // Reference: [H]
00694
00695     using traits_t = numeric_traits<Scalar_T>;
00696
00697     const auto scalar_val = val.scalar();
00698     const auto scalar_exp = traits_t::exp(scalar_val);
00699     if (traits_t::isNaN_or_isInf(scalar_exp))
00700         return traits_t::NaN();
00701     if (val == scalar_val)
00702         return scalar_exp;
00703
00704     using multivector_t = Multivector<Scalar_T,LO,HI,Tune_P>;
00705     auto A = val - scalar_val;
00706     const auto pure_scale2 = A.norm();
00707
00708     if (traits_t::isNaN_or_isInf(pure_scale2))
00709         return traits_t::NaN();
00710     if (pure_scale2 == Scalar_T(0))
00711         return scalar_exp;
00712
00713     const auto ilog2_scale =
00714         std::max(0, traits_t::to_int(ceil((log2(pure_scale2) +
Scalar_T(A.frame().count())/Scalar_T(2))) - 3));
00715     const auto i_scale = traits_t::pow(Scalar_T(2), ilog2_scale);
00716     if (traits_t::isNaN_or_isInf(i_scale))
00717         return traits_t::NaN();
00718
00719     A /= i_scale;
00720     multivector_t pure_exp;
00721     {
00722         using limits_t = std::numeric_limits<Scalar_T>;
00723         const auto nbr_even_powers = 2*(limits_t::digits / 32) + 4;
00724         using nbr_t = decltype(nbr_even_powers);
00725
00726         // Create an array of coefficients
00727         const auto max_power = 2*nbr_even_powers + 1;
00728         static std::array<Scalar_T, max_power+1> c;
00729         if (c[0] != Scalar_T(1))
00730         {
00731             c[0] = Scalar_T(1);
00732             for (auto
00733                 k = decltype(max_power)(0);
00734                 k != max_power;
00735                 ++k)
00736                 c[k+1] = c[k]*(max_power-k) / ((2*max_power-k)*(k+1));
00737         }
00738
00739         // Create an array of even powers
00740         std::array<multivector_t, nbr_even_powers> AA;
00741         AA[0] = A * A;
00742         AA[1] = AA[0] * AA[0];
00743         for (auto
00744             k = nbr_t(2);
00745             k != nbr_even_powers;
00746             ++k)
00747             AA[k] = AA[k-2] * AA[1];
00748
00749         // Use compensated summation to calculate U and AV
00750         auto residual = multivector_t();
00751         auto U = multivector_t(c[0]);

```

```

00752     for (auto
00753         k = nbr_t(0);
00754         k != nbr_even_powers;
00755         ++k)
00756     {
00757         const auto& term = AA[k]*c[2*k + 2] - residual;
00758         const auto& sum = U + term;
00759         residual = (sum - U) - term;
00760         U = sum;
00761     }
00762     residual = multivector_t();
00763     auto AV = multivector_t(c[1]);
00764     for (auto
00765         k = nbr_t(0);
00766         k != nbr_even_powers;
00767         ++k)
00768     {
00769         const auto& term = AA[k]*c[2*k + 3] - residual;
00770         const auto& sum = AV + term;
00771         residual = (sum - AV) - term;
00772         AV = sum;
00773     }
00774     AV *= A;
00775     pure_exp = (U+AV) / (U-AV);
00776 }
00777 for (auto
00778     k = decltype(ilog2_scale)(0);
00779     k != ilog2_scale;
00780     ++k)
00781     pure_exp *= pure_exp;
00782 return pure_exp * scalar_exp;
00783 }
00784
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
prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00791 { return log(val, i, prechecked); }
00792
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
00802 template< template<typename, const index_t, const index_t, typename> class Multivector,
00803           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00804 inline
00805 auto
00806 cosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00807 {
00808     using traits_t = numeric_traits<Scalar_T>;
00809     if (val.isnan())
00810         return traits_t::NaN();
00811
00812     const auto& s = val.scalar();
00813     if (val == s)
00814         return traits_t::cosh(s);
00815     return (exp(val)+exp(-val)) / Scalar_T(2);
00816 }
00817
00819 // Reference: [AS], Section 4.6, p86-89
00820 template< template<typename, const index_t, const index_t, typename> class Multivector,
00821           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00822 inline
00823 auto
00824 acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00825 {
00826     using traits_t = numeric_traits<Scalar_T>;
00827     check_complex(val, i, prechecked);
00828     if (val.isnan())
00829         return traits_t::NaN();
00830
00831     const auto radical = sqrt(val*val - Scalar_T(1), i, true);
00832     return (norm(val + radical) >= norm(val))
00833         ? log(val + radical, i, true)
00834         : -log(val - radical, i, true);
00835 }
00836
00838 // Reference: [AS], Section 4.6, p86-89
00839 template< template<typename, const index_t, const index_t, typename> class Multivector,
00840           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00841 inline

```

```

00842     auto
00843     acosh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00844     { return acosh(val, complexifier(val), true); }
00845
00846     template< template<typename, const index_t, const index_t, typename> class Multivector,
00847               typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00848     auto
00849     cos(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool
00850     prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00851     {
00852         using traits_t = numeric_traits<Scalar_T>;
00853         if (val.isnan())
00854             return traits_t::NaN();
00855
00856         const auto& s = val.scalar();
00857         if (val == s)
00858             return traits_t::cos(s);
00859
00860         check_complex(val, i, prechecked);
00861
00862         static const auto& twopi = Scalar_T(2) * traits_t::pi();
00863         const auto& z = i *
00864             (val - s + traits_t::fmod(s, twopi));
00865         return (exp(z)+exp(-z)) / Scalar_T(2);
00866     }
00867
00868     template< template<typename, const index_t, const index_t, typename> class Multivector,
00869               typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00870     inline
00871     auto
00872     cos(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00873     { return cos(val, complexifier(val), true); }
00874
00875     // Reference: [AS], Section 4.4, p79-83
00876     template< template<typename, const index_t, const index_t, typename> class Multivector,
00877               typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00878     inline
00879     auto
00880     acos(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00881     bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00882     {
00883         using traits_t = numeric_traits<Scalar_T>;
00884         if (val.isnan())
00885             return traits_t::NaN();
00886
00887         const auto& s = val.scalar();
00888         if (val == s && traits_t::abs(s) <= Scalar_T(1))
00889             return traits_t::acos(s);
00890
00891         check_complex(val, i, prechecked);
00892         return i * acosh(val, i, true);
00893     }
00894
00895     // Reference: [AS], Section 4.4, p79-83
00896     template< template<typename, const index_t, const index_t, typename> class Multivector,
00897               typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00898     inline
00899     auto
00900     acos(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00901     { return acos(val, complexifier(val), true); }
00902
00903     template< template<typename, const index_t, const index_t, typename> class Multivector,
00904               typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00905     inline
00906     auto
00907     sinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00908     {
00909         using traits_t = numeric_traits<Scalar_T>;
00910         if (val.isnan())
00911             return traits_t::NaN();
00912
00913         const auto& s = val.scalar();
00914         if (val == s)
00915             return traits_t::sinh(s);
00916
00917         return (exp(val)-exp(-val)) / Scalar_T(2);
00918     }
00919
00920     // Reference: [AS], Section 4.6, p86-89
00921     template< template<typename, const index_t, const index_t, typename> class Multivector,
00922               typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00923     inline
00924     auto
00925     asinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00926     bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00927     {
00928         using traits_t = numeric_traits<Scalar_T>;

```

```

00932     check_complex(val, i, prechecked);
00933     if (val.isnan())
00934         return traits_t::NaN();
00935
00936     const auto radical = sqrt(val*val + Scalar_T(1), i, true);
00937     return (norm(val + radical) >= norm(val))
00938         ? log( val + radical, i, true)
00939         : -log(-val + radical, i, true);
00940 }
00941
00942 // Reference: [AS], Section 4.6, p86-89
00943 template< template<typename, const index_t, const index_t, typename> class Multivector,
00944           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00945 inline
00946 auto
00947 asinh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00948 { return asinh(val, complexifier(val), true); }
00949
00950 template< template<typename, const index_t, const index_t, typename> class Multivector,
00951           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00952 inline
00953 auto
00954 sin(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool
00955 prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00956 {
00957     using traits_t = numeric_traits<Scalar_T>;
00958     if (val.isnan())
00959         return traits_t::NaN();
00960
00961     const auto& s = val.scalar();
00962     if (val == s)
00963         return traits_t::sin(s);
00964
00965     check_complex(val, i, prechecked);
00966
00967     static const auto& twopi = Scalar_T(2) * traits_t::pi();
00968     const auto& z = i *
00969         (val - s + traits_t::fmod(s, twopi));
00970     return i * (exp(-z)-exp(z)) / Scalar_T(2);
00971 }
00972
00973 template< template<typename, const index_t, const index_t, typename> class Multivector,
00974           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00975 inline
00976 auto
00977 sin(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00978 { return sin(val, complexifier(val), true); }
00979
00980 // Reference: [AS], Section 4.4, p79-83
00981 template< template<typename, const index_t, const index_t, typename> class Multivector,
00982           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00983 inline
00984 auto
00985 asin(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
00986 bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
00987 {
00988     using traits_t = numeric_traits<Scalar_T>;
00989     if (val.isnan())
00990         return traits_t::NaN();
00991
00992     const auto& s = val.scalar();
00993     if (val == s && traits_t::abs(s) <= Scalar_T(1))
00994         return traits_t::asin(s);
00995
00996     check_complex(val, i, prechecked);
00997     return -i * asinh(i * val, i, true);
00998 }
00999
01000 // Reference: [AS], Section 4.4, p79-83
01001 template< template<typename, const index_t, const index_t, typename> class Multivector,
01002           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01003 inline
01004 auto
01005 asin(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
01006 { return asin(val, complexifier(val), true); }
01007
01008 template< template<typename, const index_t, const index_t, typename> class Multivector,
01009           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01010 inline
01011 auto
01012 tanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val) -> const Multivector<Scalar_T,LO,HI,Tune_P>
01013 {
01014     using traits_t = numeric_traits<Scalar_T>;
01015     if (val.isnan())
01016         return traits_t::NaN();
01017
01018     const auto& s = val.scalar();
01019     if (val == s)

```



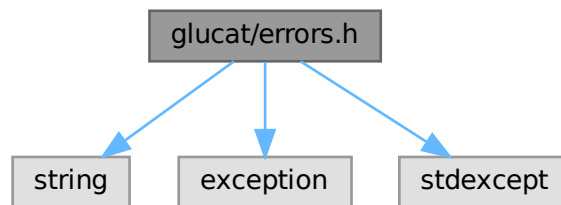
```

01023         return traits_t::tanh(s);
01024
01025     return sinh(val) / cosh(val);
01026 }
01027
01029 // Reference: [AS], Section 4.6, p86-89
01030 template< template<typename, const index_t, const index_t, typename> class Multivector,
01031           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01032 inline
01033 auto
01034 atanh(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
01035        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 }
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
01055 template< template<typename, const index_t, const index_t, typename> class Multivector,
01056           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01057 inline
01058 auto
01059 tan(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i, bool
01060      prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
01061 {
01062     using traits_t = numeric_traits<Scalar_T>;
01063     if (val.isnan())
01064         return traits_t::NaN();
01065     const auto& s = val.scalar();
01066     if (val == s)
01067         return traits_t::tan(s);
01068     check_complex(val, i, prechecked);
01069     return sin(val, i, true) / cos(val, i, true);
01070 }
01072
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
01082 // Reference: [AS], Section 4.4, p79-83
01083 template< template<typename, const index_t, const index_t, typename> class Multivector,
01084           typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01085 inline
01086 auto
01087 atan(const Multivector<Scalar_T,LO,HI,Tune_P>& val, const Multivector<Scalar_T,LO,HI,Tune_P>& i,
01088       bool prechecked) -> const Multivector<Scalar_T,LO,HI,Tune_P>
01089 {
01090     using traits_t = numeric_traits<Scalar_T>;
01091     if (val.isnan())
01092         return traits_t::NaN();
01093     const auto& s = val.scalar();
01094     if (val == s)
01095         return traits_t::atan(s);
01096     check_complex(val, i, prechecked);
01097     return -i * atanh(i * val, i, true);
01098 }
01099
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 #endif // _GLUCAT_CLIFFORD_ALGEBRA_IMP_H

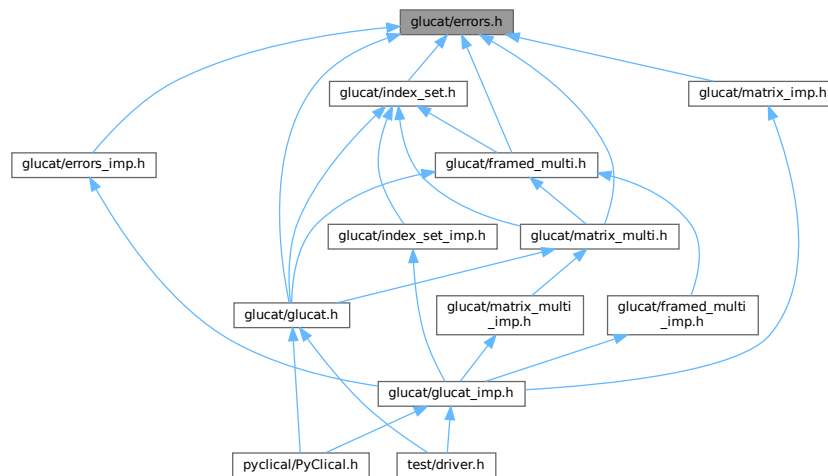
```

## 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 }
00065 #endif // _GLUCAT_ERRORS_H

```

## 7.7 glucat/errors\_imp.h File Reference

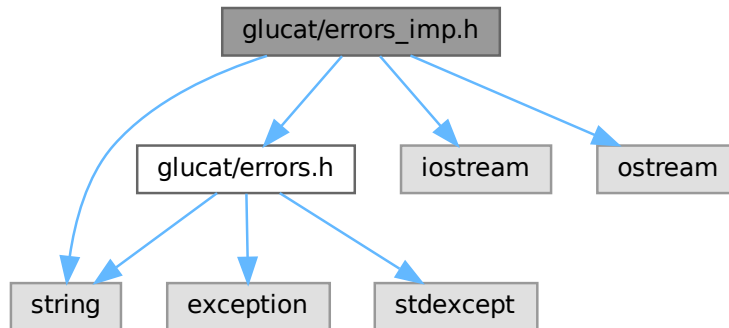
```

#include "glucate/errors.h"
#include <string>
#include <iostream>

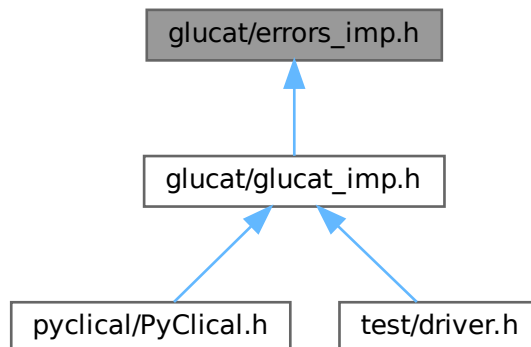
```

```
#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

```

## 7.9 glucat/framed\_multi.h File Reference

```

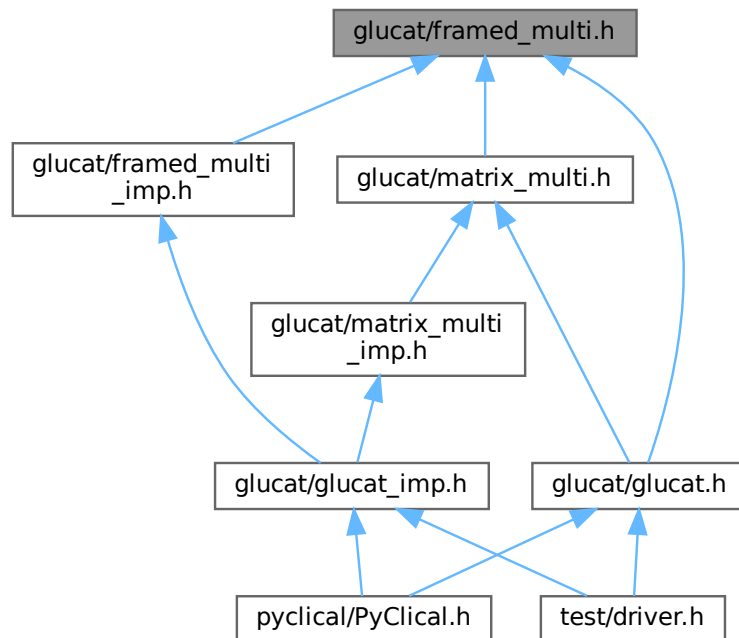
#include "glucat/global.h"
#include "glucat/errors.h"
#include "glucat/index_set.h"
#include "glucat/clifford_algebra.h"
#include "glucat/tuning.h"
#include <string>
#include <utility>
#include <map>

```

```
#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/poolfwd.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
00065     framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00066
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
00070     framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00071
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
00075     framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00076
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
00080     framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00081
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)
00085     -> Scalar_T;
00086
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
00090     framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00091
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
00095     framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>;
00096
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) ->
00110     std::ostream&;
00111
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,
00126     typename Tune_P = tuning<> >
00127     class framed_multi :
00128     public clifford_algebra< Scalar_T, index_set<LO,HI>, framed_multi<Scalar_T,LO,HI,Tune_P> >,
00129     private std::unordered_map< index_set<LO,HI>, Scalar_T, index_set_hash<LO,HI> >
00130     {
00131     public:
00132         using multivector_t = framed_multi;
00133         using framed_multi_t = multivector_t;
00134         using scalar_t = Scalar_T;
00135         using tune_p = Tune_P;
00136         using index_set_t = index_set<LO, HI>;
00137         using term_t = std::pair<const index_set_t, Scalar_T>;
00138         using vector_t = std::vector<Scalar_T>;
00139         using error_t = error<multivector_t>;
00140         using matrix_multi_t = matrix_multi<Scalar_T,LO,HI,Tune_P >;
00141         template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
00142         Other_Tune_P >
00143         friend class matrix_multi;
00144         template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
00145         Other_Tune_P >
00146         friend class framed_multi;

```

```

00145 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

```

```

00252     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](#).

Referenced by [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#), and [glucat::framed\\_multi< Scalar\\_T, LO, HI, Tune\\_P >::framed\\_multi\(\)](#).

### 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)

```

```

00088 : map_t(_GLUCAT_HASH_N(val.size()))
00089 {
00090     if (!prechecked && (val.frame() | frm) != frm)
00091         throw error_t("multivector_t(val,frm): cannot initialize with value outside of frame");
00092     for (auto& val_term : val)
00093         this->insert(term_t(val_term.first, numeric_traits<Scalar_T>::to_scalar_t(val_term.second)));
00094 }
00095
00097 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00098 framed_multi<Scalar_T,LO,HI,Tune_P>::
00099 framed_multi(const multivector_t& val,
00100             const index_set_t frm, const bool prechecked)
00101 : map_t(_GLUCAT_HASH_N(val.size()))
00102 {
00103     if (!prechecked && (val.frame() | frm) != frm)
00104         throw error_t("multivector_t(val,frm): cannot initialize with value outside of frame");
00105     for (auto& val_term : val)
00106         this->insert(val_term);
00107 }
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.size();
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     try
00221     {
00222         *this = (val.template fast_framed_multi<Scalar_T>()).truncated();
00223         return;
00224     }
00225     catch (const glucat_error& e)
00226     { }
00227
00228     const auto val_norm = traits_t::to_scalar_t(val.norm());
00229     if (traits_t::isNaN_or_isInf(val_norm))
00230     {
00231         *this = traits_t::NaN();
00232         return;
00233     }
00234     const auto frm = val.frame();
00235     const auto algebra_dim = set_value_t(1) << frm.count();
00236     auto result = multivector_t(
00237         _GLUCAT_HASH_SIZE_T(std::min<size_t>(algebra_dim, matrix::nnz(val.m_matrix))));
00238     for (auto
00239         stv = set_value_t(0);
00240         stv != algebra_dim;
00241         stv++)
00242     {
00243         const auto ist = index_set_t(stv, frm, true);
00244         const auto crd =
00245             traits_t::to_scalar_t(matrix::inner<Other_Scalar_T>(val.basis_element(ist), val.m_matrix));
00246         if (crd != Scalar_T(0))
00247             result.insert(term_t(ist, crd));
00248     }
00249     *this = result.truncated();
00250 }
00251
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
00291 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00292 inline
00293 auto
00294 framed_multi<Scalar_T,LO,HI,Tune_P>::
00295 operator+= (const Scalar_T& scr) -> multivector_t&
00296 {
00297     *this += term_t(index_set_t(), scr);
00298     return *this;
00299 }
00300
00301 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00302 inline
00303 auto
00304 framed_multi<Scalar_T,LO,HI,Tune_P>::
00305 operator+= (const multivector_t& rhs) -> multivector_t&
00306 { // simply add terms
00307     for (auto& rhs_term : rhs)
00308         *this += rhs_term;
00309     return *this;
00310 }
00311
00312 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00313 inline
00314 auto
00315 framed_multi<Scalar_T,LO,HI,Tune_P>::
00316 operator-= (const Scalar_T& scr) -> multivector_t&
00317 {
00318     *this += term_t(index_set_t(), -scr);
00319     return *this;
00320 }
00321
00322 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00323 inline
00324 auto
00325 framed_multi<Scalar_T,LO,HI,Tune_P>::
00326 operator-= (const multivector_t& rhs) -> multivector_t&
00327 {
00328     for (auto& rhs_term : rhs)
00329         *this += term_t(rhs_term.first, -(rhs_term.second));
00330     return *this;
00331 }
00332
00333 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00334 inline
00335 auto
00336 framed_multi<Scalar_T,LO,HI,Tune_P>::
00337 operator- () const -> const multivector_t
00338 { // multiply coordinates of all terms by -1
00339     auto result = *this;
00340     for (auto& result_term : result)
00341         result_term.second *= Scalar_T(-1);
00342     return result;
00343 }
00344
00345 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00346 auto
00347 framed_multi<Scalar_T,LO,HI,Tune_P>::
00348 operator*= (const Scalar_T& scr) -> multivector_t&
00349 { // multiply coordinates of all terms by scalar
00350     using traits_t = numeric_traits<Scalar_T>;
00351
00352     if (traits_t::isNaN_or_isInf(scr))
00353         return *this = traits_t::NaN();
00354     if (scr == Scalar_T(0))
00355         if (this->isnan())
00356             *this = traits_t::NaN();
00357         else
00358             this->clear();
00359     else

```

```

00366         for (auto& this_term : *this)
00367             this_term.second *= scr;
00368         return *this;
00369     }
00370
00371     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00372     auto
00373     operator* (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
00374     framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
00375     {
00376         using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00377         using traits_t = numeric_traits<Scalar_T>;
00378
00379         if (lhs.isnan() || rhs.isnan())
00380             return traits_t::NaN();
00381
00382         const double lhs_size = lhs.size();
00383         const double rhs_size = rhs.size();
00384         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
00405     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00406     inline
00407     auto
00408     framed_multi<Scalar_T,LO,HI,Tune_P>::
00409     operator*= (const multivector_t& rhs) -> multivector_t&
00410     { return *this = *this * rhs; }
00411
00412     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00413     auto
00414     operator^ (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
00415     framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
00416     { // Arvind Raja's original reference:
00417         // "old clical, outerproduct(p,q:pterm):pterm in file compmod.pas"
00418
00419         if (lhs.empty() || rhs.empty())
00420             return Scalar_T(0);
00421
00422         using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00423         using index_set_t = typename multivector_t::index_set_t;
00424         using term_t = typename multivector_t::term_t;
00425
00426         const auto empty_set = index_set_t();
00427
00428         const double lhs_size = lhs.size();
00429         const double rhs_size = rhs.size();
00430         const auto lhs_frame = lhs.frame();
00431         const auto rhs_frame = rhs.frame();
00432         const auto our_frame = lhs_frame | rhs_frame;
00433         const auto algebra_dim = set_value_t(1) << our_frame.count();
00434         auto result = multivector_t(
00435             _GLUCAT_HASH_SIZE_T(size_t(std::min(lhs_size * rhs_size, double(algebra_dim)))));
00436         const auto lhs_end = lhs.end();
00437         const auto rhs_end = rhs.end();
00438
00439         if (lhs_size * rhs_size > double(Tune_P::products_size_threshold))
00440         {
00441             for (auto
00442                 result_stv = set_value_t(0);
00443                 result_stv != algebra_dim;
00444                 ++result_stv)
00445             {
00446                 const auto result_ist = index_set_t(result_stv, our_frame, true);
00447                 const auto lhs_result_frame = lhs_frame & result_ist;
00448                 const auto lhs_result_dim = set_value_t(1) << lhs_result_frame.count();
00449                 auto result_crd = Scalar_T(0);
00450                 for (auto
00451                     lhs_stv = set_value_t(0);
00452                     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
00498 if (lhs.empty() || rhs.empty())
00499     return Scalar_T(0);
00500
00501 using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00502 using index_set_t = typename multivector_t::index_set_t;
00503 using term_t = typename multivector_t::term_t;
00504
00505 const auto lhs_end = lhs.end();
00506 const auto rhs_end = rhs.end();
00507 const double lhs_size = lhs.size();
00508 const double rhs_size = rhs.size();
00509
00510 const auto lhs_frame = lhs.frame();
00511 const auto rhs_frame = rhs.frame();
00512
00513 const auto our_frame = lhs_frame | rhs_frame;
00514 const auto algebra_dim = set_value_t(1) << our_frame.count();
00515 auto result = multivector_t(
00516     _GLUCAT_HASH_SIZE_T(size_t(std::min(lhs_size * rhs_size, double(algebra_dim)))));
00517 if (lhs_size * rhs_size > double(Tune_P::products_size_threshold))
00518 {
00519     for (auto
00520         result_stv = set_value_t(0);
00521         result_stv != algebra_dim;
00522         ++result_stv)
00523     {
00524         const auto result_ist = index_set_t(result_stv, our_frame, true);
00525         const auto comp_frame = our_frame & ~result_ist;
00526         const auto comp_dim = set_value_t(1) << comp_frame.count();
00527         auto result_crd = Scalar_T(0);
00528         for (auto
00529             comp_stv = set_value_t(1);
00530             comp_stv != comp_dim;
00531             ++comp_stv)
00532         {
00533             const auto comp_ist = index_set_t(comp_stv, comp_frame, true);
00534             const auto our_ist = result_ist ^ comp_ist;
00535             if ((our_ist | lhs_frame) == lhs_frame)
00536             {
00537                 const auto lhs_it = lhs.find(our_ist);
00538                 if (lhs_it != lhs_end)
00539                 {
00540                     const auto rhs_it = rhs.find(comp_ist);

```

```

00542         if (rhs_it != rhs_end)
00543             result_crd += crd_of_mult(*lhs_it, *rhs_it);
00544     }
00545 }
00546 if (result_stv != 0)
00547 {
00548     if ((our_ist | rhs_frame) == rhs_frame)
00549     {
00550         const auto rhs_it = rhs.find(our_ist);
00551         if (rhs_it != rhs_end)
00552         {
00553             const auto lhs_it = lhs.find(comp_ist);
00554             if (lhs_it != lhs_end)
00555                 result_crd += crd_of_mult(*lhs_it, *rhs_it);
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
00610     const auto lhs_end = lhs.end();
00611     const auto rhs_end = rhs.end();
00612     const double lhs_size = lhs.size();
00613     const double rhs_size = rhs.size();
00614     const auto lhs_frame = lhs.frame();
00615     const auto rhs_frame = rhs.frame();
00616
00617     const auto our_frame = lhs_frame | rhs_frame;
00618     const auto algebra_dim = set_value_t(1) << our_frame.count();
00619     auto result = multivector_t(
00620         _GLUCAT_HASH_SIZE_T(size_t(std::min(lhs_size * rhs_size, double(algebra_dim)))));
00621
00622     if (lhs_size * rhs_size > double(Tune_P::products_size_threshold))
00623     {
00624         for (auto
00625             result_stv = set_value_t(0);
00626             result_stv != algebra_dim;
00627             ++result_stv)
00628         {
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)
00648                     result_crd += crd_of_mult(*lhs_it, *rhs_it);
00649             }
00650         }
00651     }
00652     if (result_crd != Scalar_T(0))
00653         result.insert(term_t(result_ist, result_crd));
00654 }
00655 }
00656 else
00657 {
00658     for (auto& rhs_term : rhs)
00659     {
00660         const auto rhs_ist = rhs_term.first;
00661         for (auto& lhs_term : lhs)
00662         {
00663             const index_set_t lhs_ist = lhs_term.first;
00664             if ((lhs_ist | rhs_ist) == rhs_ist)
00665                 result += lhs_term * rhs_term;
00666         }
00667     }
00668 }
00669 return result;
00670 }
00671
00672 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00673 inline
00674 auto
00675 framed_multi<Scalar_T,LO,HI,Tune_P>::
00676 operator%=( const multivector_t& rhs) -> multivector_t&
00677 { return *this = *this % rhs; }
00678
00679 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00680 auto
00681 star(const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const framed_multi<Scalar_T,LO,HI,Tune_P>& rhs)
00682 -> Scalar_T
00683 {
00684     {
00685         auto result = Scalar_T(0);
00686         const auto small_star_large = lhs.size() < rhs.size();
00687         const auto* smallp =
00688             small_star_large
00689             ? &lhs
00690             : &rhs;
00691         const auto* largep =
00692             small_star_large
00693             ? &rhs
00694             : &lhs;
00695
00696         for (auto& small_term : *smallp)
00697         {
00698             const auto small_ist = small_term.first;
00699             const auto large_crd = (*largep)[small_ist];
00700             if (large_crd != Scalar_T(0))
00701                 result += small_ist.sign_of_square() * small_term.second * large_crd;
00702         }
00703         return result;
00704     }
00705
00706 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00707 auto
00708 framed_multi<Scalar_T,LO,HI,Tune_P>::
00709 operator/=( const Scalar_T& scr) -> multivector_t&
00710 { // Divide coordinates of all terms by scr
00711     using traits_t = numeric_traits<Scalar_T>;
00712
00713     if (traits_t::isNaN(scr))
00714         return *this = traits_t::NaN();
00715     if (traits_t::isInf(scr))
00716         if (this->isnan())
00717             *this = traits_t::NaN();
00718 }

```

```

00719         else
00720             this->clear();
00721         else
00722             for (auto& this_term : *this)
00723                 this_term.second /= scr;
00724         return *this;
00725     }
00726
00727     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00728     inline
00729     auto
00730     operator/ (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
00731     framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
00732     {
00733         using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00734         using traits_t = numeric_traits<Scalar_T>;
00735         using matrix_multi_t = typename multivector_t::matrix_multi_t;
00736
00737         if (rhs == Scalar_T(0))
00738             return traits_t::NaN();
00739
00740         const auto our_frame = lhs.frame() | rhs.frame();
00741         return matrix_multi_t(lhs, our_frame, true) / matrix_multi_t(rhs, our_frame, true);
00742     }
00743
00744     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00745     inline
00746     auto
00747     framed_multi<Scalar_T,LO,HI,Tune_P>::
00748     operator/= (const multivector_t& rhs) -> multivector_t&
00749     { return *this = *this / rhs; }
00750
00751     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00752     inline
00753     auto
00754     operator| (const framed_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
00755     framed_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
00756     {
00757         using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00758         using matrix_multi_t = typename multivector_t::matrix_multi_t;
00759
00760         return matrix_multi_t(rhs) * matrix_multi_t(lhs) / matrix_multi_t(rhs.involute());
00761     }
00762
00763     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00764     inline
00765     auto
00766     framed_multi<Scalar_T,LO,HI,Tune_P>::
00767     operator|= (const multivector_t& rhs) -> multivector_t&
00768     { return *this = *this | rhs; }
00769
00770     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00771     inline
00772     auto
00773     framed_multi<Scalar_T,LO,HI,Tune_P>::
00774     inv() const -> const multivector_t
00775     {
00776         auto result = matrix_multi_t(Scalar_T(1), this->frame());
00777         return result /= matrix_multi_t(*this);
00778     }
00779
00780     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00781     auto
00782     framed_multi<Scalar_T,LO,HI,Tune_P>::
00783     pow(int m) const -> const multivector_t
00784     { return glucat::pow(*this, m); }
00785
00786     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00787     auto
00788     framed_multi<Scalar_T,LO,HI,Tune_P>::
00789     outer_pow(int m) const -> const multivector_t
00790     {
00791         if (m < 0)
00792             throw error_t("outer_pow(int): negative exponent");
00793         auto result = multivector_t(Scalar_T(1));
00794         auto a = *this;
00795         for (;
00796             m != 0;
00797             m >= 1, a = a ^ a)
00798             if (m & 1)
00799                 result ^= a;
00800         return result;
00801     }
00802
00803     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00804     inline
00805     auto

```

```

00812     framed_multi<Scalar_T,LO,HI,Tune_P>::
00813     frame() const -> const index_set_t
00814     {
00815         auto result = index_set_t();
00816         for (auto& this_term : *this)
00817             result |= this_term.first;
00818         return result;
00819     }
00820
00822     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00823     inline
00824     auto
00825     framed_multi<Scalar_T,LO,HI,Tune_P>::
00826     grade() const -> index_t
00827     {
00828         auto result = index_t(0);
00829         for (auto& this_term : *this)
00830             result = std::max(result, this_term.first.count());
00831         return result;
00832     }
00833
00835     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00836     inline
00837     auto
00838     framed_multi<Scalar_T,LO,HI,Tune_P>::
00839     operator[] (const index_set_t ist) const -> Scalar_T
00840     {
00841         const auto& this_it = this->find(ist);
00842         if (this_it == this->end())
00843             return Scalar_T(0);
00844         else
00845             return this_it->second;
00846     }
00847
00849     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00850     auto
00851     framed_multi<Scalar_T,LO,HI,Tune_P>::
00852     operator() (index_t grade) const -> const multivector_t
00853     {
00854         if ((grade < 0) || (grade > HI-LO))
00855             return Scalar_T(0);
00856         else
00857         {
00858             auto result = multivector_t();
00859             for (auto& this_term : *this)
00860                 if (this_term.first.count() == grade)
00861                     result += this_term;
00862             return result;
00863         }
00864     }
00865
00867     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00868     inline
00869     auto
00870     framed_multi<Scalar_T,LO,HI,Tune_P>::
00871     scalar() const -> Scalar_T
00872     { return (*this)[index_set_t()]; }
00873
00875     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00876     inline
00877     auto
00878     framed_multi<Scalar_T,LO,HI,Tune_P>::
00879     pure() const -> const multivector_t
00880     { return *this - this->scalar(); }
00881
00883     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00884     auto
00885     framed_multi<Scalar_T,LO,HI,Tune_P>::
00886     even() const -> const multivector_t
00887     { // even part of x, sum of the pure(count) with even count
00888         auto result = multivector_t();
00889         for (auto& this_term : *this)
00890             if ((this_term.first.count() % 2) == 0)
00891                 result.insert(this_term);
00892         return result;
00893     }
00894
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
00909     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00910     auto
00911     framed_multi<Scalar_T,LO,HI,Tune_P>::
00912     vector_part() const -> const vector_t
00913     { return this->vector_part(this->frame(), true); }
00914
00916     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00917     auto
00918     framed_multi<Scalar_T,LO,HI,Tune_P>::
00919     vector_part(const index_set_t frm, const bool prechecked) const -> const vector_t
00920     {
00921         if (!prechecked && (this->frame() | frm) != frm)
00922             throw error_t("vector_part(frm): value is outside of requested frame");
00923         auto result = vector_t();
00924         result.reserve(frm.count());
00925         const auto frm_end = frm.max()+1;
00926         for (auto
00927             idx = frm.min();
00928             idx != frm_end;
00929             ++idx)
00930             // Frame may contain indices which do not correspond to a grade 1 term but
00931             // frame cannot omit any index corresponding to a grade 1 term
00932             if (frm[idx])
00933                 result.push_back((*this)[index_set_t(idx)]);
00934         return result;
00935     }
00936
00938     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00939     auto
00940     framed_multi<Scalar_T,LO,HI,Tune_P>::
00941     involute() const -> const multivector_t
00942     {
00943         auto result = *this;
00944         for (auto& result_term : result)
00945             { // for a k-vector u, involute(u) == (-1)^k * u
00946                 if ((result_term.first.count() % 2) == 1)
00947                     result_term.second *= Scalar_T(-1);
00948             }
00949         return result;
00950     }
00951
00953     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00954     auto
00955     framed_multi<Scalar_T,LO,HI,Tune_P>::
00956     reverse() const -> const multivector_t
00957     {
00958         auto result = *this;
00959         for (auto& result_term : result)
00960             // For a k-vector u, reverse(u) = { -u, k == 2,3 (mod 4)
00961             // { u, k == 0,1 (mod 4)
00962             switch (result_term.first.count() % 4)
00963             {
00964             case 2:
00965             case 3:
00966                 result_term.second *= Scalar_T(-1);
00967                 break;
00968             default:
00969                 break;
00970             }
00971         return result;
00972     }
00973
00975     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00976     auto
00977     framed_multi<Scalar_T,LO,HI,Tune_P>::
00978     conj() const -> const multivector_t
00979     {
00980         auto result = *this;
00981         for (auto& result_term : result)
00982             // For a k-vector u, conj(u) = { -u, k == 1,2 (mod 4)
00983             // { u, k == 0,3 (mod 4)
00984             switch (result_term.first.count() % 4)
00985             {
00986             case 1:
00987             case 2:
00988                 result_term.second *= Scalar_T(-1);
00989                 break;
00990             default:
00991                 break;
00992             }
00993         return result;
00994     }
00995
00997     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00998     auto

```

```

00999 framed_multi<Scalar_T,LO,HI,Tune_P>::
01000 quad() const -> Scalar_T
01001 {
01002     // scalar(conj(x)*x) = 2*quad(even(x)) - quad(x)
01003     // ref: old clical: quadfunction(p:pterm):pterm in file compmod.pas
01004     auto result = Scalar_T(0);
01005     for (auto& this_term : *this)
01006     {
01007         const auto sign =
01008             (this_term.first.count_neg() % 2)
01009             ? -Scalar_T(1)
01010             : Scalar_T(1);
01011         result += sign * (this_term.second) * (this_term.second);
01012     }
01013     return result;
01014 }
01015
01016 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01017 auto
01018 framed_multi<Scalar_T,LO,HI,Tune_P>::
01019 norm() const -> Scalar_T
01020 {
01021     using traits_t = numeric_traits<Scalar_T>;
01022
01023     auto result = Scalar_T(0);
01024     for (auto& this_term : *this)
01025     {
01026         const auto abs_crd = traits_t::abs(this_term.second);
01027         result += abs_crd * abs_crd;
01028     }
01029     return result;
01030 }
01031
01032 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01033 auto
01034 framed_multi<Scalar_T,LO,HI,Tune_P>::
01035 max_abs() const -> Scalar_T
01036 {
01037     using traits_t = numeric_traits<Scalar_T>;
01038
01039     auto result = Scalar_T(0);
01040     for (auto& this_term : *this)
01041     {
01042         const auto abs_crd = traits_t::abs(this_term.second);
01043         if (abs_crd > result)
01044             result = abs_crd;
01045     }
01046     return result;
01047 }
01048
01049 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01050 auto
01051 framed_multi<Scalar_T,LO,HI,Tune_P>::
01052 random(const index_set_t frm, Scalar_T fill) -> const multivector_t
01053 {
01054     using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
01055     using index_set_t = typename multivector_t::index_set_t;
01056     using term_t = typename multivector_t::term_t;
01057
01058     using random_generator_t = random_generator<Scalar_T>;
01059     auto& generator = random_generator_t::generator();
01060
01061     fill =
01062         (fill < Scalar_T(0))
01063         ? Scalar_T(0)
01064         : (fill > Scalar_T(1))
01065         ? Scalar_T(1)
01066         : fill;
01067
01068     const auto algebra_dim = set_value_t(1) << frm.count();
01069     using traits_t = numeric_traits<Scalar_T>;
01070     const auto mean_abs = traits_t::sqrt(Scalar_T(double(algebra_dim)));
01071     auto result = multivector_t();
01072     for (auto
01073         stv = set_value_t(0);
01074         stv != algebra_dim;
01075         ++stv)
01076     {
01077         if (generator.uniform() < fill)
01078         {
01079             const auto& result_crd = generator.normal() / mean_abs;
01080             result.insert(term_t(index_set_t(stv, frm, true), result_crd));
01081         }
01082     }
01083     return result;
01084 }
01085
01086 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01087 inline
01088 void

```

```

01090     framed_multi<Scalar_T,LO,HI,Tune_P>::
01091     write(const std::string& msg) const
01092     { std::cout << msg << std::endl << " " << (*this) << std::endl; }
01093
01094     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01095     inline
01096     void
01097     framed_multi<Scalar_T,LO,HI,Tune_P>::
01098     write(std::ofstream& ofile, const std::string& msg) const
01099     {
01100     {
01101         if (!ofile)
01102             throw error_t("write(ofile,msg): cannot write to output file");
01103         ofile << msg << std::endl << " " << (*this) << std::endl;
01104     }
01105
01106     template< typename Map_T,typename Sorted_Map_T >
01107     class sorted_range
01108     {
01109     public:
01110         using map_t = Map_T;
01111         using sorted_map_t = Sorted_Map_T;
01112         using sorted_iterator = typename Sorted_Map_T::const_iterator;
01113
01114         sorted_range (Sorted_Map_T &sorted_val, const Map_T& val)
01115         {
01116             for (auto& val_term : val)
01117                 sorted_val.insert(val_term);
01118             sorted_begin = sorted_val.begin();
01119             sorted_end = sorted_val.end();
01120         }
01121         sorted_iterator sorted_begin;
01122         sorted_iterator sorted_end;
01123     };
01124
01125     template< typename Sorted_Map_T >
01126     class sorted_range< Sorted_Map_T, Sorted_Map_T >
01127     {
01128     public:
01129         using map_t = Sorted_Map_T;
01130         using sorted_map_t = Sorted_Map_T;
01131         using sorted_iterator = typename Sorted_Map_T::const_iterator;
01132
01133         sorted_range (Sorted_Map_T &sorted_val, const Sorted_Map_T& val)
01134         : sorted_begin( val.begin() ),
01135           sorted_end( val.end() )
01136         { }
01137         sorted_iterator sorted_begin;
01138         sorted_iterator sorted_end;
01139     };
01140
01141     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01142     auto
01143     operator<< (std::ostream& os, const framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::ostream&
01144     {
01145         using limits_t = std::numeric_limits<Scalar_T>;
01146         if (val.empty())
01147             os << 0;
01148         else if (val.isnan())
01149             os << limits_t::quiet_NaN();
01150         else if (val.isinf())
01151         {
01152             const Scalar_T& inf = limits_t::infinity();
01153             os << (scalar(val) < 0.0 ? -inf : inf);
01154         }
01155         else
01156         {
01157             using traits_t = numeric_traits<Scalar_T>;
01158             using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
01159             Scalar_T truncation;
01160             switch (os.flags() & std::ios::floatfield)
01161             {
01162             case std::ios_base::scientific:
01163                 truncation = Scalar_T(1) / traits_t::pow(Scalar_T(10), int(os.precision()) + 1);
01164                 break;
01165             case std::ios_base::fixed:
01166                 truncation = Scalar_T(1) / (traits_t::pow(Scalar_T(10), int(os.precision())) *
01167                 val.max_abs());
01168                 break;
01169             case std::ios_base::fixed | std::ios_base::scientific:
01170                 truncation = multivector_t::default_truncation;
01171                 break;
01172             default:
01173                 truncation = Scalar_T(1) / traits_t::pow(Scalar_T(10), int(os.precision()));
01174                 break;
01175             }
01176             auto truncated_val = val.truncated(truncation);
01177             if (truncated_val.empty())
01178

```

```

01179         os << 0;
01180     else
01181     {
01182         using map_t = typename multivector_t::map_t;
01183         using sorted_map_t = typename multivector_t::sorted_map_t;
01184         auto sorted_val = sorted_map_t();
01185         const auto sorted_val_range = sorted_range< map_t, sorted_map_t >(sorted_val, truncated_val);
01186         auto sorted_it = sorted_val_range.sorted_begin();
01187         os << *sorted_it;
01188         for (++sorted_it;
01189             sorted_it != sorted_val_range.sorted_end;
01190             ++sorted_it)
01191         {
01192             const Scalar_T& scr = sorted_it->second;
01193             if (scr >= 0.0)
01194                 os << '+';
01195             os << *sorted_it;
01196         }
01197     }
01198 }
01199 return os;
01200 }
01201
01202 template< typename Scalar_T, const index_t LO, const index_t HI >
01203 auto
01204 operator<< (std::ostream& os, const std::pair< const index_set<LO,HI>, Scalar_T >& term) ->
01205 std::ostream&
01206 {
01207     const auto second_as_double = numeric_traits<Scalar_T>::to_double(term.second);
01208     const auto use_double =
01209         (os.precision() <= std::numeric_limits<double>::digits10) ||
01210         (term.second == Scalar_T(second_as_double));
01211     if (term.first.count() == 0)
01212         if (use_double)
01213             os << second_as_double;
01214         else
01215             os << term.second;
01216     else if (term.second == Scalar_T(-1))
01217     {
01218         os << '-';
01219         os << term.first;
01220     }
01221     else if (term.second != Scalar_T(1))
01222     {
01223         if (use_double)
01224         {
01225             auto tol = std::pow(10.0, -os.precision());
01226             if ( std::fabs(second_as_double + 1.0) < tol )
01227                 os << '-';
01228             else if ( std::fabs(second_as_double - 1.0) >= tol )
01229                 os << second_as_double;
01230         }
01231         else
01232             os << term.second;
01233         os << term.first;
01234     }
01235     else
01236         os << term.first;
01237     return os;
01238 }
01239
01240 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01241 auto
01242 operator<< (std::istream& s, framed_multi<Scalar_T,LO,HI,Tune_P> & val) -> std::istream&
01243 { // Input looks like 1.0-2.0{1,2}+3.2{3,4}.
01244     using multivector_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
01245     // Parsing variables.
01246     auto local_val = multivector_t();
01247     auto c = 0;
01248     // Parsing control variables.
01249     auto negative = false;
01250     auto expect_term = true;
01251     // The multivector may begin with '+' or '-'. Check for this.
01252     c = s.peek();
01253     if (s.good() && (c == int('+') || c == int('-')))
01254     { // A '-' here negates the following term.
01255         negative = (c == int('-'));
01256         // Consume the '+' or '-'.
01257         s.get();
01258     }
01259     while (s.good())
01260     { // Parse a term.
01261         // A term consists of an optional scalar, followed by an optional index set.
01262         // At least one of the two must be present.
01263         // Default coordinate is Scalar_T(1).
01264         auto coordinate = Scalar_T(1);
01265         // Default index set is empty.

```

```

01267     auto ist = index_set<LO,HI>();
01268     // First, check for an opening brace.
01269     c = s.peek();
01270     if (s.good())
01271     { // If the character is not an opening brace,
01272       // a coordinate value is expected here.
01273       if (c != int('{'))
01274       { // Try to read a coordinate value.
01275         double coordinate_as_double;
01276         s » coordinate_as_double;
01277         // Reading the coordinate may have resulted in an end of file condition.
01278         // This is not a failure.
01279         if (s)
01280           coordinate = Scalar_T(coordinate_as_double);
01281       }
01282     }
01283     else
01284     { // End of file here ends parsing while a term may still be expected.
01285       break;
01286     }
01287     // Coordinate is now Scalar_T(1) or a Scalar_T value.
01288     // Parse an optional index set.
01289     if (s.good())
01290     {
01291       c = s.peek();
01292       if (s.good() && c == int('{'))
01293       { // Try to read index set.
01294         s » ist;
01295       }
01296     }
01297     // Reading the term may have resulted in an end of file condition.
01298     // This is not a failure.
01299     if (s)
01300     {
01301       // Immediately after parsing a term, another term is not expected.
01302       expect_term = false;
01303       if (coordinate != Scalar_T(0))
01304       {
01305         // Add the term to the local multivector.
01306         coordinate =
01307             negative
01308             ? -coordinate
01309             : coordinate;
01310         using term_t = typename multivector_t::term_t;
01311         local_val += term_t(ist, coordinate);
01312       }
01313     }
01314     // Check if anything follows the current term.
01315     if (s.good())
01316     {
01317       c = s.peek();
01318       if (s.good())
01319       { // Only '+' and '-' are valid here.
01320         if (c == int('+') || c == int('-'))
01321         { // A '-' here negates the following term.
01322           negative = (c == int('-'));
01323           // Consume the '+' or '-'.
01324           s.get();
01325           // Immediately after '+' or '-',
01326           // expect another term.
01327           expect_term = true;
01328         }
01329         else
01330         { // Any other character here is a not failure,
01331           // but still ends the parsing of the multivector.
01332           break;
01333         }
01334       }
01335     }
01336     }
01337     // If a term is still expected, this is a failure.
01338     if (expect_term)
01339       s.clear(std::istream::failbit);
01340     // End of file is not a failure.
01341     if (s)
01342     { // The multivector has been successfully parsed.
01343       val = local_val;
01344     }
01345     return s;
01346   }
01347
01348   template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01349   auto
01350   framed_multi<Scalar_T,LO,HI,Tune_P>::
01351   nbr_terms () const -> unsigned long
01352   { return this->size(); }
01353
01354

```

```

01356 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01357 inline
01358 auto
01359 framed_multi<Scalar_T,LO,HI,Tune_P>::
01360 operator+=(const term_t& term) -> multivector_t&
01361 { // Do not insert terms with 0 coordinate
01362     if (term.second != Scalar_T(0))
01363     {
01364         const auto& this_it = this->find(term.first);
01365         if (this_it == this->end())
01366             this->insert(term);
01367         else if (this_it->second + term.second == Scalar_T(0))
01368             // Erase term if resulting coordinate is 0
01369             this->erase(this_it);
01370         else
01371             this_it->second += term.second;
01372     }
01373     return *this;
01374 }
01375
01376 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01377 auto
01378 framed_multi<Scalar_T,LO,HI,Tune_P>::
01379 isinf() const -> bool
01380 {
01381     using traits_t = numeric_traits<Scalar_T>;
01382     if (std::numeric_limits<Scalar_T>::has_infinity)
01383         for (auto& this_term : *this)
01384             if (traits_t::isInf(this_term.second))
01385                 return true;
01386     return false;
01387 }
01388
01389 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01390 auto
01391 framed_multi<Scalar_T,LO,HI,Tune_P>::
01392 isnan() const -> bool
01393 {
01394     using traits_t = numeric_traits<Scalar_T>;
01395     if (std::numeric_limits<Scalar_T>::has_quiet_NaN)
01396         for (auto& this_term : *this)
01397             if (traits_t::isNaN(this_term.second))
01398                 return true;
01399     return false;
01400 }
01401
01402 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01403 auto
01404 framed_multi<Scalar_T,LO,HI,Tune_P>::
01405 truncated(const Scalar_T& limit) const -> const multivector_t
01406 {
01407     using traits_t = numeric_traits<Scalar_T>;
01408     if (this->isinf() || this->isinf())
01409         return *this;
01410     const auto truncation = traits_t::abs(limit);
01411     const auto top = max_abs();
01412     auto result = multivector_t();
01413     if (top != Scalar_T(0))
01414         for (auto& this_term : *this)
01415             if (traits_t::abs(this_term.second) > top * truncation)
01416                 result.insert(this_term);
01417     return result;
01418 }
01419
01420 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01421 auto
01422 framed_multi<Scalar_T,LO,HI,Tune_P>::
01423 fold(const index_set_t frm) const -> multivector_t
01424 {
01425     if (frm.is_contiguous())
01426         return *this;
01427     else
01428     {
01429         auto result = multivector_t();
01430         for (auto& this_term : *this)
01431             result.insert(term_t(this_term.first.fold(frm), this_term.second));
01432     }
01433     return result;
01434 }
01435
01436 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01437 auto
01438 framed_multi<Scalar_T,LO,HI,Tune_P>::
01439 unfold(const index_set_t frm) const -> multivector_t

```

```

01448 {
01449     if (frm.is_contiguous())
01450         return *this;
01451     else
01452     {
01453         auto result = multivector_t();
01454         for (auto& this_term : *this)
01455             result.insert(term_t(this_term.first.unfold(frm), this_term.second));
01456         return result;
01457     }
01458 }
01459
01461 // Reference: [L] 16.4 Periodicity of 8, p216
01462 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01463 auto
01464 framed_multi<Scalar_T,LO,HI,Tune_P>::
01465 centre_pm4_qp4(index_t& p, index_t& q) -> multivector_t&
01466 {
01467     // We add 4 to q by subtracting 4 from p
01468     if (q+4 > -LO)
01469         throw error_t("centre_pm4_qp4(p,q): LO is too high to represent this value");
01470     if (this->frame().max() > p-4)
01471     {
01472         using index_pair_t = typename index_set_t::index_pair_t;
01473         const auto pm3210 = index_set_t(index_pair_t(p-3,p), true);
01474         const auto qm4321 = index_set_t(index_pair_t(-q-4,-q-1), true);
01475         const auto& tqm4321 = term_t(qm4321, Scalar_T(1));
01476         auto result = multivector_t();
01477         for (auto& this_term : *this)
01478         {
01479             const auto ist = this_term.first;
01480             if (ist.max() > p-4)
01481             {
01482                 auto var_term = var_term_t();
01483                 for (auto
01484                     n = index_t(0);
01485                     n != index_t(4);
01486                     ++n)
01487                     if (ist[n+p-3])
01488                         var_term *= term_t(index_set_t(n-q-4), Scalar_T(1)) * tqm4321;
01489                 // Mask out {p-3}..{p}
01490                 result.insert(term_t(ist & ~pm3210, this_term.second) *
01491                             term_t(var_term.first, var_term.second));
01492             }
01493             else
01494                 result.insert(this_term);
01495         }
01496         *this = result;
01497     }
01498     p -=4; q += 4;
01499     return *this;
01500 }
01501
01503 // Reference: [L] 16.4 Periodicity of 8, p216
01504 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01505 auto
01506 framed_multi<Scalar_T,LO,HI,Tune_P>::
01507 centre_pp4_qm4(index_t& p, index_t& q) -> multivector_t&
01508 {
01509     // We add 4 to p by subtracting 4 from q
01510     if (p+4 > HI)
01511         throw error_t("centre_pp4_qm4(p,q): HI is too low to represent this value");
01512     if (this->frame().min() < -q+4)
01513     {
01514         using index_pair_t = typename index_set_t::index_pair_t;
01515         const auto qp0123 = index_set_t(index_pair_t(-q,-q+3), true);
01516         const auto pp1234 = index_set_t(index_pair_t(p+1,p+4), true);
01517         const auto& tpp1234 = term_t(pp1234, Scalar_T(1));
01518         auto result = multivector_t();
01519         for (auto& this_term : *this)
01520         {
01521             index_set_t ist = this_term.first;
01522             if (ist.min() < -q+4)
01523             {
01524                 auto var_term = var_term_t();
01525                 for (auto
01526                     n = index_t(0);
01527                     n != index_t(4);
01528                     ++n)
01529                     if (ist[n-q])
01530                         var_term *= term_t(index_set_t(n+p+1), Scalar_T(1)) * tpp1234;
01531                 // Mask out {-q}..{-q+3}
01532                 result.insert(term_t(var_term.first, var_term.second) *
01533                             term_t(ist & ~qp0123, this_term.second));
01534             }
01535             else
01536                 result.insert(this_term);

```

```

01537     }
01538     *this = result;
01539 }
01540 p +=4; q -= 4;
01541 return *this;
01542 }
01543
01544 // Reference: [P] Proposition 15.20, p 131
01545 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01546 auto
01547 framed_multi<Scalar_T,LO,HI,Tune_P>::
01548 centre_qpl_pml(index_t& p, index_t& q) -> multivector_t&
01549 {
01550     {
01551         if (q+1 > HI)
01552             throw error_t("centre_qpl_pml(p,q): HI is too low to represent this value");
01553         if (p-1 > -LO)
01554             throw error_t("centre_qpl_pml(p,q): LO is too high to represent this value");
01555         const auto qpl = index_set_t(q+1);
01556         const auto& tqpl = term_t(qpl, Scalar_T(1));
01557         auto result = multivector_t();
01558         for (auto& this_term : *this)
01559         {
01560             const auto ist = this_term.first;
01561             auto var_term = var_term_t(index_set_t(), this_term.second);
01562             for (auto
01563                 n = -q;
01564                 n != p;
01565                 ++n)
01566                 if (n != 0 && ist[n])
01567                     var_term *= term_t(index_set_t(-n) | qpl, Scalar_T(1));
01568             if (p != 0 && ist[p])
01569                 var_term *= tqpl;
01570             result.insert(term_t(var_term.first, var_term.second));
01571         }
01572         index_t orig_p = p;
01573         p = q+1;
01574         q = orig_p-1;
01575         return *this = result;
01576     }
01577
01578 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01579 auto
01580 framed_multi<Scalar_T,LO,HI,Tune_P>::
01581 divide(const index_set_t ist) const -> const framed_pair_t
01582 {
01583     {
01584         auto quo = multivector_t();
01585         auto rem = multivector_t();
01586         for (auto& this_term : *this)
01587             if ((this_term.first | ist) == this_term.first)
01588                 quo.insert(term_t(this_term.first ^ ist, this_term.second));
01589             else
01590                 rem.insert(this_term);
01591         return framed_pair_t(quo, rem);
01592     }
01593
01594 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01595 auto
01596 framed_multi<Scalar_T,LO,HI,Tune_P>::
01597 fast(const index_t level, const bool odd) const -> const matrix_t
01598 {
01599     {
01600         // Assume val is already folded and centred
01601         if (this->empty())
01602             {
01603                 using matrix_index_t = typename matrix_multi_t::matrix_index_t;
01604                 const auto dim = matrix_index_t(1) << level;
01605                 auto result = matrix_t(dim, dim);
01606                 result.clear();
01607                 return result;
01608             }
01609         if (level == 0)
01610             return matrix::unit<matrix_t>(1) * this->scalar();
01611
01612         using basis_matrix_t = typename matrix_multi_t::basis_matrix_t;
01613         using basis_scalar_t = typename basis_matrix_t::value_type;
01614
01615         const auto& I = matrix::unit<basis_matrix_t>(2);
01616         auto J = basis_matrix_t(2,2,2);
01617         J.clear();
01618         J(0,1) = basis_scalar_t(-1);
01619         J(1,0) = basis_scalar_t( 1);
01620         auto K = J;
01621         K(0,1) = basis_scalar_t( 1);
01622         auto JK = I;
01623         JK(0,0) = basis_scalar_t(-1);
01624
01625         const auto ist_mn = index_set_t(-level);
01626         const auto ist_pn = index_set_t(level);

```



```

01627     if (level == 1)
01628     {
01629         if (odd)
01630             return matrix_t(J) * (*this)[ist_mn] + matrix_t(K) * (*this)[ist_pn];
01631         else
01632             return matrix_t(I) * this->scalar() + matrix_t(JK) * (*this)[ist_mn ^ ist_pn];
01633     }
01634     else
01635     {
01636         const auto& pair_mn = this->divide(ist_mn);
01637         const auto& quo_mn = pair_mn.first;
01638         const auto& rem_mn = pair_mn.second;
01639         const auto& pair_quo_mnpn = quo_mn.divide(ist_pn);
01640         const auto& val_mnpn = pair_quo_mnpn.first;
01641         const auto& val_mn = pair_quo_mnpn.second;
01642         const auto& pair_rem_mnpn = rem_mn.divide(ist_pn);
01643         const auto& val_pn = pair_rem_mnpn.first;
01644         const auto& val_l = pair_rem_mnpn.second;
01645         using matrix::kron;
01646         if (odd)
01647             return - kron(JK, val_l.fast (level-1, 1))
01648                 + kron(I, val_mnpn.fast (level-1, 1))
01649                 + kron(J, val_mn.fast (level-1, 0))
01650                 + kron(K, val_pn.fast (level-1, 0));
01651         else
01652             return kron(I, val_l.fast (level-1, 0))
01653                 + kron(JK, val_mnpn.fast (level-1, 0))
01654                 + kron(K, val_mn.fast (level-1, 1))
01655                 - kron(J, val_pn.fast (level-1, 1));
01656     }
01657 }
01658
01660 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01661 template< typename Other_Scalar_T >
01662 auto
01663 framed_multi<Scalar_T,LO,HI,Tune_P>::
01664 fast_matrix_multi(const index_set_t frm) const -> const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>
01665 {
01666     // Fold val
01667     auto val = this->fold(frm);
01668     auto p = frm.count_pos();
01669     auto q = frm.count_neg();
01670     const auto bott_offset = gen::offset_to_super[pos_mod(p - q, 8)];
01671     p += std::max(bott_offset, index_t(0));
01672     q -= std::min(bott_offset, index_t(0));
01673     if (p > HI)
01674         throw error_t("fast_matrix_multi(frm): HI is too low to represent this value");
01675     if (q > -LO)
01676         throw error_t("fast_matrix_multi(frm): LO is too high to represent this value");
01677     // Centre val
01678     while (p - q > 4)
01679         val.centre_pm4_qp4(p, q);
01680     while (p - q < -3)
01681         val.centre_pp4_qm4(p, q);
01682     if (p - q > 1)
01683         val.centre_qp1_pm1(p, q);
01684     const index_t level = (p + q)/2;
01685
01686     // Do the fast transform
01687     const auto& ev_val = val.even();
01688     const auto& od_val = val.odd();
01689     return matrix_multi<Other_Scalar_T,LO,HI,Tune_P>(ev_val.fast(level, 0) + od_val.fast(level, 1),
01690 frm);
01691 }
01692
01692 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01693 inline
01694 auto
01695 framed_multi<Scalar_T,LO,HI,Tune_P>::
01696 fast_framed_multi() const -> const multivector_t
01697 { return *this; }
01698
01699 template< typename Scalar_T, const index_t LO, const index_t HI >
01700 inline
01701 static
01702 auto
01703 crd_of_mult(const std::pair<const index_set<LO,HI>, Scalar_T>& lhs,
01704             const std::pair<const index_set<LO,HI>, Scalar_T>& rhs) -> Scalar_T
01705 { return lhs.first.sign_of_mult(rhs.first) * lhs.second * rhs.second; }
01706
01707 template< typename Scalar_T, const index_t LO, const index_t HI >
01708 inline
01709 auto
01710 operator* (const std::pair<const index_set<LO,HI>, Scalar_T>& lhs,
01711           const std::pair<const index_set<LO,HI>, Scalar_T>& rhs) -> const std::pair<const
index_set<LO,HI>, Scalar_T>
01712 {

```

```

01715     using term_t = std::pair<const index_set<LO,HI>, Scalar_T>;
01716     return term_t(lhs.first ^ rhs.first, crd_of_mult(lhs, rhs));
01717 }
01718
01720 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01721 auto
01722 sqrt(const framed_multi<Scalar_T,LO,HI,Tune_P>& val, const framed_multi<Scalar_T,LO,HI,Tune_P>& i,
01723 bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
01724 {
01725     using traits_t = numeric_traits<Scalar_T>;
01726     if (val.isnan())
01727         return traits_t::NaN();
01728     check_complex(val, i, prechecked);
01729
01730     const auto realval = val.scalar();
01731     if (val == realval)
01732     {
01733         if (realval < Scalar_T(0))
01734             return i * traits_t::sqrt(-realval);
01735         else
01736             return traits_t::sqrt(realval);
01737     }
01738     using matrix_multi_t = typename framed_multi<Scalar_T,LO,HI,Tune_P>::matrix_multi_t;
01739     return sqrt(matrix_multi_t(val), matrix_multi_t(i), prechecked);
01740 }
01741
01743 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01744 auto
01745 exp(const framed_multi<Scalar_T,LO,HI,Tune_P>& val) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
01746 {
01747     using traits_t = numeric_traits<Scalar_T>;
01748     if (val.isnan())
01749         return traits_t::NaN();
01750
01751     const auto s = scalar(val);
01752     if (val == s)
01753         return traits_t::exp(s);
01754
01755     const double size = val.size();
01756     const auto frm_count = val.frame().count();
01757     const auto algebra_dim = set_value_t(1) << frm_count;
01758
01759     if ( (size * size <= double(algebra_dim)) || (frm_count < Tune_P::mult_matrix_threshold) )
01760     {
01761         switch (Tune_P::function_precision)
01762         {
01763             case precision_demoted:
01764             {
01765                 using demoted_scalar_t = typename traits_t::demoted::type;
01766                 using demoted_multivector_t = framed_multi<demoted_scalar_t,LO,HI,Tune_P>;
01767
01768                 const auto& demoted_val = demoted_multivector_t(val);
01769                 return clifford_exp(demoted_val);
01770             }
01771             break;
01772             case precision_promoted:
01773             {
01774                 using promoted_scalar_t = typename traits_t::promoted::type;
01775                 using promoted_multivector_t = framed_multi<promoted_scalar_t,LO,HI,Tune_P>;
01776
01777                 const auto& promoted_val = promoted_multivector_t(val);
01778                 return clifford_exp(promoted_val);
01779             }
01780             break;
01781             default:
01782                 return clifford_exp(val);
01783         }
01784     }
01785     else
01786     {
01787         using matrix_multi_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
01788         return exp(matrix_multi_t(val));
01789     }
01790 }
01791
01793 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01794 auto
01795 log(const framed_multi<Scalar_T,LO,HI,Tune_P>& val, const framed_multi<Scalar_T,LO,HI,Tune_P>& i,
01796 bool prechecked) -> const framed_multi<Scalar_T,LO,HI,Tune_P>
01797 {
01798     using traits_t = numeric_traits<Scalar_T>;
01799     if (val == Scalar_T(0) || val.isnan())
01800         return traits_t::NaN();
01801     check_complex(val, i, prechecked);
01802 }

```

```

01803     const auto realval = val.scalar();
01804     if (val == realval)
01805     {
01806         if (realval < Scalar_T(0))
01807             return i * traits_t::pi() + traits_t::log(-realval);
01808         else
01809             return traits_t::log(realval);
01810     }
01811     using matrix_multi_t = typename framed_multi<Scalar_T,I,O,HI,Tune_P>::matrix_multi_t;
01812     return log(matrix_multi_t(val), matrix_multi_t(i), prechecked);
01813 }
01814 }
01815 #endif // _GLUCAT_FRAMED_MULTI_IMP_H

```

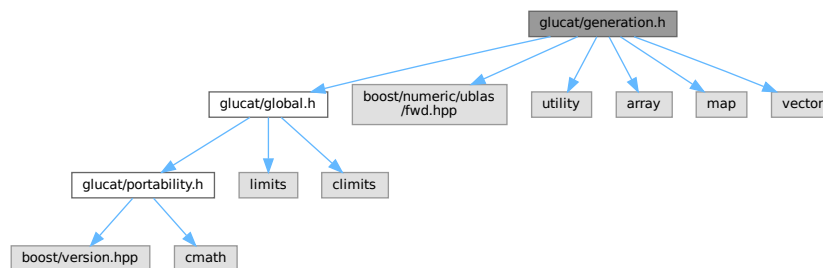
## 7.13 glucat/generation.h File Reference

```

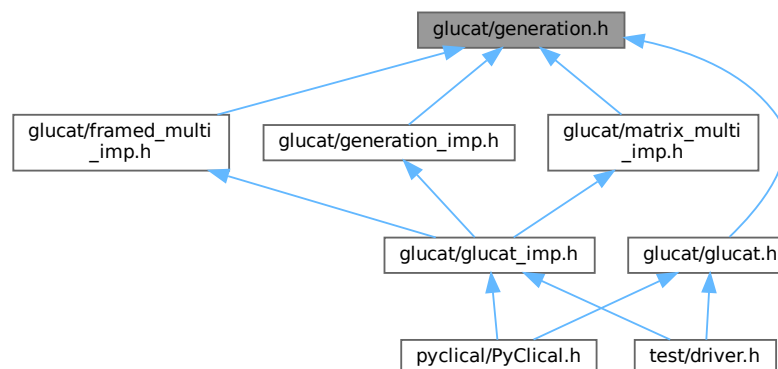
#include "glucat/global.h"
#include <boost/numeric/ublas/fwd.hpp>
#include <utility>
#include <array>
#include <map>
#include <vector>

```

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
00048     using signature_t = std::pair<index_t, index_t>;
00049
00051     template< class Matrix_T >
00052     class generator_table :
00053     private std::map< signature_t, std::vector<Matrix_T> >
00054     {
00055     public:
00057         auto operator() (const index_t p, const index_t q) -> const Matrix_T*;
00059         static auto generator() -> generator_table<Matrix_T>&;
00060     private:
00062         auto gen_vector(const index_t p, const index_t q) -> const std::vector<Matrix_T>&;
00064         void gen_from_pml_qml(const std::vector<Matrix_T>& old, const signature_t sig);
00066         void gen_from_pm4_qp4(const std::vector<Matrix_T>& old, const signature_t sig);
00068         void gen_from_pp4_qm4(const std::vector<Matrix_T>& old, const signature_t sig);
00070         void gen_from_qp1_pml(const std::vector<Matrix_T>& old, const signature_t sig);
00071
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 } }
00089 #endif // _GLUCAT_GENERATION_H

```

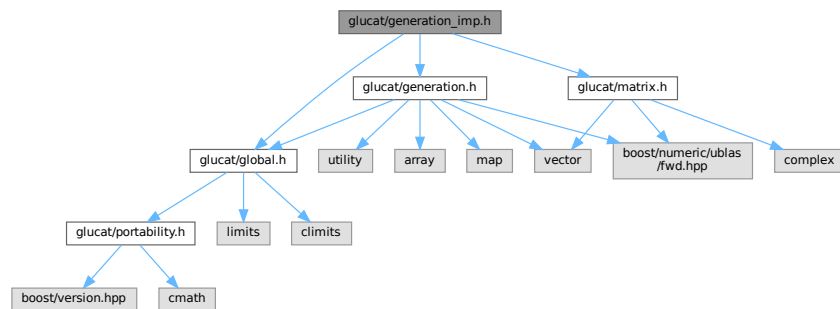
## 7.15 glucat/generation\_imp.h File Reference

```

#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
00046     // Reference: [M] Item 47
00047     template< class Matrix_T >
00048     auto
00049     generator_table<Matrix_T>::
00050     generator() -> generator_table<Matrix_T>&
00051     { static generator_table<Matrix_T> g; return g;}
00052
00054     // Reference: [P] Table 15.27, p 133
00055     template< class Matrix_T >
00056     inline
00057     auto
00058     generator_table<Matrix_T>::
00059     operator() (const index_t p, const index_t q) -> const Matrix_T*
00060     {
00061         const auto bott = pos_mod(p-q, 8);
00062         switch(bott)
00063         {
00064             case 0:
00065             case 2:
00066                 // Construct generators
00067                 return &(gen_vector(p, q)[q]);
00068             default:
00069                 // Select generators from the vector for a larger frame
00070                 const auto super_p = p + std::max(offset_to_super[bott], index_t(0));
00071                 const auto super_q = q - std::min(offset_to_super[bott], index_t(0));
00072                 return &(gen_vector(super_p, super_q)[super_q]);
00073         }
00074     }
00075
00077     template< class Matrix_T >
00078     auto
00079     generator_table<Matrix_T>::
00080     gen_vector(const index_t p, const index_t q) -> const std::vector<Matrix_T>&
00081     {
00082         using result_t = std::vector<Matrix_T>;
00083         const auto card = p + q;
00084         const auto bias = p - q;
00085         const auto bott = pos_mod(bias, 8);
00086         const auto sig = signature_t(p, q);
00087         if (this->find(sig) == this->end())
00088             switch(bott)
00089             {
00090                 case 0:
00091                     if (bias < 0)
00092                         // Construct generators for p,q given generators for p+4,q-4
00093                         gen_from_pp4_qm4(gen_vector(p+4, q-4), sig);
00094                     else if (bias > 0)
00095                         // Construct generators for p,q given generators for p-4,q+4
00096                         gen_from_pm4_qp4(gen_vector(p-4, q+4), sig);
00097                     else if (card == 0)
00098                     { // Base case. Save a generator vector containing one matrix, size 1.
00099                         auto result = result_t(1, matrix::unit<Matrix_T>(1));
00100                         this->insert(make_pair(sig, result));
00101                     }
00102                     else
00103                         // Construct generators for p,q given generators for p-1,q-1
00104                         gen_from_pml_qml(gen_vector(p-1, q-1), sig);
00105                     break;
00106                 case 2:
00107                     if (bias < 2)
00108                         // Construct generators for p,q given generators for p+4,q-4
00109                         gen_from_pp4_qm4(gen_vector(p+4, q-4), sig);
00110                     else if (bias > 2)
00111                         // Construct generators for p,q given generators for p-4,q+4
00112                         gen_from_pm4_qp4(gen_vector(p-4, q+4), sig);
00113                     else
00114                         // Construct generators for p,q given generators for q+1,p-1
00115                         gen_from_qp1_pml(gen_vector(q+1, p-1), sig);
00116                     break;
00117             default:
00118                 break;
00119             }
00120         return (*this)[sig];
00121     }
00122
00124     // Reference: [P] Proposition 15.17, p 131
00125     template< class Matrix_T >
00126     void
00127     generator_table<Matrix_T>::
00128     gen_from_pml_qml(const std::vector<Matrix_T>& old, const signature_t sig)

```

```

00129 {
00130     const auto new_size = old.size() + 2;
00131     using size_t = decltype(new_size);
00132     using result_t = std::vector<Matrix_T>;
00133     auto result = result_t(new_size);
00134
00135     const auto old_dim = old[0].size1();
00136     const auto& eye = matrix::unit<Matrix_T>(old_dim);
00137
00138     auto neg = Matrix_T(2,2,2);
00139     neg(0,1) = -1;
00140     neg(1,0) = 1;
00141
00142     auto pos = neg;
00143     pos(0,1) = 1;
00144
00145     auto dup = Matrix_T(2,2,2);
00146     dup(0,0) = 1;
00147     dup(1,1) = -1;
00148
00149     result[0] = matrix::mono_kron(neg, eye);
00150     for (auto
00151         k = size_t(1);
00152         k != new_size-1;
00153         ++k)
00154         result[k] = matrix::mono_kron(dup, old[k-1]);
00155     result[new_size-1] = matrix::mono_kron(pos, eye);
00156
00157     // Save the resulting generator array.
00158     this->insert(make_pair(sig, result));
00159 }
00160
00162 // Reference: [L] 16.4 Periodicity of 8, p216
00163 template< class Matrix_T >
00164 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
00195 // Reference: [L] 16.4 Periodicity of 8, p216
00196 template< class Matrix_T >
00197 void
00198 generator_table<Matrix_T>::
00199 gen_from_pp4_qm4(const std::vector<Matrix_T>& old, const signature_t sig)
00200 {
00201     const auto old_size = old.size();
00202     using size_t = decltype(old_size);
00203     using result_t = std::vector<Matrix_T>;
00204     auto result = result_t(old_size);
00205
00206     auto h = old[old_size-1];
00207     for (auto
00208         k = size_t(1);
00209         k != size_t(4);
00210         ++k)
00211         h = matrix::mono_prod(old[old_size-1-k], h);
00212
00213     for (auto
00214         k = size_t(0);
00215         k != size_t(4);
00216         ++k)
00217         result[k] = matrix::mono_prod(old[k+old_size-4], h);

```



```

00218     for (auto
00219         k = size_t(4);
00220         k != old_size;
00221         ++k)
00222         result[k] = old[k-4];
00223     // Save the resulting generator array.
00224     this->insert(make_pair(sig, result));
00225 }
00226
00228 // Reference: [P] Proposition 15.20, p 131
00229 template< class Matrix_T >
00230 void
00231 generator_table<Matrix_T>::
00232 gen_from_gpl_pml(const std::vector<Matrix_T>& old, const signature_t sig)
00233 {
00234     const auto old_size = old.size();
00235     using size_t = decltype(old_size);
00236     using result_t = std::vector<Matrix_T>;
00237     auto result = result_t(old_size);
00238
00239     const auto& h = old[old_size-1];
00240     for (auto
00241         k = size_t(0);
00242         k != old_size-1;
00243         ++k)
00244         result[k] = matrix::mono_prod(old[old_size-2-k], h);
00245     result[old_size-1] = h;
00246
00247     // Save the resulting generator array.
00248     this->insert(make_pair(sig, result));
00249 }
00250
00251 } }
00252 #endif // _GLUCAT_GENERATION_IMP_H

```

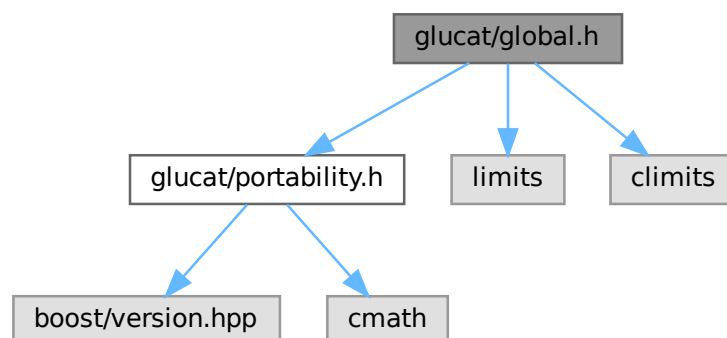
## 7.17 glucat/global.h File Reference

```

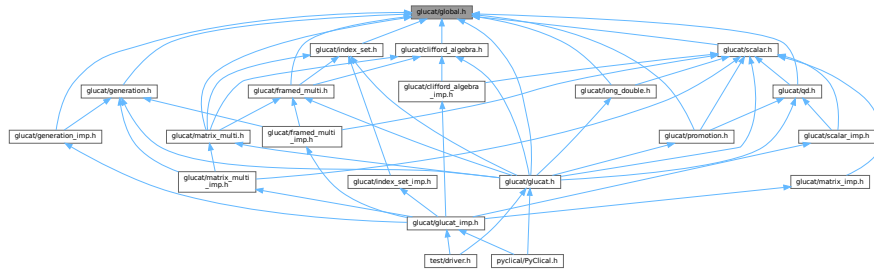
#include "glucat/portability.h"
#include <limits>
#include <climits>

```

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
00045 // Reference: [AA], p. 25
00046 template<bool> struct CTAAssertion;
00047 template<> struct CTAAssertion<true> { };
00048 #define _GLUCAT_CTAssert(expr, msg) \
00049     namespace { struct msg { glucat::CTAssertion<(expr)> ERROR_##msg; }; }
00050
00052 // Reference: [AA], pp. 34--37
00053 template < typename LHS_T, typename RHS_T >
00054 class compare_types
00055 {
00056 public:
00057     enum { are_same = false };
00058 };
00059 template < typename T >
00060 class compare_types<T, T>
00061 {
00062 public:
00063     enum { are_same = true };
00064 };
00065
00067 // Reference: [AA], 2.4, p. 29
00068 template< bool truth_value >
00069 class bool_to_type
00070 {
00071 private:
00072     enum { value = truth_value };
00073 };
00074
00075 // Global types which determine sizes
00077 using index_t = int;
00079 using set_value_t = unsigned long;
00080
00081 // Global constants
00083 const double MS_PER_S = 1000.0;
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

```

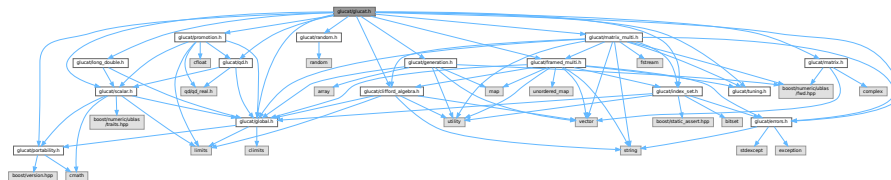
## 7.19 glucat/glucat.h File Reference

```

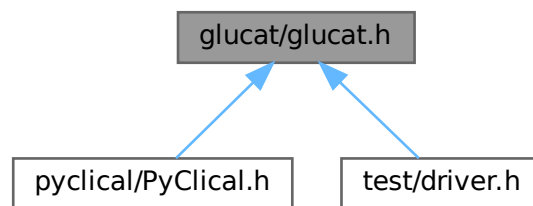
#include "glucat/portability.h"
#include "glucat/global.h"
#include "glucat/errors.h"
#include "glucat/index_set.h"
#include "glucat/scalar.h"
#include "glucat/long_double.h"

```

```
#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 0.13.0"`
- `#define GLUCAT_PACKAGE_TARNAME "glucat"`
- `#define GLUCAT_PACKAGE_URL ""`
- `#define GLUCAT_PACKAGE_VERSION "0.13.0"`
- `#define GLUCAT_STDC_HEADERS 1`
- `#define GLUCAT_VERSION "0.13.0"`

## 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 0.13.0"
```

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 "0.13.0"
```

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 "0.13.0"
```

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 0.13.0"
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 "0.13.0"
00110 #endif
00111
00112 /* Define to 1 if all of the C89 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 "0.13.0"
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&;

```

```

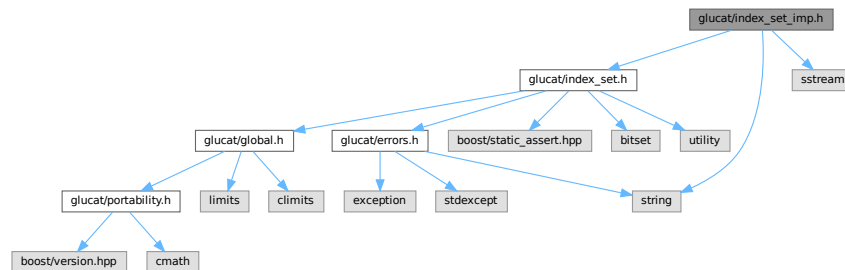
00127     auto reset() -> index_set_t&;
00129     auto reset(const index_t idx) -> index_set_t&;
00131     auto flip() -> index_set_t&;
00133     auto flip(const index_t idx) -> index_set_t&;
00135     auto count() const -> index_t;
00137     auto count_neg() const -> index_t;
00139     auto count_pos() 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

```

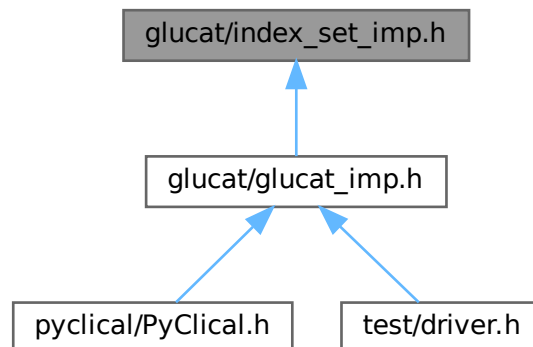


## 7.27 glucat/index\_set\_imp.h File Reference

```
#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     index_set<LO,HI>::
00159     operator^ (const index_set<LO,HI>& lhs,
00160                const index_set<LO,HI>& rhs) -> const
00161     index_set<LO,HI>
00162     {
00163         using index_set_t = index_set<LO, HI>;
00164         using bitset_t = typename index_set_t::bitset_t;
00165         return static_cast<bitset_t>(lhs) ^ static_cast<bitset_t>(rhs);
00166     }
00167
00168     template<const index_t LO, const index_t HI>
00169     inline
00170     auto
00171     index_set<LO,HI>::
00172     operator&= (const index_set_t rhs) -> index_set_t&
00173     {
00174         bitset_t* pthis = this;
00175         *pthis &= static_cast<bitset_t>(rhs);
00176         return *this;
00177     }
00178
00179     template<const index_t LO, const index_t HI>
00180     inline
00181     auto
00182     index_set<LO,HI>::
00183     operator& (const index_set<LO,HI>& lhs,
00184                const index_set<LO,HI>& rhs) -> const
00185     index_set<LO,HI>
00186     {
00187         using index_set_t = index_set<LO, HI>;
00188         using bitset_t = typename index_set_t::bitset_t;
00189         return static_cast<bitset_t>(lhs) & static_cast<bitset_t>(rhs);
00190     }
00191
00192     template<const index_t LO, const index_t HI>
00193     inline
00194     auto
00195     index_set<LO,HI>::
00196     operator|= (const index_set_t rhs) -> index_set_t&
00197     {
00198         bitset_t* pthis = this;
00199         *pthis |= static_cast<bitset_t>(rhs);
00200         return *this;
00201     }
00202
00203     template<const index_t LO, const index_t HI>
00204     inline
00205     auto
00206     index_set<LO,HI>::
00207     operator| (const index_set<LO,HI>& lhs,
00208                const index_set<LO,HI>& rhs) -> const
00209     index_set<LO,HI>
00210     {
00211         using index_set_t = index_set<LO, HI>;
00212         using bitset_t = typename index_set_t::bitset_t;
00213         return static_cast<bitset_t>(lhs) | static_cast<bitset_t>(rhs);
00214     }
00215
00216     template<const index_t LO, const index_t HI>
00217     inline
00218     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 & 0aaaaaaaaaaaaaul)
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         // 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
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 namespace glucat
00038 {
00039 #if defined(__USE_GNU)
00040     static const long double l_pi    = M_PI1;
00041     static const long double l_ln2  = M_LN21;
00042 #else
00043     static const long double l_pi    = 3.1415926535897932384626433832795029L;
00044     static const long double l_ln2  = 0.6931471805599453094172321214581766L;
00045 #endif
00046
00047     template<>
00048     inline
00049     auto
00050     numeric_traits<long double>::
00051     pi() -> long double
00052     { return l_pi; }
00053
00054     template<>
00055     inline
00056     auto
00057     numeric_traits<long double>::
00058     ln_2() -> long double
00059     { return l_ln2; }
00060 }
00061 #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:  $\text{sum}(x(i,j)*y(i,j))/x.\text{nrows}()$*
- 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 <blaze/Math.h>
#include <blaze/math/DynamicMatrix.h>
#include <blaze/math/DynamicVector.h>
#include <set>

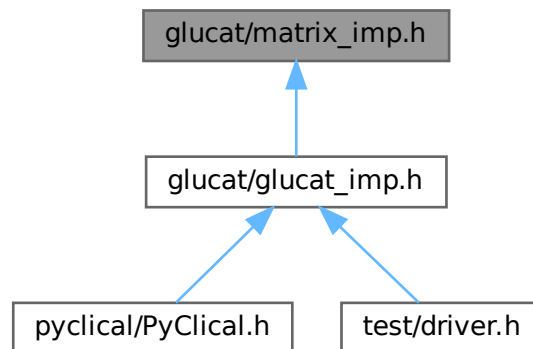
```

```
#include <vector>
```

Include dependency graph for matrix\_imp.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [glucat](#)
- namespace [glucat::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>`  
`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.*
- `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>  
static auto `glucat::matrix::to_blaze` (const Matrix\_T &val) -> blaze::DynamicMatrix< double, blaze::rowMajor >  
*Convert matrix to Blaze 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_BLAZE)
00056 #include <blaze/Math.h>
00057 #include <blaze/math/DynamicMatrix.h>
00058 #include <blaze/math/DynamicVector.h>
00059 #endif
00060
00061 # if defined(_GLUCAT_GCC_IGNORE_UNUSED_LOCAL_TYPEDEFS)
00062 # pragma GCC diagnostic pop
00063 # endif
00064
00065 #include <set>
00066 #include <vector>
00067
00068 namespace glucat { namespace matrix
00069 {
00070     // References for algorithms:
00071     // [v]: C. F. van Loan and N. Pitsianis, "Approximation with Kronecker products",
00072     // in Linear Algebra for Large Scale and Real-Time Applications, Marc S. Moonen,
00073     // Gene H. Golub, and Bart L. R. Moor (eds.), 1993, pp. 293--314.
00074
00075     template< typename LHS_T, typename RHS_T >
00076     auto
00077     00078     kron(const LHS_T& lhs, const RHS_T& rhs) -> const
00079     RHS_T
00080     {
00081         const auto rhs_s1 = rhs.size1();
00082         const auto rhs_s2 = rhs.size2();
00083         auto result = RHS_T(lhs.size1()*rhs_s1, lhs.size2()*rhs_s2);
00084         result.clear();
00085
00086         for (auto
00087             lhs_it1 = lhs.begin1();
00088             lhs_it1 != lhs.end1();
00089             ++lhs_it1)
00090             for (auto
00091                 lhs_it2 = lhs_it1.begin();
00092                 lhs_it2 != lhs_it1.end();
00093                 ++lhs_it2)
00094             {
00095                 const auto start1 = rhs_s1 * lhs_it2.index1();

```

```

00096         const auto start2 = rhs_s2 * lhs_it2.index2();
00097         const auto& lhs_val = *lhs_it2;
00098         for (auto
00099             rhs_it1 = rhs.begin1();
00100             rhs_it1 != rhs.end1();
00101             ++rhs_it1)
00102             for (auto
00103                 rhs_it2 = rhs_it1.begin();
00104                 rhs_it2 != rhs_it1.end();
00105                 ++rhs_it2)
00106                 result(start1 + rhs_it2.index1(), start2 + rhs_it2.index2()) = lhs_val * *rhs_it2;
00107     }
00108     return result;
00109 }
00110
00112 template< typename LHS_T, typename RHS_T >
00113 auto
00114 mono_kron(const LHS_T& lhs, const RHS_T& rhs) -> const
00115 RHS_T
00116 {
00117     const auto rhs_s1 = rhs.size1();
00118     const auto rhs_s2 = rhs.size2();
00119     const auto dim = lhs.size1()*rhs_s1;
00120     auto result = RHS_T(dim, dim, dim);
00121     result.clear();
00122
00123     for (auto
00124         lhs_it1 = lhs.begin1();
00125         lhs_it1 != lhs.end1();
00126         ++lhs_it1)
00127     {
00128         const auto lhs_it2 = lhs_it1.begin();
00129         const auto start1 = rhs_s1 * lhs_it2.index1();
00130         const auto start2 = rhs_s2 * lhs_it2.index2();
00131         const auto& lhs_val = *lhs_it2;
00132         for (auto
00133             rhs_it1 = rhs.begin1();
00134             rhs_it1 != rhs.end1();
00135             ++rhs_it1)
00136         {
00137             const auto rhs_it2 = rhs_it1.begin();
00138             result(start1 + rhs_it2.index1(), start2 + rhs_it2.index2()) = lhs_val * *rhs_it2;
00139         }
00140     }
00141     return result;
00142 }
00143
00145 template< typename LHS_T, typename RHS_T >
00146 void
00147 nork_range(RHS_T& result,
00148            const typename LHS_T::const_iterator2 lhs_it2,
00149            const RHS_T& rhs,
00150            const typename RHS_T::size_type res_s1,
00151            const typename RHS_T::size_type res_s2)
00152 {
00153     // Definition matches [v] Section 4, Theorem 4.1.
00154     const auto start1 = res_s1 * lhs_it2.index1();
00155     const auto start2 = res_s2 * lhs_it2.index2();
00156     using ublas::range;
00157     const auto& range1 = range(start1, start1 + res_s1);
00158     const auto& range2 = range(start2, start2 + res_s2);
00159     using matrix_range_t = ublas::matrix_range<const RHS_T>;
00160     const auto& rhs_range = matrix_range_t(rhs, range1, range2);
00161     using Scalar_T = typename RHS_T::value_type;
00162     const auto lhs_val = numeric_traits<Scalar_T>::to_scalar_t(*lhs_it2);
00163     for (auto
00164         rhs_it1 = rhs_range.begin1();
00165         rhs_it1 != rhs_range.end1();
00166         ++rhs_it1)
00167         for (auto
00168             rhs_it2 = rhs_it1.begin();
00169             rhs_it2 != rhs_it1.end();
00170             ++rhs_it2)
00171             result(rhs_it2.index1(), rhs_it2.index2()) += lhs_val * *rhs_it2;
00172 }
00173
00175 template< typename LHS_T, typename RHS_T >
00176 auto
00177 nork(const LHS_T& lhs, const RHS_T& rhs, const bool mono) -> const
00178 RHS_T
00179 {
00180     // nork(A, kron(A, B)) is close to B
00181     // Definition matches [v] Section 4, Theorem 4.1.
00182     const auto lhs_s1 = lhs.size1();
00183     const auto lhs_s2 = lhs.size2();
00184     const auto rhs_s1 = rhs.size1();
00185     const auto rhs_s2 = rhs.size2();

```

```

00186     const auto res_s1 = rhs_s1 / lhs_s1;
00187     const auto res_s2 = rhs_s2 / lhs_s2;
00188     using Scalar_T = typename RHS_T::value_type;
00189     const auto norm_frob2_lhs = norm_frob2(lhs);
00190     if (!mono)
00191     {
00192         using error_t = error<RHS_T>;
00193         if (rhs_s1 == 0)
00194             throw error_t("matrix", "nork: number of rows must not be 0");
00195         if (rhs_s2 == 0)
00196             throw error_t("matrix", "nork: number of cols must not be 0");
00197         if (res_s1 * lhs_s1 != rhs_s1)
00198             throw error_t("matrix", "nork: incompatible numbers of rows");
00199         if (res_s2 * lhs_s2 != rhs_s2)
00200             throw error_t("matrix", "nork: incompatible numbers of cols");
00201         if (norm_frob2_lhs == Scalar_T(0))
00202             throw error_t("matrix", "nork: LHS must not be 0");
00203     }
00204     auto result = RHS_T(res_s1, res_s2);
00205     result.clear();
00206     for (auto
00207         lhs_it1 = lhs.begin1();
00208         lhs_it1 != lhs.end1();
00209         ++lhs_it1)
00210     for (auto
00211         lhs_it2 = lhs_it1.begin();
00212         lhs_it2 != lhs_it1.end();
00213         ++lhs_it2)
00214         if (*lhs_it2 != Scalar_T(0))
00215             nork_range<LHS_T, RHS_T>(result, lhs_it2, rhs, res_s1, res_s2);
00216     result /= norm_frob2_lhs;
00217     return result;
00218 }
00219
00221 template< typename LHS_T, typename RHS_T >
00222 auto
00223 signed_perm_nork(const LHS_T& lhs, const RHS_T& rhs) -> const
00224 RHS_T
00225 {
00226     // signed_perm_nork(A, kron(A, B)) is close to B
00227     // Definition matches [v] Section 4, Theorem 4.1.
00228     const auto lhs_s1 = lhs.size1();
00229     const auto lhs_s2 = lhs.size2();
00230     const auto rhs_s1 = rhs.size1();
00231     const auto rhs_s2 = rhs.size2();
00232     const auto res_s1 = rhs_s1 / lhs_s1;
00233     const auto res_s2 = rhs_s2 / lhs_s2;
00234     using Scalar_T = typename RHS_T::value_type;
00235     const auto norm_frob2_lhs = Scalar_T( double(lhs_s1) );
00236     auto result = RHS_T(res_s1, res_s2);
00237     result.clear();
00238     for (auto
00239         lhs_it1 = lhs.begin1();
00240         lhs_it1 != lhs.end1();
00241         ++lhs_it1)
00242     {
00243         const auto lhs_it2 = lhs_it1.begin();
00244         nork_range<LHS_T, RHS_T>(result, lhs_it2, rhs, res_s1, res_s2);
00245     }
00246     result /= norm_frob2_lhs;
00247     return result;
00248 }
00249
00251 template< typename Matrix_T >
00252 auto
00253 nnz(const Matrix_T& m) -> typename Matrix_T::size_type
00254 {
00255     using size_t = typename Matrix_T::size_type;
00256     auto result = size_t(0);
00257     for (auto
00258         it1 = m.begin1();
00259         it1 != m.end1();
00260         ++it1)
00261     for (auto& entry : it1)
00262         if (entry != 0)
00263             ++result;
00264     return result;
00265 }
00266
00268 template< typename Matrix_T >
00269 auto
00270 isinf(const Matrix_T& m) -> bool
00271 {
00272     using Scalar_T = typename Matrix_T::value_type;
00273     for (auto
00274         it1 = m.begin1();
00275         it1 != m.end1();

```



```

00276         ++it1)
00277     for (auto& entry : it1)
00278     if (numeric_traits<Scalar_T>::isInf(entry))
00279         return true;
00280
00281     return false;
00282 }
00283
00284 template< typename Matrix_T >
00285 auto
00286 isnan(const Matrix_T& m) -> bool
00287 {
00288     using Scalar_T = typename Matrix_T::value_type;
00289     for (auto
00290         it1 = m.begin1();
00291         it1 != m.end1();
00292         ++it1)
00293     for (auto& entry : it1)
00294     if (numeric_traits<Scalar_T>::isNaN(entry))
00295         return true;
00296
00297     return false;
00298 }
00299
00300 template< typename Matrix_T >
00301 inline
00302 auto
00303 unit(const typename Matrix_T::size_type dim) -> const
00304 Matrix_T
00305 {
00306     using Scalar_T = typename Matrix_T::value_type;
00307     return ublas::identity_matrix<Scalar_T>(dim);
00308 }
00309
00310 template< typename LHS_T, typename RHS_T >
00311 auto
00312 mono_prod(const ublas::matrix_expression<LHS_T>& lhs,
00313           const ublas::matrix_expression<RHS_T>& rhs) -> const typename RHS_T::expression_type
00314 {
00315     using rhs_expression_t = const RHS_T;
00316     using matrix_row_t = typename ublas::matrix_row<rhs_expression_t>;
00317
00318     const auto dim = lhs().size1();
00319     // The following assumes that RHS_T is a sparse matrix type.
00320     auto result = RHS_T(dim, dim, dim);
00321     for (auto
00322         lhs_row = lhs().begin1();
00323         lhs_row != lhs().end1();
00324         ++lhs_row)
00325     {
00326         const auto& lhs_it = lhs_row.begin();
00327         if (lhs_it != lhs_row.end())
00328         {
00329             const auto& rhs_row = matrix_row_t(rhs(), lhs_it.index2());
00330             const auto& rhs_it = rhs_row.begin();
00331             if (rhs_it != rhs_row.end())
00332                 result(lhs_it.index1(), rhs_it.index()) = (*lhs_it) * (*rhs_it);
00333         }
00334     }
00335     return result;
00336 }
00337
00338 template< typename LHS_T, typename RHS_T >
00339 inline
00340 auto
00341 sparse_prod(const ublas::matrix_expression<LHS_T>& lhs,
00342            const ublas::matrix_expression<RHS_T>& rhs) -> const typename RHS_T::expression_type
00343 {
00344     using expression_t = typename RHS_T::expression_type;
00345     return ublas::sparse_prod<expression_t>(lhs(), rhs());
00346 }
00347
00348 template< typename LHS_T, typename RHS_T >
00349 inline
00350 auto
00351 prod(const ublas::matrix_expression<LHS_T>& lhs,
00352      const ublas::matrix_expression<RHS_T>& rhs) -> const typename RHS_T::expression_type
00353 {
00354     const auto dim = lhs().size1();
00355     RHS_T result(dim, dim);
00356     ublas::axpy_prod(lhs, rhs, result, true);
00357     return result;
00358 }
00359
00360 template< typename Scalar_T, typename LHS_T, typename RHS_T >
00361 auto
00362 inner(const LHS_T& lhs, const RHS_T& rhs) -> Scalar_T

```

```

00369 {
00370     auto result = Scalar_T(0);
00371     for (auto
00372         lhs_it1 = lhs.begin1();
00373         lhs_it1 != lhs.end1();
00374         ++lhs_it1)
00375     for (auto
00376         lhs_it2 = lhs_it1.begin();
00377         lhs_it2 != lhs_it1.end();
00378         ++lhs_it2)
00379     {
00380         const auto& rhs_val = rhs(lhs_it2.index1(), lhs_it2.index2());
00381         if (rhs_val != Scalar_T(0))
00382             result += (*lhs_it2) * rhs_val;
00383     }
00384     return result / lhs.size1();
00385 }
00386
00387 template< typename Matrix_T >
00388 auto
00389 norm_frob2(const Matrix_T& val) -> typename Matrix_T::value_type
00390 {
00391     using Scalar_T = typename Matrix_T::value_type;
00392
00393     auto result = Scalar_T(0);
00394     for (auto
00395         val_it1 = val.begin1();
00396         val_it1 != val.end1();
00397         ++val_it1)
00398     for (auto& val_entry : val_it1)
00399     {
00400         if (numeric_traits<Scalar_T>::isNaN(val_entry))
00401             return numeric_traits<Scalar_T>::NaN();
00402         result += val_entry * val_entry;
00403     }
00404     return result;
00405 }
00406
00407 template< typename Matrix_T >
00408 auto
00409 trace(const Matrix_T& val) -> typename Matrix_T::value_type
00410 {
00411     using Scalar_T = typename Matrix_T::value_type;
00412
00413     auto result = Scalar_T(0);
00414     auto dim = val.size1();
00415     for (auto
00416         ndx = decltype(dim)(0);
00417         ndx != dim;
00418         ++ndx)
00419     {
00420         const Scalar_T crd = val(ndx, ndx);
00421         if (numeric_traits<Scalar_T>::isNaN(crd))
00422             return numeric_traits<Scalar_T>::NaN();
00423         result += crd;
00424     }
00425     return result;
00426 }
00427
00428 #if defined(_GLUCAT_USE_BLAZE)
00429 template< typename Matrix_T >
00430 static
00431 auto
00432 to_blaze(const Matrix_T& val) -> blaze::DynamicMatrix<double, blaze::rowMajor>
00433 {
00434     const auto s1 = val.size1();
00435     const auto s2 = val.size2();
00436
00437     using blaze_matrix_t = typename blaze::DynamicMatrix<double, blaze::rowMajor>;
00438     auto result = blaze_matrix_t(s1, s2);
00439
00440     using Scalar_T = typename Matrix_T::value_type;
00441     using traits_t = numeric_traits<Scalar_T>;
00442
00443     for (auto
00444         val_it1 = val.begin1();
00445         val_it1 != val.end1();
00446         ++val_it1)
00447     for (auto
00448         val_it2 = val_it1.begin();
00449         val_it2 != val_it1.end();
00450         ++val_it2)
00451     result(val_it2.index1(), val_it2.index2()) = traits_t::to_double(*val_it2);
00452
00453     return result;
00454 }
00455
00456 }
00457
00458

```

```

00459 #endif
00460
00462 template< typename Matrix_T >
00463 auto
00464 eigenvalues(const Matrix_T& val) -> std::vector< std::complex<double> >
00465 {
00466     using complex_t = std::complex<double>;
00467     using complex_vector_t = typename std::vector<complex_t>;
00468
00469     const auto dim = val.size1();
00470     auto lambda = complex_vector_t(dim);
00471
00472 #if defined(_GLUCAT_USE_BLAZE)
00473     using complex_t = std::complex<double>;
00474     using blaze_complex_vector_t = blaze::DynamicVector<complex_t, blaze::columnVector>;
00475
00476     auto blaze_val = to_blaze(val);
00477     auto blaze_lambda = blaze_complex_vector_t(dim);
00478     blaze::geev(blaze_val, blaze_lambda);
00479
00480     for (auto
00481         k = decltype(dim)(0);
00482         k != dim;
00483         ++k)
00484         lambda[k] = blaze_lambda[k];
00485 #endif
00486     return lambda;
00487 }
00488
00490 template< typename Matrix_T >
00491 auto
00492 classify_eigenvalues(const Matrix_T& val) -> eig_genus<Matrix_T>
00493 {
00494     using Scalar_T = typename Matrix_T::value_type;
00495     eig_genus<Matrix_T> result;
00496
00497     auto lambda = eigenvalues(val);
00498
00499     std::set<double> arg_set;
00500
00501     const auto dim = lambda.size();
00502     static const auto epsilon =
00503         std::max(std::numeric_limits<double>::epsilon(),
00504                 numeric_traits<Scalar_T>::to_double(std::numeric_limits<Scalar_T>::epsilon()));
00505     static const auto zero_eig_tol = 4096.0*epsilon;
00506
00507     bool neg_real_eig_found = false;
00508     bool imag_eig_found = false;
00509     bool zero_eig_found = false;
00510
00511     for (auto
00512         k = decltype(dim)(0);
00513         k != dim;
00514         ++k)
00515     {
00516         const auto lambda_k = lambda[k];
00517         arg_set.insert(std::arg(lambda_k));
00518
00519         const auto real_lambda_k = std::real(lambda_k);
00520         const auto imag_lambda_k = std::imag(lambda_k);
00521         const auto norm_tol = 4096.0*epsilon*std::norm(lambda_k);
00522
00523         if (!neg_real_eig_found &&
00524             real_lambda_k < -epsilon &&
00525             (imag_lambda_k == 0.0 ||
00526              imag_lambda_k * imag_lambda_k < norm_tol))
00527             neg_real_eig_found = true;
00528         if (!imag_eig_found &&
00529             imag_lambda_k > epsilon &&
00530             (real_lambda_k == 0.0 ||
00531              real_lambda_k * real_lambda_k < norm_tol))
00532             imag_eig_found = true;
00533         if (!zero_eig_found &&
00534             std::norm(lambda_k) < zero_eig_tol)
00535             zero_eig_found = true;
00536     }
00537
00538     if (zero_eig_found)
00539         result.m_is_singular = true;
00540
00541     static const auto pi = numeric_traits<double>::pi();
00542     if (neg_real_eig_found)
00543     {
00544         if (imag_eig_found)
00545             result.m_eig_case = both_eigs;
00546         else
00547             {

```

```

00548         result.m_eig_case = neg_real_eigs;
00549         result.m_safe_arg = Scalar_T(-pi / 2.0);
00550     }
00551 }
00552
00553 if (result.m_eig_case == both_eigs)
00554 {
00555     auto arg_it = arg_set.begin();
00556     auto first_arg = *arg_it;
00557     auto best_arg = first_arg;
00558     auto best_diff = 0.0;
00559     auto previous_arg = first_arg;
00560     for (++arg_it;
00561         arg_it != arg_set.end();
00562         ++arg_it)
00563     {
00564         const auto arg_diff = *arg_it - previous_arg;
00565         if (arg_diff > best_diff)
00566         {
00567             best_diff = arg_diff;
00568             best_arg = previous_arg;
00569         }
00570         previous_arg = *arg_it;
00571     }
00572     const auto arg_diff = first_arg + 2.0*pi - previous_arg;
00573     if (arg_diff > best_diff)
00574     {
00575         best_diff = arg_diff;
00576         best_arg = previous_arg;
00577     }
00578     result.m_safe_arg = Scalar_T(pi - (best_arg + best_diff / 2.0));
00579 }
00580 return result;
00581 }
00582 } }
00583
00584 #endif // _GLUCAT_MATRIX_IMP_H

```

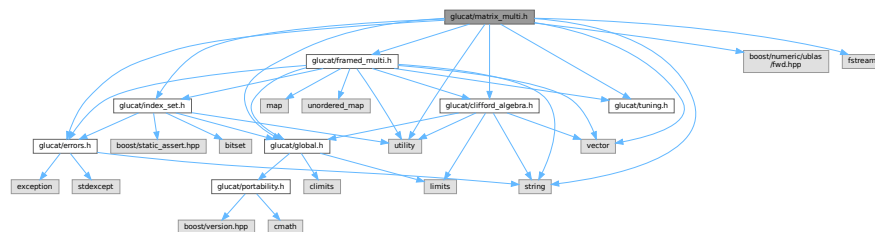
## 7.35 glucat/matrix\_multi.h File Reference

```

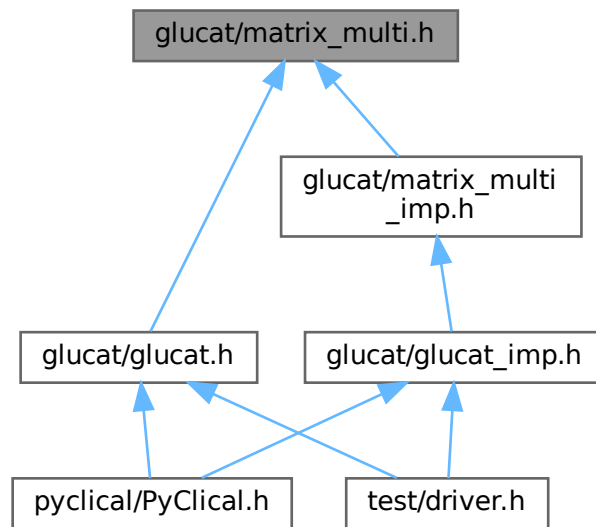
#include "glucat/global.h"
#include "glucat/errors.h"
#include "glucat/index_set.h"
#include "glucat/clifford_algebra.h"
#include "glucat/tuning.h"
#include "glucat/framed_multi.h"
#include <boost/numeric/ublas/fwd.hpp>
#include <fstream>
#include <string>
#include <utility>
#include <vector>

```

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
00065     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00066     auto
00067     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
00070     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00071     auto
00072     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
00075     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00076     auto
00077     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
00080     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00081     auto
00082     star(const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs)
-> Scalar_T;
00084
00085     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00086     auto
00087     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
00110     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00111     auto
00112     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>;
00113
00115     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00116     auto
00117     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>;
00118
00119     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00120     auto
00121     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>;
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>;
00132
00133     template< typename Scalar_T = double, const index_t LO = DEFAULT_LO, const index_t HI = DEFAULT_HI,
typename Tune_P = tuning<> >
00134     class matrix_multi {
00135     public:
00136         clifford_algebra< Scalar_T, index_set<LO,HI>, matrix_multi<Scalar_T,LO,HI,Tune_P> >
00137         {
00138         public:
00139             using multivector_t = matrix_multi;
00140             using matrix_multi_t = multivector_t;
00141             using scalar_t = Scalar_T;
00142             using tune_p = Tune_P;
00143             using index_set_t = index_set<LO, HI>;
00144             using term_t = std::pair<const index_set_t, Scalar_T>;
00145             using vector_t = std::vector<Scalar_T>;
00146             using error_t = error<multivector_t>;
00147             using framed_multi_t = framed_multi<Scalar_T,LO,HI,Tune_P>;
00148             template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
00149             friend class framed_multi;
00150             template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
Other_Tune_P >
00151             friend class matrix_multi;
00152
00153     private:
00154         using orientation_t = ublas::row_major;
00155         using basis_matrix_t = ublas::compressed_matrix<int, orientation_t>;
00156         using matrix_t = ublas::matrix<Scalar_T, orientation_t>;
00157         using matrix_index_t = typename matrix_t::size_type;
00158
00159     public:
00160         static auto classname() -> const std::string;
00161         ~matrix_multi() override = default;
00162         matrix_multi();
00163         template< typename Other_Scalar_T >
00164         matrix_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val);
00165         template< typename Other_Scalar_T >
00166         matrix_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val,
const index_set_t frm, const bool prechecked = false);
00167         matrix_multi(const multivector_t& val,
const index_set_t frm, const bool prechecked = false);
00168         matrix_multi(const index_set_t ist, const Scalar_T& crd = Scalar_T(1));
00169         matrix_multi(const index_set_t ist, const Scalar_T& crd,
const index_set_t frm, const bool prechecked = false);
00170         matrix_multi(const Scalar_T& scr, const index_set_t frm = index_set_t());

```



```

00186     matrix_multi(const int scr, const index_set_t frm = index_set_t());
00188     matrix_multi(const vector_t& vec,
00189                 const index_set_t frm, const bool prechecked = false);
00191     matrix_multi(const std::string& str);
00193     matrix_multi(const std::string& str,
00194                 const index_set_t frm, const bool prechecked = false);
00196     matrix_multi(const char* str)
00197     { *this = matrix_multi(std::string(str)); };
00199     matrix_multi(const char* str,
00200                 const index_set_t frm, const bool prechecked = false)
00201     { *this = matrix_multi(std::string(str), frm, prechecked); };
00203     template< typename Other_Scalar_T >
00204     matrix_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val);
00206     template< typename Other_Scalar_T >
00207     matrix_multi(const framed_multi<Other_Scalar_T,LO,HI,Tune_P>& val,
00208                 const index_set_t frm, const bool prechecked = false);
00210     auto fast_matrix_multi(const index_set_t frm) const -> const matrix_multi_t;
00212     template< typename Other_Scalar_T >
00213     auto fast_framed_multi() const -> const framed_multi<Other_Scalar_T,LO,HI,Tune_P>;
00214
00215 private:
00217     template< typename Matrix_T >
00218     matrix_multi(const Matrix_T& mtx, const index_set_t frm);
00220     matrix_multi(const matrix_t& mtx, const index_set_t frm);
00222     auto basis_element(const index_set<LO,HI>& ist) const -> const basis_matrix_t;
00223
00224 public:
00225     _GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS
00226
00228     auto operator= (const multivector_t& rhs) -> multivector_t&;
00229
00231     static auto random(const index_set_t frm, Scalar_T fill = Scalar_T(1)) -> const matrix_multi_t;
00232
00233     // Friend declarations
00234
00235     friend auto
00236     operator* <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
00237     friend auto
00238     operator^ <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
00239     friend auto
00240     operator& <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
00241     friend auto
00242     operator% <>(const matrix_multi_t& lhs, const matrix_multi_t& rhs) -> const matrix_multi_t;
00243     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
00255     Other_Tune_P >
00256     friend auto
00257     reframe (const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& lhs, const
00258     matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& rhs,
00259     matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& lhs_reframed,
00260     matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& rhs_reframed) -> const
00261     index_set<Other_LO,Other_HI>;
00262     template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
00263     Other_Tune_P >
00264     friend auto
00265     matrix_sqrt (const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& val,
00266     const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& i,
00267     const index_t level)
00268     -> const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>;
00269     template< typename Other_Scalar_T, const index_t Other_LO, const index_t Other_HI, typename
00270     Other_Tune_P >
00271     friend auto
00272     matrix_log (const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& val,
00273     const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>& i,
00274     const index_t level)
00275     -> const matrix_multi<Other_Scalar_T,Other_LO,Other_HI,Other_Tune_P>;
00276
00277     auto operator+= (const term_t& rhs) -> multivector_t&;
00278
00279 private:
00280     // Data members
00281     index_set_t      m_frame;
00282     matrix_t         m_matrix;
00283 };
00284
00285 // 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

```

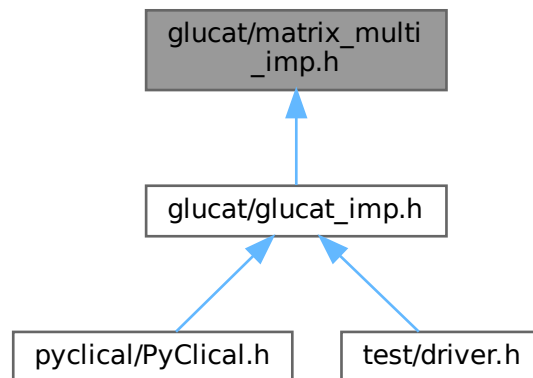
#include "glucat/matrix_multi.h"
#include "glucat/scalar.h"
#include "glucat/generation.h"
#include "glucat/matrix.h"
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/matrix_expression.hpp>
#include <boost/numeric/ublas/matrix_proxy.hpp>
#include <boost/numeric/ublas/matrix_sparse.hpp>
#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.*

## Variables

- template<typename Scalar\_T>  
const [pade\\_sqrt\\_numer](#)< Scalar\_T >::array [pade::pade\\_sqrt\\_numer](#)< Scalar\_T >::numer
- template<typename Scalar\_T>  
const [pade\\_sqrt\\_denom](#)< Scalar\_T >::array [pade::pade\\_sqrt\\_denom](#)< Scalar\_T >::denom
- const [pade\\_sqrt\\_numer](#)< float >::array [pade::pade\\_sqrt\\_numer](#)< float >::numer
- const [pade\\_sqrt\\_denom](#)< float >::array [pade::pade\\_sqrt\\_denom](#)< float >::denom
- const [pade\\_sqrt\\_numer](#)< longdouble >::array [pade::pade\\_sqrt\\_numer](#)< longdouble >::numer
- const [pade\\_sqrt\\_denom](#)< longdouble >::array [pade::pade\\_sqrt\\_denom](#)< longdouble >::denom
- const [pade\\_sqrt\\_numer](#)< dd\_real >::array [pade::pade\\_sqrt\\_numer](#)< dd\_real >::numer
- const [pade\\_sqrt\\_denom](#)< dd\_real >::array [pade::pade\\_sqrt\\_denom](#)< dd\_real >::denom

- const `pade_sqrt_numer`< qd\_real >::array `pade::pade_sqrt_numer`< qd\_real >::numer
- const `pade_sqrt_denom`< qd\_real >::array `pade::pade_sqrt_denom`< qd\_real >::denom
- template<typename Scalar\_T>  
const `pade_log_numer`< Scalar\_T >::array `pade::pade_log_numer`< Scalar\_T >::numer
- template<typename Scalar\_T>  
const `pade_log_denom`< Scalar\_T >::array `pade::pade_log_denom`< Scalar\_T >::denom
- const `pade_log_numer`< float >::array `pade::pade_log_numer`< float >::numer
- const `pade_log_denom`< float >::array `pade::pade_log_denom`< float >::denom
- const `pade_log_numer`< longdouble >::array `pade::pade_log_numer`< longdouble >::numer
- const `pade_log_denom`< longdouble >::array `pade::pade_log_denom`< longdouble >::denom
- const `pade_log_numer`< dd\_real >::array `pade::pade_log_numer`< dd\_real >::numer
- const `pade_log_denom`< dd\_real >::array `pade::pade_log_denom`< dd\_real >::denom
- const `pade_log_numer`< qd\_real >::array `pade::pade_log_numer`< qd\_real >::numer
- const `pade_log_denom`< qd\_real >::array `pade::pade_log_denom`< qd\_real >::denom

## 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
00084     auto
00085     offset_level(const index_t p, const index_t q) -> index_t
00086     {
00087         // Offsets between the log2 of the matrix dimension for the current signature
00088         // and that of the real superalgebra
00089         static const std::array<int, 8> offset_log2_dim = {0, 1, 0, 1, 1, 2, 1, 1};
00090         const index_t bott = pos_mod(p-q, 8);
00091         return (p+q)/2 + offset_log2_dim[bott];
00092     }
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
00104     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00105     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00106     matrix_multi()
00107     : m_frame( index_set_t() ),
00108       m_matrix( matrix_t( 1, 1 ) )
00109     { this->m_matrix.clear(); }
00110
00111
00112     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00113     template< typename Other_Scalar_T >
00114     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00115     matrix_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val)
00116     : m_frame( val.m_frame ), m_matrix( val.m_matrix.size1(), val.m_matrix.size2() )
00117     {
00118         this->m_matrix.clear();
00119         for (auto
00120             val_it1 = val.m_matrix.begin1();
00121             val_it1 != val.m_matrix.end1();
00122             ++val_it1)
00123             for (auto
00124                 val_it2 = val_it1.begin();
00125                 val_it2 != val_it1.end();
00126                 ++val_it2)
00127                 this->m_matrix(val_it2.index1(), val_it2.index2()) =
00128                 numeric_traits<Scalar_T>::to_scalar_t(*val_it2);
00129     }
00130
00131
00132     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00133     template< typename Other_Scalar_T >
00134     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00135     matrix_multi(const matrix_multi<Other_Scalar_T,LO,HI,Tune_P>& val, const index_set_t frm, const bool
00136     prechecked)
00137     : m_frame( frm )
00138     {
00139         if (frm != val.m_frame)
00140             *this = multivector_t(framed_multi_t(val), frm);
00141         else
00142         {
00143             const matrix_index_t dim = folded_dim<matrix_index_t>(frm);
00144             this->m_matrix.resize(dim, dim, false);
00145             this->m_matrix.clear();
00146             for (auto
00147                 val_it1 = val.m_matrix.begin1();

```

```

00147         val_it1 != val.m_matrix.end1();
00148         ++val_it1)
00149     for (auto
00150         val_it2 = val_it1.begin();
00151         val_it2 != val_it1.end();
00152         ++val_it2)
00153         this->m_matrix(val_it2.index1(), val_it2.index2()) =
numeric_traits<Scalar_T>::to_scalar_t(*val_it2);
00154     }
00155 }
00156
00157 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00158 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00159 matrix_multi(const multivector_t& val, const index_set_t frm, const bool prechecked)
00160 : m_frame( frm )
00161 {
00162     if (frm != val.m_frame)
00163         *this = multivector_t(framed_multi_t(val), frm);
00164     else
00165         this->m_matrix = val.m_matrix;
00166 }
00167
00168 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00169 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00170 matrix_multi(const index_set_t ist, const Scalar_T& crd)
00171 : m_frame( ist )
00172 {
00173     const auto dim = folded_dim<matrix_index_t>(this->m_frame);
00174     this->m_matrix.resize(dim, dim, false);
00175     this->m_matrix.clear();
00176     *this += term_t(ist, crd);
00177 }
00178
00179 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00180 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00181 matrix_multi(const index_set_t ist, const Scalar_T& crd, const index_set_t frm, const bool
prechecked)
00182 : m_frame( frm )
00183 {
00184     if (!prechecked && (ist | frm) != frm)
00185         throw error_t("multivector_t(ist,crd,frm): cannot initialize with value outside of frame");
00186     const matrix_index_t dim = folded_dim<matrix_index_t>(frm);
00187     this->m_matrix.resize(dim, dim, false);
00188     this->m_matrix.clear();
00189     *this += term_t(ist, crd);
00190 }
00191
00192 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00193 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00194 matrix_multi(const Scalar_T& scr, const index_set_t frm)
00195 : m_frame( frm )
00196 {
00197     const auto dim = folded_dim<matrix_index_t>(frm);
00198     this->m_matrix.resize(dim, dim, false);
00199     this->m_matrix.clear();
00200     *this += term_t(index_set_t(), scr);
00201 }
00202
00203 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00204 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00205 matrix_multi(const int scr, const index_set_t frm)
00206 { *this = multivector_t(Scalar_T(scr), frm); }
00207
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 vector_t& vec,
00211             const index_set_t frm, const bool prechecked)
00212 : m_frame( frm )
00213 {
00214     if (!prechecked && index_t(vec.size()) != frm.count())
00215         throw error_t("multivector_t(vec,frm): cannot initialize with vector not matching frame");
00216     const auto dim = folded_dim<matrix_index_t>(frm);
00217     this->m_matrix.resize(dim, dim, false);
00218     this->m_matrix.clear();
00219     auto idx = frm.min();
00220     const auto frm_end = frm.max()+1;
00221     for (auto& crd : vec)
00222     {
00223         *this += term_t(index_set_t(idx), crd);
00224         for (
00225             ++idx;
00226             idx != frm_end && !frm[idx];
00227             ++idx)
00228         ;
00229     }
00230 }
00231
00232
00233
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
00244     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00245     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00246     matrix_multi(const std::string& str, const index_set_t frm, const bool prechecked)
00247     { *this = multivector_t(framed_multi_t(str), frm, prechecked); }
00248
00249     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00250     template< typename Other_Scalar_T >
00251     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00252     matrix_multi(const framed_multi_t<Other_Scalar_T,LO,HI,Tune_P>& val)
00253     : m_frame( val.frame() )
00254     {
00255         if (val.size() >= Tune_P::fast_size_threshold)
00256         {
00257             try
00258             {
00259                 *this = val.template fast_matrix_multi<Scalar_T>(this->m_frame);
00260                 return;
00261             }
00262             catch (const glucat_error& e)
00263             { }
00264             const auto dim = folded_dim<matrix_index_t>(this->m_frame);
00265             this->m_matrix.resize(dim, dim, false);
00266             this->m_matrix.clear();
00267
00268             for (auto& val_term : val)
00269                 *this += val_term;
00270         }
00271     }
00272
00273     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00274     template< typename Other_Scalar_T >
00275     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00276     matrix_multi(const framed_multi_t<Other_Scalar_T,LO,HI,Tune_P>& framed_val, const index_set_t frm,
00277     const bool prechecked)
00278     {
00279         const auto val = framed_val.truncated();
00280         const auto our_frame = val.frame() | frm;
00281         if (val.size() >= Tune_P::fast_size_threshold)
00282         {
00283             try
00284             {
00285                 *this = val.template fast_matrix_multi<Scalar_T>(our_frame);
00286                 return;
00287             }
00288             catch (const glucat_error& e)
00289             { }
00290             this->m_frame = our_frame;
00291             const auto dim = folded_dim<matrix_index_t>(our_frame);
00292             this->m_matrix.resize(dim, dim, false);
00293             this->m_matrix.clear();
00294
00295             for (auto& val_term : val)
00296                 *this += val_term;
00297         }
00298
00299     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00300     template< typename Matrix_T >
00301     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00302     matrix_multi(const Matrix_T& mtx, const index_set_t frm)
00303     : m_frame( frm ), m_matrix( mtx.size1(), mtx.size2() )
00304     {
00305         this->m_matrix.clear();
00306
00307         for (auto
00308             mtx_it1 = mtx.begin1();
00309             mtx_it1 != mtx.end1();
00310             ++mtx_it1)
00311             for (auto
00312                 mtx_it2 = mtx_it1.begin();
00313                 mtx_it2 != mtx_it1.end();
00314                 ++mtx_it2)
00315                 this->m_matrix(mtx_it2.index1(), mtx_it2.index2()) =
00316                 numeric_traits<Scalar_T>::to_scalar_t(*mtx_it2);
00317
00318     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00319     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00320     matrix_multi(const matrix_t& mtx, const index_set_t frm)
00321     : m_frame( frm ), m_matrix( mtx )
00322     { }
00323
00324     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00325     auto
00326     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00327     operator= (const multivector_t& rhs) -> multivector_t&

```

```

00330 {
00331     // Check for assignment to self
00332     if (this == &rhs)
00333         return *this;
00334     this->m_frame = rhs.m_frame;
00335     this->m_matrix = rhs.m_matrix;
00336     return *this;
00337 }
00338
00340 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00341 inline
00342 auto
00343 reframe (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs,      const
00344         matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs,
00345         matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs_reframed,
00346         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

```

```

00419     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00420     inline
00421     auto
00422     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00423     operator+= (const multivector_t& rhs) -> multivector_t&
00424     {
00425         // Ensure that there is no aliasing
00426         if (this == &rhs)
00427             return *this *= Scalar_T(2);
00428
00429         // Operate only within a common frame
00430         multivector_t rhs_reframed;
00431         const index_set_t our_frame = reframe(*this, rhs, *this, rhs_reframed);
00432         const multivector_t& rhs_ref = (rhs.m_frame == our_frame)
00433             ? rhs
00434             : rhs_reframed;
00435
00436         noalias(this->m_matrix) += rhs_ref.m_matrix;
00437         return *this;
00438     }
00439
00441     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00442     inline
00443     auto
00444     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00445     operator-= (const Scalar_T& scr) -> multivector_t&
00446     { return *this += term_t(index_set_t(), -scr); }
00447
00449     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00450     inline
00451     auto
00452     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00453     operator-= (const multivector_t& rhs) -> multivector_t&
00454     {
00455         // Ensure that there is no aliasing
00456         if (this == &rhs)
00457             return *this = Scalar_T(0);
00458
00459         // Operate only within a common frame
00460         multivector_t rhs_reframed;
00461         const index_set_t our_frame = reframe(*this, rhs, *this, rhs_reframed);
00462         const multivector_t& rhs_ref = (rhs.m_frame == our_frame)
00463             ? rhs
00464             : rhs_reframed;
00465
00466         noalias(this->m_matrix) -= rhs_ref.m_matrix;
00467         return *this;
00468     }
00469
00471     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00472     inline
00473     auto
00474     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00475     operator- () const -> const multivector_t
00476     { return multivector_t(-(this->m_matrix), this->m_frame); }
00477
00479     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00480     inline
00481     auto
00482     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00483     operator*= (const Scalar_T& scr) -> multivector_t&
00484     { // multiply coordinates of all terms by scalar
00485
00486         using traits_t = numeric_traits<Scalar_T>;
00487         if (traits_t::isNan_or_isInf(scr) || this->isnan())
00488             return *this = traits_t::NaN();
00489         if (scr == Scalar_T(0))
00490             *this = Scalar_T(0);
00491         else
00492             this->m_matrix *= scr;
00493         return *this;
00494     }
00495
00497     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00498     inline
00499     auto
00500     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>
00501     {
00502         using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
00503         using index_set_t = typename multivector_t::index_set_t;
00504
00505         if (lhs.isnan() || rhs.isnan())
00506             return numeric_traits<Scalar_T>::NaN();
00507
00508         // Operate only within a common frame
00509         multivector_t lhs_reframed;

```

```

00510     multivector_t rhs_reframed;
00511     const index_set_t our_frame = reframe(lhs, rhs, lhs_reframed, rhs_reframed);
00512     const multivector_t& lhs_ref = (lhs.m_frame == our_frame)
00513         ? lhs
00514         : lhs_reframed;
00515     const multivector_t& rhs_ref = (rhs.m_frame == our_frame)
00516         ? rhs
00517         : rhs_reframed;
00518
00519     using matrix_t = typename multivector_t::matrix_t;
00520     using matrix_index_t = typename matrix_t::size_type;
00521
00522     const matrix_index_t dim = lhs_ref.m_matrix.size1();
00523     multivector_t result = multivector_t(matrix_t(dim, dim), our_frame);
00524     result.m_matrix.clear();
00525     ublas::axpy_prod(lhs_ref.m_matrix, rhs_ref.m_matrix, result.m_matrix, true);
00526     return result;
00527 }
00528
00530 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00531 inline
00532 auto
00533 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00534 operator*= (const multivector_t& rhs) -> multivector_t&
00535 { return *this = *this * rhs; }
00536
00538 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00539 inline
00540 auto
00541 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>
00542 {
00543     using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
00544     using framed_multi_t = typename multivector_t::framed_multi_t;
00545     return framed_multi_t(lhs) ^ framed_multi_t(rhs);
00546 }
00547
00549 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00550 inline
00551 auto
00552 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00553 operator^= (const multivector_t& rhs) -> multivector_t&
00554 { return *this = *this ^ rhs; }
00555
00557 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00558 inline
00559 auto
00560 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>
00561 {
00562     using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
00563     using framed_multi_t = typename multivector_t::framed_multi_t;
00564     return framed_multi_t(lhs) & framed_multi_t(rhs);
00565 }
00566
00568 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00569 inline
00570 auto
00571 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00572 operator&= (const multivector_t& rhs) -> multivector_t&
00573 { return *this = *this & rhs; }
00574
00576 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00577 inline
00578 auto
00579 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>
00580 {
00581     using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
00582     using framed_multi_t = typename multivector_t::framed_multi_t;
00583     return framed_multi_t(lhs) % framed_multi_t(rhs);
00584 }
00585
00587 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00588 inline
00589 auto
00590 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00591 operator%= (const multivector_t& rhs) -> multivector_t&
00592 { return *this = *this % rhs; }
00593
00595 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00596 inline
00597 auto
00598 star(const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs)
-> Scalar_T
00599 { return (lhs * rhs).scalar(); }
00600

```

```

00602     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00603     inline
00604     auto
00605     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00606     operator/=(const Scalar_T& scr) -> multivector_t&
00607     { return *this *= Scalar_T(1)/scr; }
00608
00610     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00611     auto
00612     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>
00613     {
00614         using traits_t = numeric_traits<Scalar_T>;
00615
00616         if (lhs.isnan() || rhs.isnan())
00617             return traits_t::NaN();
00618
00619         if (rhs == Scalar_T(0))
00620             return traits_t::NaN();
00621
00622         using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
00623
00624         // Operate only within a common frame
00625         multivector_t lhs_reframed;
00626         multivector_t rhs_reframed;
00627         const auto our_frame = reframe(lhs, rhs, lhs_reframed, rhs_reframed);
00628         const auto& lhs_ref = (lhs.m_frame == our_frame)
00629             ? lhs
00630             : lhs_reframed;
00631         const auto& rhs_ref = (rhs.m_frame == our_frame)
00632             ? rhs
00633             : rhs_reframed;
00634
00635         // Solve result == lhs_ref/rhs_ref <=> result*rhs_ref == lhs_ref
00636         // We now solve X == B/A
00637         // (where X == result, B == lhs_ref.m_matrix and A == rhs_ref.m_matrix)
00638         // X == B/A <=> X*A == B <=> AT*XT == BT
00639         // So, we solve AT*XT == BT
00640
00641         using matrix_t = typename multivector_t::matrix_t;
00642         using matrix_index_t = typename matrix_t::size_type;
00643
00644         const auto& AT = matrix_t(ublas::trans(rhs_ref.m_matrix));
00645         auto LU = AT;
00646
00647         using permutation_t = ublas::permutation_matrix<matrix_index_t>;
00648
00649         auto pvector = permutation_t(AT.size());
00650         if (! ublas::lu_factorize(LU, pvector))
00651         {
00652             const auto& BT = matrix_t(ublas::trans(lhs_ref.m_matrix));
00653             auto XT = BT;
00654             ublas::lu_substitute(LU, pvector, XT);
00655             if (matrix::isnan(XT))
00656                 return traits_t::NaN();
00657
00658             // Iterative refinement.
00659             // Reference: Nicholas J. Higham, "Accuracy and Stability of Numerical Algorithms",
00660             // SIAM, 1996, ISBN 0-89871-355-2, Chapter 11
00661             if (Tune_P::div_max_steps > 0)
00662             {
00663                 // matrix_t R = ublas::prod(AT, XT) - BT;
00664                 auto R = matrix_t(-BT);
00665                 ublas::axpy_prod(AT, XT, R, false);
00666                 if (matrix::isnan(R))
00667                     return traits_t::NaN();
00668
00669                 auto nr = Scalar_T(ublas::norm_inf(R));
00670                 if ( nr != Scalar_T(0) && !traits_t::isNaN_or_isInf(nr) )
00671                 {
00672                     auto XTnew = XT;
00673                     auto nrold = nr + Scalar_T(1);
00674                     for (auto
00675                         step = 0;
00676                         step != Tune_P::div_max_steps &&
00677                         nr < nrold &&
00678                         nr != Scalar_T(0) &&
00679                         nr == nr;
00680                         ++step)
00681                     {
00682                         nrold = nr;
00683                         if (step != 0)
00684                             XT = XTnew;
00685                         auto& D = R;
00686                         ublas::lu_substitute(LU, pvector, D);
00687                         XTnew -= D;
00688                         // noalias(R) = ublas::prod(AT, XTnew) - BT;

```

```

00689         R = -BT;
00690         ublas::axpy_prod(AT, XTnew, R, false);
00691         nr = ublas::norm_inf(R);
00692     }
00693 }
00694 }
00695     return multivector_t(ublas::trans(XT), our_frame);
00696 }
00697 else
00698     // AT is singular. Return NaN
00699     return traits_t::NaN();
00700 }
00701
00702 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00703 inline
00704 auto
00705 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00706 operator/=(const multivector_t& rhs) -> multivector_t&
00707 { return *this = *this / rhs; }
00708
00709 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00710 inline
00711 auto
00712 operator| (const matrix_multi<Scalar_T,LO,HI,Tune_P>& lhs, const
00713 matrix_multi<Scalar_T,LO,HI,Tune_P>& rhs) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
00714 { return rhs * lhs / rhs.involute(); }
00715
00716 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00717 inline
00718 auto
00719 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00720 operator|= (const multivector_t& rhs) -> multivector_t&
00721 { return *this = rhs * *this / rhs.involute(); }
00722
00723 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00724 inline
00725 auto
00726 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00727 inv() const -> const multivector_t
00728 { return multivector_t(Scalar_T(1), this->m_frame) / *this; }
00729
00730 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00731 inline
00732 auto
00733 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00734 pow(int m) const -> const multivector_t
00735 { return glucat::pow(*this, m); }
00736
00737 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00738 auto
00739 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00740 outer_pow(int m) const -> const multivector_t
00741 {
00742     if (m < 0)
00743         throw error_t("outer_pow(m): negative exponent");
00744     framed_multi_t a = *this;
00745     return a.outer_pow(m);
00746 }
00747
00748 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00749 inline
00750 auto
00751 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00752 grade() const -> index_t
00753 { return framed_multi_t(*this).grade(); }
00754
00755 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00756 inline
00757 auto
00758 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00759 frame() const -> const index_set_t
00760 { return this->m_frame; }
00761
00762 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00763 inline
00764 auto
00765 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00766 operator[] (const index_set_t ist) const -> Scalar_T
00767 {
00768     // Use matrix inner product only if ist is in frame
00769     if ( (ist | this->m_frame) == this->m_frame)
00770         return matrix::inner<Scalar_T>(this->basis_element(ist), this->m_matrix);
00771     else
00772         return Scalar_T(0);
00773 }
00774
00775 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >

```

```

00785     inline
00786     auto
00787     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00788     operator() (index_t grade) const -> const multivector_t
00789     {
00790         if ((grade < 0) || (grade > HI-LO))
00791             return 0;
00792         else
00793             return (framed_multi_t(*this))(grade);
00794     }
00795
00796     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00797     inline
00798     auto
00799     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00800     scalar() const -> Scalar_T
00801     {
00802         const matrix_index_t dim = this->m_matrix.size1();
00803         return matrix::trace(this->m_matrix) / Scalar_T( double(dim) );
00804     }
00805
00806     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00807     inline
00808     auto
00809     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00810     pure() const -> const multivector_t
00811     { return *this - this->scalar(); }
00812
00813     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00814     inline
00815     auto
00816     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00817     even() const -> const multivector_t
00818     { return framed_multi_t(*this).even(); }
00819
00820     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00821     inline
00822     auto
00823     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00824     odd() const -> const multivector_t
00825     { return framed_multi_t(*this).odd(); }
00826
00827     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00828     auto
00829     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00830     vector_part() const -> const vector_t
00831     { return this->vector_part(this->frame(), true); }
00832
00833     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00834     auto
00835     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00836     vector_part(const index_set_t frm, const bool prechecked) const -> const vector_t
00837     {
00838         if (!prechecked && (this->frame() | frm) != frm)
00839             throw error_t("vector_part(frm): value is outside of requested frame");
00840         vector_t result;
00841         // If we need to enlarge the frame we may as well use a framed_multi_t
00842         if (this->frame() != frm)
00843             return framed_multi_t(*this).vector_part(frm, true);
00844
00845         const auto begin_index = frm.min();
00846         const auto end_index = frm.max()+1;
00847         for (auto
00848             idx = begin_index;
00849             idx != end_index;
00850             ++idx)
00851             if (frm[idx])
00852                 // Frame may contain indices which do not correspond to a grade 1 term but
00853                 // frame cannot omit any index corresponding to a grade 1 term
00854                 result.push_back(
00855                     matrix::inner<Scalar_T>(this->basis_element(index_set_t(idx)),
00856                     this->m_matrix));
00857         return result;
00858     }
00859
00860     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00861     inline
00862     auto
00863     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00864     involute() const -> const multivector_t
00865     { return framed_multi_t(*this).involute(); }
00866
00867     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00868     inline
00869     auto
00870     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00871     reverse() const -> const multivector_t

```

```

00880 { return framed_multi_t(*this).reverse(); }
00881
00882 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00883 inline
00884 auto
00885 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00886 conj() const -> const multivector_t
00887 { return framed_multi_t(*this).conj(); }
00888
00889
00890 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00891 inline
00892 auto
00893 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00894 quad() const -> Scalar_T
00895 { // scalar(conj(x)*x) = 2*quad(even(x)) - quad(x)
00896   // Arvind Raja ref: "old clical: quadfunction(p:pter):pterm in file compmod.pas"
00897   return framed_multi_t(*this).quad();
00898 }
00899
00900
00901 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00902 inline
00903 auto
00904 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00905 norm() const -> Scalar_T
00906 {
00907   const matrix_index_t dim = this->m_matrix.size1();
00908   return matrix::norm_frob2(this->m_matrix) / Scalar_T( double(dim) );
00909 }
00910
00911
00912 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00913 inline
00914 auto
00915 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00916 max_abs() const -> Scalar_T
00917 { return framed_multi_t(*this).max_abs(); }
00918
00919
00920 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00921 auto
00922 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00923 random(const index_set<LO,HI> frm, Scalar_T fill) -> const multivector_t
00924 {
00925   return framed_multi<Scalar_T,LO,HI,Tune_P>::random(frm, fill);
00926 }
00927
00928
00929 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00930 inline
00931 void
00932 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00933 write(const std::string& msg) const
00934 { framed_multi_t(*this).write(msg); }
00935
00936
00937 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00938 inline
00939 void
00940 matrix_multi<Scalar_T,LO,HI,Tune_P>::
00941 write(std::ofstream& ofile, const std::string& msg) const
00942 {
00943   if (!ofile)
00944     throw error_t("write(ofile,msg): cannot write to output file");
00945   framed_multi_t(*this).write(ofile, msg);
00946 }
00947
00948
00949 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00950 inline
00951 auto
00952 operator<< (std::ostream& os, const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::ostream&
00953 {
00954   os << typename matrix_multi<Scalar_T,LO,HI,Tune_P>::framed_multi_t(val);
00955   return os;
00956 }
00957
00958
00959 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00960 inline
00961 auto
00962 operator>> (std::istream& s, matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> std::istream&
00963 { // Input looks like 1.0-2.0{1,2}+3.2{3,4}
00964   framed_multi<Scalar_T,LO,HI,Tune_P> local;
00965   s >> local;
00966   // If s.bad() then we have a corrupt input
00967   // otherwise we are fine and can copy the resulting matrix_multi
00968   if (!s.bad())
00969     val = local;
00970   return s;
00971 }
00972
00973
00974 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00975 inline

```



```

00977     auto
00978     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00979     isinf() const -> bool
00980     {
00981         if (std::numeric_limits<Scalar_T>::has_infinity)
00982             return matrix::isinf(this->m_matrix);
00983         else
00984             return false;
00985     }
00986
00987     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
00988     inline
00989     auto
00990     matrix_multi<Scalar_T,LO,HI,Tune_P>::
00991     isnan() const -> bool
00992     {
00993         if (std::numeric_limits<Scalar_T>::has_quiet_NaN)
00994             return matrix::isnan(this->m_matrix);
00995         else
00996             return false;
00997     }
00998
00999     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01000     inline
01001     auto
01002     matrix_multi<Scalar_T,LO,HI,Tune_P>::
01003     truncated(const Scalar_T& limit) const -> const multivector_t
01004     { return framed_multi_t(*this).truncated(limit); }
01005
01006     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01007     inline
01008     auto
01009     matrix_multi<Scalar_T,LO,HI,Tune_P>::
01010     operator+= (const term_t& term) -> multivector_t&
01011     {
01012         if (term.second != Scalar_T(0))
01013             this->m_matrix.plus_assign(matrix_t(this->basis_element(term.first)) * term.second);
01014         return *this;
01015     }
01016
01017     template< typename Multivector_T, typename Matrix_T, typename Basis_Matrix_T >
01018     static
01019     auto
01020     fast(const Matrix_T& X, index_t level) -> Multivector_T
01021     {
01022         using framed_multi_t = Multivector_T;
01023
01024         using index_set_t = typename framed_multi_t::index_set_t;
01025         using Scalar_T = typename framed_multi_t::scalar_t;
01026         using matrix_t = Matrix_T;
01027         using basis_matrix_t = Basis_Matrix_T;
01028         using basis_scalar_t = typename basis_matrix_t::value_type;
01029         using traits_t = numeric_traits<Scalar_T>;
01030
01031         if (level == 0)
01032             return framed_multi_t(traits_t::to_scalar_t(X(0,0)));
01033
01034         if (ublas::norm_inf(X) == 0)
01035             return Scalar_T(0);
01036
01037         const basis_matrix_t& I = matrix::unit<basis_matrix_t>(2);
01038         basis_matrix_t J(2,2,2);
01039         J.clear();
01040         J(0,1) = basis_scalar_t(-1);
01041         J(1,0) = basis_scalar_t( 1);
01042         basis_matrix_t K = J;
01043         K(0,1) = basis_scalar_t( 1);
01044         basis_matrix_t JK = I;
01045         JK(0,0) = basis_scalar_t(-1);
01046
01047         using matrix::signed_perm_nork;
01048         const index_set_t ist_mn = index_set_t(-level);
01049         const index_set_t ist_pn = index_set_t(level);
01050         const index_set_t ist_mnpn = ist_mn | ist_pn;
01051         if (level == 1)
01052         {
01053             using term_t = typename framed_multi_t::term_t;
01054             const Scalar_T i_x = traits_t::to_scalar_t(signed_perm_nork(I, X)(0, 0));
01055             const Scalar_T j_x = traits_t::to_scalar_t(signed_perm_nork(J, X)(0, 0));
01056             const Scalar_T k_x = traits_t::to_scalar_t(signed_perm_nork(K, X)(0, 0));
01057             const Scalar_T jk_x = traits_t::to_scalar_t(signed_perm_nork(JK,X)(0, 0));
01058             framed_multi_t
01059                 result = i_x;
01060             result += term_t(ist_mn, j_x); // j_x * mn;
01061             result += term_t(ist_pn, k_x); // k_x * pn;
01062             return result += term_t(ist_mnpn, jk_x); // jk_x * mnpn;
01063         }
01064     }

```

```

01068     else
01069     {
01070         const framed_multi_t& mn = framed_multi_t(ist_mn);
01071         const framed_multi_t& pn = framed_multi_t(ist_pn);
01072         const framed_multi_t& mnpn = framed_multi_t(ist_mnpn);
01073         const framed_multi_t& i_x = fast<framed_multi_t, matrix_t, basis_matrix_t>
01074             (signed_perm_nork(I, X), level-1);
01075         const framed_multi_t& j_x = fast<framed_multi_t, matrix_t, basis_matrix_t>
01076             (signed_perm_nork(J, X), level-1);
01077         const framed_multi_t& k_x = fast<framed_multi_t, matrix_t, basis_matrix_t>
01078             (signed_perm_nork(K, X), level-1);
01079         const framed_multi_t& jk_x = fast<framed_multi_t, matrix_t, basis_matrix_t>
01080             (signed_perm_nork(JK, X), level-1);
01081         framed_multi_t
01082             result = i_x.even() - jk_x.odd();
01083             result += (j_x.even() - k_x.odd()) * mn;
01084             result += (k_x.even() - j_x.odd()) * pn;
01085         return result += (jk_x.even() - i_x.odd()) * mnpn;
01086     }
01087 }
01088
01090 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01091 inline
01092 auto
01093 matrix_multi<Scalar_T,LO,HI,Tune_P>::
01094 fast_matrix_multi(const index_set_t frm) const -> const multivector_t
01095 {
01096     if (this->m_frame == frm)
01097         return *this;
01098     else
01099         return (this->template fast_framed_multi<Scalar_T>()).template fast_matrix_multi<Scalar_T>(frm);
01100 }
01101
01103 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01104 template <typename Other_Scalar_T>
01105 auto
01106 matrix_multi<Scalar_T,LO,HI,Tune_P>::
01107 fast_framed_multi() const -> const framed_multi<Other_Scalar_T,LO,HI,Tune_P>
01108 {
01109     // Determine the amount of off-centering needed
01110     index_t p = this->m_frame.count_pos();
01111     index_t q = this->m_frame.count_neg();
01112
01113     const index_t bott = pos_mod(p-q, 8);
01114     p += std::max(gen::offset_to_super[bott],index_t(0));
01115     q -= std::min(gen::offset_to_super[bott],index_t(0));
01116
01117     const index_t orig_p = p;
01118     const index_t orig_q = q;
01119     while (p-q > 4)
01120     { p -= 4; q += 4; }
01121     while (p-q < -3)
01122     { p += 4; q -= 4; }
01123     if (p-q > 1)
01124     {
01125         index_t old_p = p;
01126         p = q+1;
01127         q = old_p-1;
01128     }
01129     const index_t level = (p+q)/2;
01130
01131     // Do the inverse fast transform
01132     using framed_multi_t = framed_multi<Other_Scalar_T,LO,HI,Tune_P>;
01133     framed_multi_t val = fast<framed_multi_t, matrix_t, basis_matrix_t>(this->m_matrix, level);
01134
01135     // Off-centre val
01136     switch (pos_mod(orig_p-orig_q, 8))
01137     {
01138     case 2:
01139     case 3:
01140     case 4:
01141         val.centred_qpl_pml(p, q);
01142         break;
01143     default:
01144         break;
01145     }
01146     if (orig_p-orig_q > 4)
01147         while (p != orig_p)
01148             val.centred_pp4_qm4(p, q);
01149     if (orig_p-orig_q < -3)
01150         while (p != orig_p)
01151             val.centred_pm4_qp4(p, q);
01152
01153     // Return unfolded val
01154     return val.unfold(this->m_frame);
01155 }
01156

```

```

01158     template< typename Scalar_T, const index_t LO, const index_t HI, typename Matrix_T >
01159     class basis_table :
01160     public std::map< std::pair< const index_set<LO,HI>, const index_set<LO,HI> >,
01161         Matrix_T* >
01162     {
01163     public:
01164         static auto basis() -> basis_table& { static basis_table b; return b;}
01165     private:
01166         friend class friend_for_private_destructor;
01167         // Enforce singleton
01168         // Reference: A. Alexandrescu, "Modern C++ Design", Chapter 6
01169         basis_table() = default;
01170         ~basis_table() = default;
01171     public:
01172         basis_table(const basis_table&) = delete;
01173         auto operator= (const basis_table&) -> basis_table& = delete;
01174     };
01175
01176     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01177     auto
01178     matrix_multi<Scalar_T,LO,HI,Tune_P>::
01179     basis_element(const index_set_t& ist) const -> const basis_matrix_t
01180     {
01181         using index_set_pair_t = std::pair<const index_set_t, const index_set_t>;
01182         const auto& unfolded_pair = index_set_pair_t(ist, this->m_frame);
01183
01184         using basis_table_t = basis_table<Scalar_T, LO, HI, basis_matrix_t>;
01185         auto& basis_cache = basis_table_t::basis();
01186
01187         const auto frame_count = this->m_frame.count();
01188         const auto use_cache = frame_count <= index_t(Tune_P::basis_max_count);
01189
01190         if (use_cache)
01191         {
01192             const auto basis_it = basis_cache.find(unfolded_pair);
01193             if (basis_it != basis_cache.end())
01194                 return *(basis_it->second);
01195         }
01196         const auto folded_set = ist.fold(this->m_frame);
01197         const auto folded_frame = this->m_frame.fold();
01198         const auto& folded_pair = index_set_pair_t(folded_set, folded_frame);
01199         using basis_pair_t = std::pair<const index_set_pair_t, basis_matrix_t*>;
01200         if (use_cache)
01201         {
01202             const auto basis_it = basis_cache.find(folded_pair);
01203             if (basis_it != basis_cache.end())
01204             {
01205                 auto* result_ptr = basis_it->second;
01206                 basis_cache.insert(basis_pair_t(unfolded_pair, result_ptr));
01207                 return *result_ptr;
01208             }
01209         }
01210         const auto folded_max = folded_frame.max();
01211         const auto folded_min = folded_frame.min();
01212         const auto p = std::max(folded_max, index_t(0));
01213         const auto q = std::max(index_t(-folded_min), index_t(0));
01214         const auto* e = (gen::generator_table<basis_matrix_t>::generator())(p, q);
01215         const auto dim = matrix_index_t(1) << offset_level(p, q);
01216         auto result = matrix::unit<basis_matrix_t>(dim);
01217         for (auto
01218             k = folded_min;
01219             k <= folded_max;
01220             ++k)
01221             if (folded_set[k])
01222                 result = matrix::mono_prod(result, e[k]);
01223         if (use_cache)
01224         {
01225             auto* result_ptr = new basis_matrix_t(result);
01226             basis_cache.insert(basis_pair_t(folded_pair, result_ptr));
01227             basis_cache.insert(basis_pair_t(unfolded_pair, result_ptr));
01228         }
01229         return result;
01230     }
01231
01232     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P, const size_t Size
01233 >
01234     inline
01235     static
01236     auto
01237     pade_approx(
01238         const std::array<Scalar_T, Size>& numer,
01239         const std::array<Scalar_T, Size>& denom,
01240         const matrix_multi<Scalar_T,LO,HI,Tune_P>& X) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
01241     {
01242         // Pade' approximation
01243         // Reference: [GW], Section 4.3, pp318-322
01244         // Reference: [GL], Section 11.3, p572-576.
01245     }

```

```

01250
01251     using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
01252     using traits_t = numeric_traits<Scalar_T>;
01253
01254     if (X.isnan())
01255         return traits_t::NaN();
01256
01257     // Array size is assumed to be even
01258     const auto nbr_even_powers = Size/2 - 1;
01259
01260     // Create an array of even powers
01261     auto XX = std::vector<multivector_t>(nbr_even_powers);
01262     XX[0] = X * X;
01263     XX[1] = XX[0] * XX[0];
01264     for (auto
01265         k = size_t(2);
01266         k != nbr_even_powers;
01267         ++k)
01268         XX[k] = XX[k-2] * XX[1];
01269
01270     // Calculate numerator N and denominator D
01271     auto N = multivector_t(1);
01272     for (auto
01273         k = size_t(0);
01274         k != nbr_even_powers;
01275         ++k)
01276         N += XX[k] * numer[2*k + 3];
01277     N *= X;
01278     N += numer[0];
01279     for (auto
01280         k = size_t(0);
01281         k != nbr_even_powers;
01282         ++k)
01283         N += XX[k] * numer[2*k + 2];
01284     auto D = multivector_t(1);
01285     for (auto
01286         k = size_t(0);
01287         k != nbr_even_powers;
01288         ++k)
01289         D += XX[k] * denom[2*k + 3];
01290     D *= X;
01291     D += denom[0];
01292     for (auto
01293         k = size_t(0);
01294         k != nbr_even_powers;
01295         ++k)
01296         D += XX[k] * denom[2*k + 2];
01297     return N / D;
01298 }
01299
01300 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01301 inline
01302 static
01303 void
01304 db_step(matrix_multi<Scalar_T,LO,HI,Tune_P>& M, matrix_multi<Scalar_T,LO,HI,Tune_P>& Y)
01305 {
01306     // Reference: [CHKL]
01307     const auto& invM = inv(M);
01308     M = ((M + invM)/Scalar_T(2) + Scalar_T(1)) / Scalar_T(2);
01309     Y *= (invM + Scalar_T(1)) / Scalar_T(2);
01310 }
01311
01312 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01313 static
01314 auto
01315 db_sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
01316         Scalar_T norm_tol=std::pow(std::numeric_limits<Scalar_T>::epsilon(), 4)) -> const
01317 matrix_multi<Scalar_T,LO,HI,Tune_P>
01318 {
01319     // Reference: [CHKL]
01320     if (val == Scalar_T(0))
01321         return val;
01322
01323     static const auto sqrt_max_steps = Tune_P::db_sqrt_max_steps;
01324     auto M = val;
01325     auto Y = val;
01326
01327     for (auto
01328         step = 0;
01329         step != sqrt_max_steps && norm(M - Scalar_T(1)) > norm_tol;
01330         ++step)
01331     {
01332         if (Y.isnan())
01333             return numeric_traits<Scalar_T>::NaN();
01334         db_step(M, Y);
01335     }
01336     return Y;
01337 }

```

```

01338     }
01339
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
matrix_multi<Scalar_T,LO,HI,Tune_P>
01346     {
01347         // Reference: [MB]
01348         if (val == Scalar_T(0))
01349             return val;
01350
01351         static const auto sqrt_max_steps = Tune_P::cr_sqrt_max_steps;
01352         auto Z = Scalar_T(2) * (Scalar_T(1) + val);
01353         auto Y = Scalar_T(1) - val;
01354         auto norm_Y = norm(Y);
01355         for (auto
01356             step = 0;
01357             step != sqrt_max_steps && norm_Y > norm_Y_tol;
01358             ++step)
01359         {
01360             const auto old_norm_Y = norm_Y;
01361             Y = (-Y / Z) * Y;
01362             norm_Y = norm(Y);
01363             if (Y.isnan() || (norm_Y > old_norm_Y * Scalar_T(2)))
01364                 return numeric_traits<Scalar_T>::NaN();
01365
01366             Z += Y * Scalar_T(2);
01367         }
01368         return Z / Scalar_T(4);
01369     }
01370 }
01371
01372 namespace pade {
01373     // Reference: [Z], Padel
01374     template< typename Scalar_T >
01375     struct pade_sqrt_number
01376     {
01377     {
01378         using array = std::array<Scalar_T, 14>;
01379         static const array number;
01380     };
01381     template< typename Scalar_T >
01382     const typename pade_sqrt_number<Scalar_T>::array pade_sqrt_number<Scalar_T>::number =
01383     {
01384         1.0,          27.0/4.0,          81.0/4.0,          2277.0/64.0,
01385         10395.0/256.0,  32319.0/1024.0,   8721.0/512.0,   26163.0/4096.0,
01386         53703.0/32768.0, 36465.0/131072.0, 3861.0/131072.0, 7371.0/4194304.0,
01387         819.0/16777216.0, 27.0/67108864.0
01388     };
01389
01391     // Reference: [Z], Padel
01392     template< typename Scalar_T >
01393     struct pade_sqrt_denom
01394     {
01395         using array = std::array<Scalar_T, 14>;
01396         static const array denom;
01397     };
01398     template< typename Scalar_T >
01399     const typename pade_sqrt_denom<Scalar_T>::array pade_sqrt_denom<Scalar_T>::denom =
01400     {
01401         1.0,          25.0/4.0,          69.0/4.0,          1771.0/64.0,
01402         7315.0/256.0,  20349.0/1024.0,   4845.0/512.0,   12597.0/4096.0,
01403         21879.0/32768.0, 12155.0/131072.0, 1001.0/131072.0, 1365.0/4194304.0,
01404         91.0/16777216.0, 1.0/67108864.0
01405     };
01406
01407     template< >
01408     struct pade_sqrt_number<float>
01409     {
01410         using array = std::array<float, 10>;
01411         static const array number;
01412     };
01413     const typename pade_sqrt_number<float>::array pade_sqrt_number<float>::number =
01414     {
01415         1.0,          19.0/4.0,          19.0/2.0,          665.0/64.0,
01416         1729.0/256.0,  2717.0/1024.0,   627.0/1024.0,   627.0/8192.0,
01417         285.0/65536.0, 19.0/262144.0
01418     };
01419     template< >
01420     struct pade_sqrt_denom<float>
01421     {
01422         using array = std::array<float, 10>;
01423         static const array denom;
01424     };
01425     const typename pade_sqrt_denom<float>::array pade_sqrt_denom<float>::denom =
01426     {

```

```

01427         1.0,          17.0/4.0,          15.0/2.0,          455.0/64.0,
01428         1001.0/256.0,    1287.0/1024.0,    231.0/1024.0,    165.0/8192.0,
01429         45.0/65536,      1.0/262144.0
01430     };
01431
01432     template< >
01433     struct pade_sqrt_numer<long double>
01434     {
01435         using array = std::array<long double, 18>;
01436         static const array number;
01437     };
01438     const typename pade_sqrt_numer<long double>::array pade_sqrt_numer<long double>::number =
01439     {
01440         1.0L,          35.0L/4.0L,          35.0L,          5425.0L/64.0L,
01441         35525.0L/256.0L,    166257.0L/1024.0L,    143325.0L/1024.0L,    740025.0L/8192.0L,
01442         2877875.0L/65536.0L,    4206125.0L/262144.0L,    572033.0L/131072.0L,    1820105.0L/2097152.0L,
01443         1028755.0L/8388608.0L,    395675.0L/33554432.0L,    24225.0L/33554432.0L,    6783.0L/268435456.0L,
01444         1785.0L/4294967296.0L,    35.0L/17179869184.0L
01445     };
01446     template< >
01447     struct pade_sqrt_denom<long double>
01448     {
01449         using array = std::array<long double, 18>;
01450         static const array denom;
01451     };
01452     const typename pade_sqrt_denom<long double>::array pade_sqrt_denom<long double>::denom =
01453     {
01454         1.0L,          33.0L/4.0L,          31.0L,          4495.0L/64.0L,
01455         27405.0L/256.0L,    118755.0L/1024.0L,    94185.0L/1024.0L,    444015.0L/8192.0L,
01456         1562275.0L/65536.0L,    2042975.0L/262144.0L,    245157.0L/131072.0L,    676039.0L/2097152.0L,
01457         323323.0L/8388608.0L,    101745.0L/33554432.0L,    4845.0L/33554432.0L,    969.0L/268435456.0L,
01458         153.0L/4294967296.0L,    1.0L/17179869184.0L
01459     };
01460
01461     #if defined(GLUCAT_USE_QD)
01462     template< >
01463     struct pade_sqrt_numer<dd_real>
01464     {
01465         using array = std::array<dd_real, 22>;
01466         static const array number;
01467     };
01468     const typename pade_sqrt_numer<dd_real>::array pade_sqrt_numer<dd_real>::number =
01469     {
01470         dd_real("1"),          dd_real("43")/dd_real("4"),
01471         dd_real("215")/dd_real("4"),    dd_real("10621")/dd_real("64"),
01472         dd_real("90687")/dd_real("256"),    dd_real("567987")/dd_real("1024"),
01473         dd_real("168861")/dd_real("256"),    dd_real("1246355")/dd_real("2048"),
01474         dd_real("7228859")/dd_real("16384"),    dd_real("16583853")/dd_real("65536"),
01475         dd_real("7538115")/dd_real("65536"),    dd_real("173376645")/dd_real("4194304"),
01476         dd_real("195747825")/dd_real("16777216"),    dd_real("171655785")/dd_real("67108864"),
01477         dd_real("14375115")/dd_real("33554432"),    dd_real("14375115")/dd_real("268435456"),
01478         dd_real("20764055")/dd_real("4294967296"),    dd_real("5167525")/dd_real("17179869184"),
01479         dd_real("206701")/dd_real("17179869184"),    dd_real("76153")/dd_real("274877906944"),
01480         dd_real("3311")/dd_real("1099511627776") ,    dd_real("43")/dd_real("4398046511104")
01481     };
01482     template< >
01483     struct pade_sqrt_denom<dd_real>
01484     {
01485         using array = std::array<dd_real, 22>;
01486         static const array denom;
01487     };
01488     const typename pade_sqrt_denom<dd_real>::array pade_sqrt_denom<dd_real>::denom =
01489     {
01490         dd_real("1"),          dd_real("41")/dd_real("4"),
01491         dd_real("195")/dd_real("4"),    dd_real("9139")/dd_real("64"),
01492         dd_real("73815")/dd_real("256"),    dd_real("435897")/dd_real("1024"),
01493         dd_real("121737")/dd_real("256"),    dd_real("840565")/dd_real("2048"),
01494         dd_real("4539051")/dd_real("16384"),    dd_real("9641775")/dd_real("65536"),
01495         dd_real("4032015")/dd_real("65536"),    dd_real("84672315")/dd_real("4194304"),
01496         dd_real("86493225")/dd_real("16777216"),    dd_real("67863915")/dd_real("67108864"),
01497         dd_real("5014575")/dd_real("33554432"),    dd_real("4345965")/dd_real("268435456"),
01498         dd_real("5311735")/dd_real("4294967296"),    dd_real("1081575")/dd_real("17179869184"),
01499         dd_real("33649")/dd_real("17179869184"),    dd_real("8855")/dd_real("274877906944"),
01500         dd_real("231")/dd_real("1099511627776"),    dd_real("1")/dd_real("4398046511104")
01501     };
01502
01503     template< >
01504     struct pade_sqrt_numer<qd_real>
01505     {
01506         using array = std::array<qd_real, 34>;
01507         static const array number;
01508     };
01509     const typename pade_sqrt_numer<qd_real>::array pade_sqrt_numer<qd_real>::number =
01510     {
01511         qd_real("1"),          qd_real("67")/qd_real("4"),
01512         qd_real("134"),          qd_real("43617")/qd_real("64"),
01513         qd_real("633485")/qd_real("256"),    qd_real("6992857")/qd_real("1024"),

```

```

01514         qd_real("15246721")/qd_real("1024"),           qd_real("215632197")/qd_real("8192"),
01515         qd_real("2518145487")/qd_real("65536"),
01516         qd_real("12301285425")/qd_real("262144"),
01517         qd_real("6344873535")/qd_real("131072"),
01518         qd_real("89075432355")/qd_real("2097152"),
01519         qd_real("267226297065")/qd_real("8388608"),
01520         qd_real("687479618945")/qd_real("33554432"),
01521         qd_real("379874182975")/qd_real("33554432"),
01522         qd_real("1443521895305")/qd_real("268435456"),
01523         qd_real("9425348845815")/qd_real("4294967296"),
01524         qd_real("13195488384141")/qd_real("17179869184"),
01525         qd_real("987417498133")/qd_real("4294967296"),
01526         qd_real("8055248011085")/qd_real("137438953472"),
01527         qd_real("6958363175533")/qd_real("549755813888"),
01528         qd_real("5056698705201")/qd_real("2199023255552"),
01529         qd_real("766166470485")/qd_real("2199023255552"),
01530         qd_real("766166470485")/qd_real("17592186044416"),
01531         qd_real("623623871325")/qd_real("140737488355328"),
01532         qd_real("203123203803")/qd_real("562949953421312"),
01533         qd_real("6478601247")/qd_real("281474976710656"),
01534         qd_real("5038912081")/qd_real("4503599627370496"),
01535         qd_real("719844583")/qd_real("18014398509481984"),
01536         qd_real("71853815")/qd_real("72057594037927936"),
01537         qd_real("1165197")/qd_real("72057594037927936"),
01538         qd_real("87703")/qd_real("576460752303423488"),
01539         qd_real("12529")/qd_real("18446744073709551616"),
01540         qd_real("67")/qd_real("73786976294838206464")
01541     };
01542     template< >
01543     struct pade_sqrt_denom<qd_real>
01544     {
01545         using array = std::array<qd_real, 34>;
01546         static const array denom;
01547     };
01548     const typename pade_sqrt_denom<qd_real>::array pade_sqrt_denom<qd_real>::denom =
01549     {
01550         qd_real("1"),           qd_real("65")/qd_real("4"),
01551         qd_real("126"),         qd_real("39711")/qd_real("64"),
01552         qd_real("557845")/qd_real("256"),         qd_real("5949147")/qd_real("1024"),
01553         qd_real("12515965")/qd_real("1024"),         qd_real("170574723")/qd_real("8192"),
01554         qd_real("1916797311")/qd_real("65536"),
01555         qd_real("8996462475")/qd_real("262144"),
01556         qd_real("4450881435")/qd_real("131072"),
01557         qd_real("59826782925")/qd_real("2097152"),
01558         qd_real("171503444385")/qd_real("8388608"),
01559         qd_real("420696483235")/qd_real("33554432"),
01560         qd_real("221120793075")/qd_real("33554432"),
01561         qd_real("797168807855")/qd_real("268435456"),
01562         qd_real("4923689695575")/qd_real("4294967296"),
01563         qd_real("6499270398159")/qd_real("17179869184"),
01564         qd_real("456864812569")/qd_real("4294967296"),
01565         qd_real("3486599885395")/qd_real("137438953472"),
01566         qd_real("2804116503573")/qd_real("549755813888"),
01567         qd_real("1886827875075")/qd_real("2199023255552"),
01568         qd_real("263012370465")/qd_real("2199023255552"),
01569         qd_real("240141729555")/qd_real("17592186044416"),
01570         qd_real("176848560525")/qd_real("140737488355328"),
01571         qd_real("51538723353")/qd_real("562949953421312"),
01572         qd_real("1450433115")/qd_real("281474976710656"),
01573         qd_real("977699359")/qd_real("4503599627370496"),
01574         qd_real("118183439")/qd_real("18014398509481984"),
01575         qd_real("9652005")/qd_real("72057594037927936"),
01576         qd_real("121737")/qd_real("72057594037927936"),
01577         qd_real("6545")/qd_real("576460752303423488"),
01578         qd_real("561")/qd_real("18446744073709551616"),
01579         qd_real("1")/qd_real("73786976294838206464")
01580     };
01581 #endif
01582 }
01583 namespace glucat
01584 {
01585     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01586     auto
01587     matrix_sqrt(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
01588                 const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
01589                 const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
01590     {
01591         // Reference: [GW], Section 4.3, pp318-322
01592         // Reference: [GL], Section 11.3, p572-576
01593         // Reference: [Z], Padel
01594         using traits_t = numeric_traits<Scalar_T>;
01595         if (val.isnan())
01596             return traits_t::NaN();
01597     }

```

```

01576     const auto scr_val = val.scalar();
01577     if (val == scr_val)
01578     {
01579         if (scr_val < Scalar_T(0))
01580             return i * traits_t::sqrt(-scr_val);
01581         else
01582             return traits_t::sqrt(scr_val);
01583     }
01584
01585     // Scale val towards abs(A) == 1 or towards A == 1 as appropriate
01586     const auto scale =
01587         (scr_val != Scalar_T(0) && norm(val/scr_val - Scalar_T(1)) < Scalar_T(1))
01588         ? scr_val
01589         : (scr_val < Scalar_T(0))
01590         ? -abs(val)
01591         : abs(val);
01592     const auto sqrt_scale = traits_t::sqrt(traits_t::abs(scale));
01593     if (traits_t::isNaN_or_isInf(sqrt_scale))
01594         return traits_t::NaN();
01595
01596     using multivector_t = matrix_multi<Scalar_T, LO, HI, Tune_P>;
01597     auto rescale = multivector_t(sqrt_scale);
01598     if (scale < Scalar_T(0))
01599         rescale = i * sqrt_scale;
01600
01601     const auto& unitval = val / scale;
01602     static const auto max_norm = Scalar_T(1.0/4.0);
01603     auto use_approx_sqrt = true;
01604     auto use_cr_sqrt = false;
01605     auto scaled_result = multivector_t();
01606 #if defined(_GLUCAT_USE_EIGENVALUES)
01607     static const auto sqrt_2 = traits_t::sqrt(Scalar_T(2));
01608     if (level == 0)
01609     {
01610         // What kind of eigenvalues does the matrix contain?
01611         const auto genus = matrix::classify_eigenvalues(unitval.m_matrix);
01612         const index_t next_level =
01613             (genus.m_is_singular)
01614             ? level
01615             : level + 1;
01616         switch (genus.m_eig_case)
01617         {
01618             case matrix::neg_real_eigs:
01619                 scaled_result = matrix_sqrt(-i * unitval, i, next_level) * (i + Scalar_T(1)) / sqrt_2;
01620                 use_approx_sqrt = false;
01621                 break;
01622             case matrix::both_eigs:
01623             {
01624                 const auto safe_arg = genus.m_safe_arg;
01625                 scaled_result = matrix_sqrt(exp(i*safe_arg) * unitval, i, next_level) * exp(-i*safe_arg /
01626                     Scalar_T(2));
01627                 use_approx_sqrt = false;
01628                 break;
01629             default:
01630                 break;
01631             }
01632             use_cr_sqrt = genus.m_is_singular;
01633         }
01634 #endif
01635         if (use_approx_sqrt)
01636         {
01637             scaled_result =
01638                 (norm(unitval - Scalar_T(1)) < max_norm)
01639                 // Pade' approximation of square root
01640                 ? pade_approx(pade::pade_sqrt_numer<Scalar_T>::numer,
01641                     pade::pade_sqrt_denom<Scalar_T>::denom,
01642                     unitval - Scalar_T(1))
01643                 // Product form of Denman-Beavers square root iteration
01644                 : (use_cr_sqrt)
01645                 ? cr_sqrt(unitval)
01646                 : db_sqrt(unitval);
01647         }
01648         return (scaled_result.isnan() ||
01649             !approx_equal(pow(scaled_result, 2), unitval))
01650             ? traits_t::NaN()
01651             : scaled_result * rescale;
01652     }
01653
01654     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01655     auto
01656     sqrt(const matrix_multi<Scalar_T, LO, HI, Tune_P>& val, const matrix_multi<Scalar_T, LO, HI, Tune_P>& i,
01657         bool prechecked) -> const matrix_multi<Scalar_T, LO, HI, Tune_P>
01658     {
01659         // Reference: [GW], Section 4.3, pp318-322
01660         // Reference: [GL], Section 11.3, p572-576
01661         // Reference: [Z], Padel

```



```

01662
01663     using traits_t = numeric_traits<Scalar_T>;
01664
01665     if (val.isnan())
01666         return traits_t::NaN();
01667
01668     check_complex(val, i, prechecked);
01669
01670     switch (Tune_P::function_precision)
01671     {
01672     case precision_demoted:
01673     {
01674         using demoted_scalar_t = typename traits_t::demoted::type;
01675         using demoted_multivector_t = matrix_multi<demoted_scalar_t, LO, HI, Tune_P>;
01676
01677         const auto& demoted_val = demoted_multivector_t(val);
01678         const auto& demoted_i = demoted_multivector_t(i);
01679
01680         return matrix_sqrt(demoted_val, demoted_i, 0);
01681     }
01682     break;
01683     case precision_promoted:
01684     {
01685         using promoted_scalar_t = typename traits_t::promoted::type;
01686         using promoted_multivector_t = matrix_multi<promoted_scalar_t, LO, HI, Tune_P>;
01687
01688         const auto& promoted_val = promoted_multivector_t(val);
01689         const auto& promoted_i = promoted_multivector_t(i);
01690
01691         return matrix_sqrt(promoted_val, promoted_i, 0);
01692     }
01693     break;
01694     default:
01695         return matrix_sqrt(val, i, 0);
01696     }
01697 }
01698 }
01699
01700 namespace pade {
01701 // Reference: [Z], Padel
01702 template< typename Scalar_T >
01703 struct pade_log_number
01704 {
01705     using array = std::array<Scalar_T, 14>;
01706     static const array number;
01707 };
01708 template< typename Scalar_T >
01709 const typename pade_log_number<Scalar_T>::array pade_log_number<Scalar_T>::number =
01710 {
01711     {
01712         0.0,          1.0,          6.0,          4741.0/300.0,
01713         1441.0/60.0,   107091.0/4600.0,   8638.0/575.0,   263111.0/40250.0,
01714         153081.0/80500.0,   395243.0/1101240.0,   28549.0/688275.0,   605453.0/228813200.0,
01715         785633.0/10296594000.0,   1145993.0/1873980108000.0
01716     };
01717 };
01718 // Reference: [Z], Padel
01719 template< typename Scalar_T >
01720 struct pade_log_denom
01721 {
01722     using array = std::array<Scalar_T, 14>;
01723     static const array denom;
01724 };
01725 template< typename Scalar_T >
01726 const typename pade_log_denom<Scalar_T>::array pade_log_denom<Scalar_T>::denom =
01727 {
01728     {
01729         1.0,          13.0/2.0,          468.0/25.0,          1573.0/50.0,
01730         1573.0/46.0,   11583.0/460.0,          10296.0/805.0,          2574.0/575.0,
01731         11583.0/10925.0,   143.0/874.0,          572.0/37145.0,          117.0/148580.0,
01732         13.0/742900.0,          1.0/10400600.0
01733     };
01734 };
01735 template< >
01736 struct pade_log_number<float>
01737 {
01738     using array = std::array<float, 10>;
01739     static const array number;
01740 };
01741 const typename pade_log_number<float>::array pade_log_number<float>::number =
01742 {
01743     {
01744         0.0,          1.0,          4.0,          1337.0/204.0,
01745         385.0/68.0,   1879.0/680.0,          193.0/255.0,          197.0/1820.0,
01746         419.0/61880.0,   7129.0/61261200.0
01747     };
01748 };
01749 template< >
01750 struct pade_log_denom<float>
01751 {
01752     using array = std::array<float, 10>;

```

```

01751     static const array denom;
01752 };
01753 const typename pade_log_denom<float>::array pade_log_denom<float>::denom =
01754 {
01755     1.0,          9.0/2.0,          144.0/17.0,    147.0/17.0,
01756     441.0/85.0,    63.0/34.0,        84.0/221.0,    9.0/221.0,
01757     9.0/4862.0,    1.0/48620.0
01758 };
01759
01760 template< >
01761 struct pade_log_number<long double>
01762 {
01763     using array = std::array<long double, 18>;
01764     static const array number;
01765 };
01766 const typename pade_log_number<long double>::array pade_log_number<long double>::number =
01767 {
01768     0.0L,          1.0L,          8.0L,
01769     3835.0L/132.0L, 11363807.0L/122760.0L, 162981.0L/1705.0L,
01770     8365.0L/132.0L, 9036157.0L/125860.0L, 4149566.0L/849555.0L,
01771     18009875.0L/453096.0L, 44211925.0L/2718576.0L, 16973929.0L/16020180.0L,
01772     172459.0L/1068012.0L, 116317061.0L/7025382936.0L, 50747.0L/79318839600.0L,
01773     23763863.0L/614721006900.0L, 42142223.0L/14295951736466400.0L
01774 };
01775 template< >
01776 struct pade_log_denom<long double>
01777 {
01778     using array = std::array<long double, 18>;
01779     static const array denom;
01780 };
01781 const typename pade_log_denom<long double>::array pade_log_denom<long double>::denom =
01782 {
01783     1.0L,          17.0L/2.0L,          1088.0L/33.0L,
01784     850.0L/11.0L, 41650.0L/341.0L, 140777.0L/1023.0L, 1126216.0L/9889.0L,
01785     63206.0L/899.0L, 790075.0L/24273.0L, 60775.0L/5394.0L, 38896.0L/13485.0L,
01786     21658.0L/40455.0L, 21658.0L/310155.0L, 4165.0L/682341.0L, 680.0L/2047023.0L,
01787     34.0L/3411705.0L, 17.0L/129644790.0L, 1.0L/2333606220
01788 };
01789 #if defined(_GLUCAT_USE_QD)
01790 template< >
01791 struct pade_log_number<dd_real>
01792 {
01793     using array = std::array<dd_real, 22>;
01794     static const array number;
01795 };
01796 const typename pade_log_number<dd_real>::array pade_log_number<dd_real>::number =
01797 {
01798     dd_real("0"),          dd_real("1"),
01799     dd_real("21603"),/dd_real("164"),          dd_real("22781")/dd_real("492"),
01800     dd_real("978724")/dd_real("2665"),          dd_real("5492649")/dd_real("21320"),
01801     dd_real("12874933")/dd_real("39442"),          dd_real("4191605")/dd_real("10619"),
01802     dd_real("2406734")/dd_real("22755"),          dd_real("11473457")/dd_real("54612"),
01803     dd_real("30653165")/dd_real("2402928"),          dd_real("166770367")/dd_real("4004880"),
01804     dd_real("25346331")/dd_real("47074027"),          dd_real("647746389")/dd_real("215195552"),
01805     dd_real("105689791")/dd_real("15601677520"),          dd_real("278270613")/dd_real("3900419380"),
01806     dd_real("969715")/dd_real("53502994116"),          dd_real("606046475")/dd_real("1379188292768"),
01807     dd_real("118999")/dd_real("26204577562592"),          dd_real("11098301")/dd_real("26204577562592"),
01808     dd_real("18858053")/dd_real("1392249205900512960")
01809 };
01810 template< >
01811 struct pade_log_denom<dd_real>
01812 {
01813     using array = std::array<dd_real, 22>;
01814     static const array denom;
01815 };
01816 const typename pade_log_denom<dd_real>::array pade_log_denom<dd_real>::denom =
01817 {
01818     dd_real("1"),          dd_real("21")/dd_real("2"),
01819     dd_real("2100")/dd_real("41"),          dd_real("12635")/dd_real("82"),
01820     dd_real("341145")/dd_real("1066"),          dd_real("1037799")/dd_real("2132"),
01821     dd_real("11069856")/dd_real("19721"),          dd_real("9883800")/dd_real("19721"),
01822     dd_real("6918660")/dd_real("19721"),          dd_real("293930")/dd_real("1517"),
01823     dd_real("1410864")/dd_real("16687"),          dd_real("88179")/dd_real("3034"),
01824     dd_real("734825")/dd_real("94054"),          dd_real("305235")/dd_real("188108"),
01825     dd_real("348840")/dd_real("1363783"),          dd_real("40698")/dd_real("1363783"),
01826     dd_real("6783")/dd_real("2727566"),          dd_real("9975")/dd_real("70916716"),
01827     dd_real("266")/dd_real("53187537"),          dd_real("7")/dd_real("70916716"),
01828     dd_real("7")/dd_real("8155422340"),          dd_real("1")/dd_real("538257874440")
01829 };

```

```

01830     template< >
01831     struct pade_log_number<qd_real>
01832     {
01833         using array = std::array<qd_real, 34>;
01834         static const array number;
01835     };
01836     const typename pade_log_number<qd_real>::array pade_log_number<qd_real>::number =
01837     {
01838         qd_real("0"),
01839         qd_real("16"),
01840         qd_real("95201")/qd_real("780"),
01841         qd_real("30721")/qd_real("52"),
01842         qd_real("7416257")/qd_real("3640"),
01843         qd_real("1039099")/qd_real("195"),
01844         qd_real("6097772319")/qd_real("555100"),
01845         qd_real("1564058073")/qd_real("85400"),
01846         qd_real("30404640205")/qd_real("1209264"),
01847         qd_real("725351278")/qd_real("25193"),
01848         qd_real("4092322670789")/qd_real("147429436"),
01849         qd_real("4559713849589")/qd_real("201040140"),
01850         qd_real("5049361751189")/qd_real("320023080"),
01851         qd_real("74979677195")/qd_real("8000577"),
01852         qd_real("16569850691873")/qd_real("3481514244"),
01853         qd_real("1065906022369")/qd_real("515779888"),
01854         qd_real("335956770855841")/qd_real("438412904800"),
01855         qd_real("1462444287585964")/qd_real("6041877844275"),
01856         qd_real("397242326339851")/qd_real("6122436215532"),
01857         qd_real("64211291334131")/qd_real("4373168725380"),
01858         qd_real("142322343550859")/qd_real("51080680851480"),
01859         qd_real("154355972958659")/qd_real("351179680853925"),
01860         qd_real("167483568676259")/qd_real("2937139148960100"),
01861         qd_real("4230788929433")/qd_real("704913395750424"),
01862         qd_real("197968763176019")/qd_real("392923948371995600"),
01863         qd_real("10537522306718")/qd_real("319250708052246425"),
01864         qd_real("236648286272519")/qd_real("144249197475035425500"),
01865         qd_real("260715545088119")/qd_real("4375558990076074573500"),
01866         qd_real("289596255666839")/qd_real("192874640282553367199880"),
01867         qd_real("8802625510547")/qd_real("361639950529787563499775"),
01868         qd_real("373831661521439")/qd_real("1659204093030665341336967700"),
01869         qd_real("446033437968239")/qd_real("464577146048586295574350956000"),
01870         qd_real("53676090078349")/qd_real("47386868896955802148583797512000")
01871     };
01872     template< >
01873     struct pade_log_denom<qd_real>
01874     {
01875         using array = std::array<qd_real, 34>;
01876         static const array denom;
01877     };
01878     const typename pade_log_denom<qd_real>::array pade_log_denom<qd_real>::denom =
01879     {
01880         qd_real("1"),
01881         qd_real("33")/qd_real("2"),
01882         qd_real("8448")/qd_real("65"),
01883         qd_real("42284")/qd_real("65"),
01884         qd_real("211420")/qd_real("91"),
01885         qd_real("573562")/qd_real("91"),
01886         qd_real("32119472")/qd_real("2379"),
01887         qd_real("92917044")/qd_real("3965"),
01888         qd_real("603960786")/qd_real("17995"),
01889         qd_real("144626625")/qd_real("3599"),
01890         qd_real("2776831200")/qd_real("68381"),
01891         qd_real("16692542100")/qd_real("478667"),
01892         qd_real("12241197540")/qd_real("478667"),
01893         qd_real("1098569010")/qd_real("68381"),
01894         qd_real("31387686000")/qd_real("3624193"),
01895         qd_real("9939433900")/qd_real("2479711"),
01896         qd_real("67091178825")/qd_real("42155087"),
01897         qd_real("2683647153")/qd_real("4959422"),
01898         qd_real("19083713088")/qd_real("121505839"),
01899         qd_real("4708152900")/qd_real("121505839"),
01900         qd_real("941630580")/qd_real("116546417"),
01901         qd_real("88704330")/qd_real("62755763"),
01902         qd_real("12902448")/qd_real("62755763"),
01903         qd_real("1542684")/qd_real("62755763"),
01904         qd_real("6427850")/qd_real("2698497809"),
01905         qd_real("3471039")/qd_real("18889484663"),
01906         qd_real("8544096")/qd_real("774468871183"),
01907         qd_real("39556")/qd_real("79027435835"),
01908         qd_real("118668")/qd_real("7191496660985"),
01909         qd_real("10230")/qd_real("27327687311743"),
01910         qd_real("5456")/qd_real("1011124430534491"),
01911         qd_real("44")/qd_real("1011124430534491"),
01912         qd_real("11")/qd_real("70778710137414370"),
01913         qd_real("1")/qd_real("7219428434016265740")
01914     };
01915 #endif
01916 }

```

```

01884
01885 namespace glucat{
01886     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01887     static
01888     auto
01889     pade_log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> const
01890     matrix_multi<Scalar_T,LO,HI,Tune_P>
01891     {
01892         // Reference: [GW], Section 4.3, pp318-322
01893         // Reference: [CHKL]
01894         // Reference: [GL], Section 11.3, p572-576
01895         // Reference: [Z], Padel
01896
01897         using traits_t = numeric_traits<Scalar_T>;
01898         if (val == Scalar_T(0) || val.isnan())
01899             return traits_t::NaN();
01900         else
01901             return pade_approx(pade::pade_log_numer<Scalar_T>::numer,
01902                                pade::pade_log_denom<Scalar_T>::denom,
01903                                val - Scalar_T(1));
01904     }
01905
01906     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01907     static
01908     auto
01909     cascade_log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> const
01910     matrix_multi<Scalar_T,LO,HI,Tune_P>
01911     {
01912         // Reference: [CHKL]
01913         using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
01914         using traits_t = numeric_traits<Scalar_T>;
01915         if (val == Scalar_T(0) || val.isnan())
01916             return traits_t::NaN();
01917
01918         using limits_t = std::numeric_limits<Scalar_T>;
01919         static const auto epsilon = limits_t::epsilon();
01920         static const auto max_inner_norm = traits_t::pow(epsilon, 2);
01921         static const auto max_outer_norm = Scalar_T(6.0/limits_t::digits);
01922         auto Y = val;
01923         auto E = multivector_t(Scalar_T(0));
01924         Scalar_T norm_Y_1;
01925         auto pow_2_outer_step = Scalar_T(1);
01926         auto pow_4_outer_step = Scalar_T(1);
01927         int outer_step;
01928         for (outer_step = 0, norm_Y_1 = norm(Y - Scalar_T(1));
01929             outer_step != Tune_P::log_max_outer_steps && norm_Y_1 * pow_2_outer_step > max_outer_norm;
01930             ++outer_step, norm_Y_1 = norm(Y - Scalar_T(1)))
01931         {
01932             if (Y == Scalar_T(0) || Y.isnan())
01933                 return traits_t::NaN();
01934
01935             // Incomplete product form of Denman-Beavers square root iteration
01936             auto M = Y;
01937             for (auto
01938                 inner_step = 0;
01939                 inner_step != Tune_P::log_max_inner_steps &&
01940                     norm(M - Scalar_T(1)) * pow_4_outer_step > max_inner_norm;
01941                     ++inner_step)
01942                 db_step(M, Y);
01943
01944             E += (M - Scalar_T(1)) * pow_2_outer_step;
01945             pow_2_outer_step *= Scalar_T(2);
01946             pow_4_outer_step *= Scalar_T(4);
01947         }
01948         if (outer_step == Tune_P::log_max_outer_steps && norm_Y_1 * pow_2_outer_step > max_outer_norm)
01949             return traits_t::NaN();
01950         else
01951             return pade_log(Y) * pow_2_outer_step - E;
01952     }
01953
01954     template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
01955     auto
01956     matrix_log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val,
01957               const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
01958               const index_t level) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
01959     {
01960         // Scaled incomplete square root cascade and scaled Pade' approximation of log
01961         // Reference: [CHKL]
01962
01963         using traits_t = numeric_traits<Scalar_T>;
01964         if (val == Scalar_T(0) || val.isnan())
01965             return traits_t::NaN();
01966
01967         static const auto pi = traits_t::pi();
01968         const auto scr_val = val.scalar();
01969         if (val == scr_val)
01970             {
01971

```

```

01972     if (scr_val < Scalar_T(0))
01973         return i * pi + traits_t::log(-scr_val);
01974     else
01975         return traits_t::log(scr_val);
01976 }
01977
01978 // Scale val towards abs(A) == 1 or towards A == 1 as appropriate
01979 const auto max_norm = Scalar_T(1.0/9.0);
01980 const auto scale =
01981     (scr_val != Scalar_T(0) && norm(val/scr_val - Scalar_T(1)) < max_norm)
01982     ? scr_val
01983     : (scr_val < Scalar_T(0))
01984       ? -abs(val)
01985       : abs(val);
01986 if (scale == Scalar_T(0))
01987     return traits_t::NaN();
01988
01989 using multivector_t = matrix_multi<Scalar_T,LO,HI,Tune_P>;
01990 const auto log_scale = traits_t::log(traits_t::abs(scale));
01991 auto rescale = multivector_t(log_scale);
01992 if (scale < Scalar_T(0))
01993     rescale = i * pi + log_scale;
01994 const auto unitval = val/scale;
01995 if (inv(unitval).isnan())
01996     return traits_t::NaN();
01997
01998 #if defined(_GLUCAT_USE_EIGENVALUES)
01999 auto scaled_result = multivector_t();
02000 if (level == 0)
02001 {
02002     // What kind of eigenvalues does the matrix contain?
02003     auto genus = matrix::classify_eigenvalues(unitval.m_matrix);
02004     switch (genus.m_eig_case)
02005     {
02006     case matrix::neg_real_eigs:
02007         scaled_result = matrix_log(-i * unitval, i, level + 1) + i * pi/Scalar_T(2);
02008         break;
02009     case matrix::both_eigs:
02010     {
02011         const Scalar_T safe_arg = genus.m_safe_arg;
02012         scaled_result = matrix_log(exp(i*safe_arg) * unitval, i, level + 1) - i * safe_arg;
02013     }
02014         break;
02015     default:
02016         scaled_result = cascade_log(unitval);
02017         break;
02018     }
02019 }
02020 else
02021     scaled_result = cascade_log(unitval);
02022 #else
02023 auto scaled_result = cascade_log(unitval);
02024 #endif
02025 return (scaled_result.isnan())
02026     ? traits_t::NaN()
02027     : scaled_result + rescale;
02028 }
02029
02030 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
02031 auto
02032 log(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val, const matrix_multi<Scalar_T,LO,HI,Tune_P>& i,
02033 bool prechecked) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
02034 {
02035     using traits_t = numeric_traits<Scalar_T>;
02036
02037     if (val == Scalar_T(0) || val.isnan())
02038         return traits_t::NaN();
02039
02040     check_complex(val, i, prechecked);
02041
02042     switch (Tune_P::function_precision)
02043     {
02044     case precision_demoted:
02045     {
02046         using demoted_scalar_t = typename traits_t::demoted::type;
02047         using demoted_multivector_t = matrix_multi<demoted_scalar_t,LO,HI,Tune_P>;
02048
02049         const auto& demoted_val = demoted_multivector_t(val);
02050         const auto& demoted_i = demoted_multivector_t(i);
02051
02052         return matrix_log(demoted_val, demoted_i, 0);
02053     }
02054     break;
02055     case precision_promoted:
02056     {
02057         using promoted_scalar_t = typename traits_t::promoted::type;
02058         using promoted_multivector_t = matrix_multi<promoted_scalar_t,LO,HI,Tune_P>;

```

```

02059
02060     const auto& promoted_val = promoted_multivector_t(val);
02061     const auto& promoted_i = promoted_multivector_t(i);
02062
02063     return matrix_log(promoted_val, promoted_i, 0);
02064 }
02065 break;
02066 default:
02067     return matrix_log(val, i, 0);
02068 }
02069 }
02070
02071 template< typename Scalar_T, const index_t LO, const index_t HI, typename Tune_P >
02072 auto
02073 exp(const matrix_multi<Scalar_T,LO,HI,Tune_P>& val) -> const matrix_multi<Scalar_T,LO,HI,Tune_P>
02074 {
02075     using traits_t = numeric_traits<Scalar_T>;
02076     if (val.isnan())
02077         return traits_t::NaN();
02078
02079     const auto scr_val = val.scalar();
02080     if (val == scr_val)
02081         return traits_t::exp(scr_val);
02082
02083     switch (Tune_P::function_precision)
02084     {
02085     case precision_demoted:
02086     {
02087         using demoted_scalar_t = typename traits_t::demoted::type;
02088         using demoted_multivector_t = matrix_multi<demoted_scalar_t,LO,HI,Tune_P>;
02089
02090         const auto& demoted_val = demoted_multivector_t(val);
02091         return clifford_exp(demoted_val);
02092     }
02093     break;
02094     case precision_promoted:
02095     {
02096         using promoted_scalar_t = typename traits_t::promoted::type;
02097         using promoted_multivector_t = matrix_multi<promoted_scalar_t,LO,HI,Tune_P>;
02098
02099         const auto& promoted_val = promoted_multivector_t(val);
02100         return clifford_exp(promoted_val);
02101     }
02102     break;
02103     default:
02104         return clifford_exp(val);
02105     }
02106 }
02107 }
02108 }
02109 #endif // _GLUCAT_MATRIX_MULTI_IMP_H

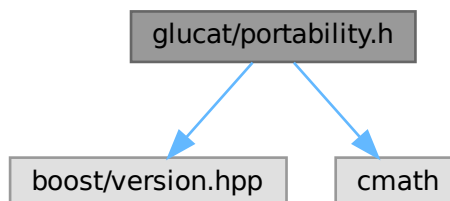
```

## 7.39 glucat/portability.h File Reference

```
#include <boost/version.hpp>
```

```
#include <cmath>
```

Include dependency graph for portability.h:



[illegible]

- `#define _GLUCAT_ISNAN(x)`
- `#define _GLUCAT_ISINF(x)`
- `#define UBLAS_ABS abs`
- `#define UBLAS_SQRT sqrt`

#### 7.39.1.1 \_GLUCAT\_ISINF

Referenced by `glucat::numeric_traits< Scalar_T >::isInf()`.

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
```

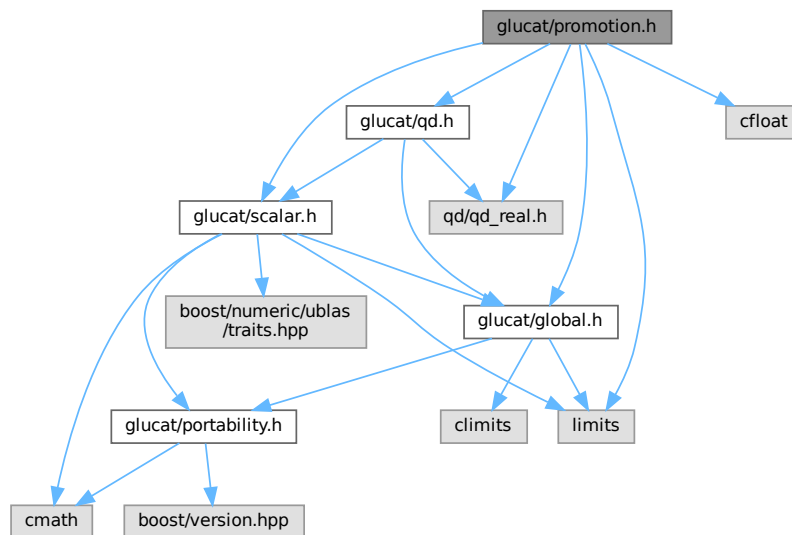
## 7.41 glucat/promotion.h File Reference

```
#include "glucat/global.h"
#include "glucat/scalar.h"
```

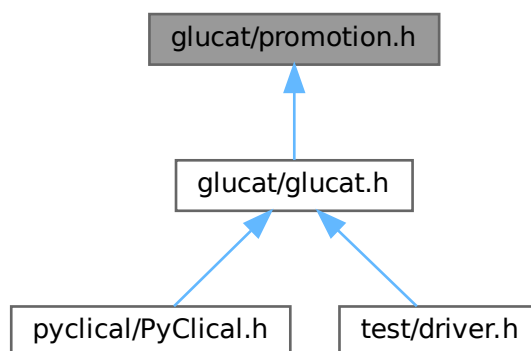


```
#include "glucat/qd.h"
#include <cfloat>
#include <limits>
#include <qd/qd_real.h>
```

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 <cfloat>
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
00062     # else
00063
00064         template<>
00065         struct
00066         numeric_traits<double>::
00067         promoted {using type = double;};
00068
00069         template<>
00070         struct
00071         numeric_traits<long double>::
00072         demoted {using type = float;};
00073
00074     # endif
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

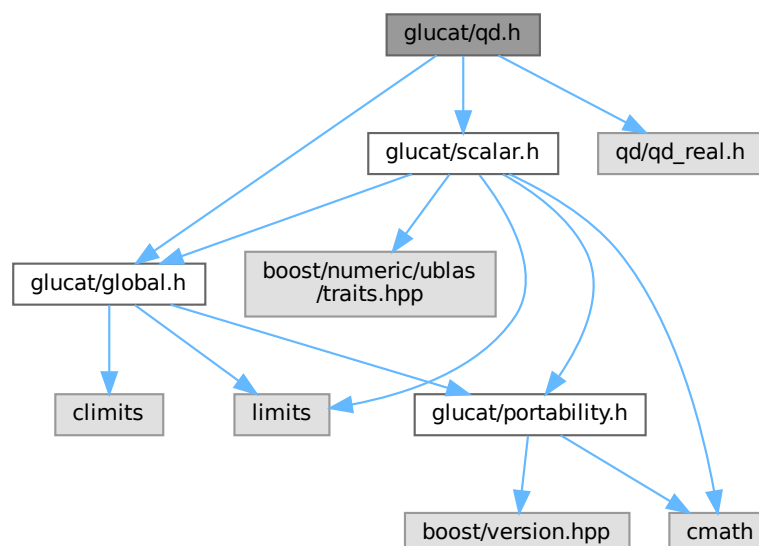
```

## 7.43 glucat/qd.h File Reference

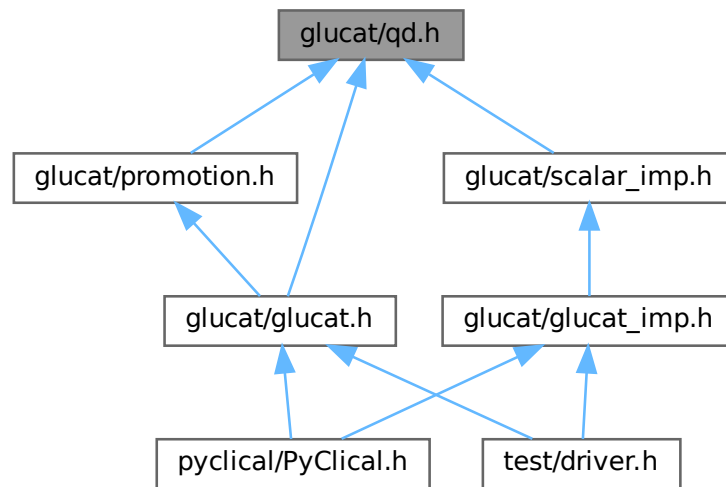
```

#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

```

```

00129         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

```

```

00223     numeric_traits<qd_real>::
00224     to_double(const qd_real& val) -> double
00225     { return ::to_double(val); }
00226
00227     template<>
00228     inline
00229     auto
00230     numeric_traits<qd_real>::
00231     fmod(const qd_real& lhs, const qd_real& rhs) -> qd_real
00232     { return ::fmod(lhs, rhs); }
00233
00234     template<>
00235     inline
00236     auto
00237     numeric_traits<qd_real>::
00238     pow(const qd_real& val, int n) -> qd_real
00239     {
00240         if (val == qd_real(0))
00241         {
00242             return
00243             (n < 0)
00244             ? NaN()
00245             : (n == 0)
00246             ? qd_real(1)
00247             : qd_real(0);
00248         }
00249         auto result = qd_real(1);
00250         auto power =
00251         (n < 0)
00252         ? qd_real(1)/val
00253         : val;
00254         for (auto
00255             k = std::abs(n);
00256             k != 0;
00257             k /= 2)
00258         {
00259             if (k % 2)
00260                 result *= power;
00261             power *= power;
00262         }
00263         return result;
00264     }
00265
00266     template<>
00267     inline
00268     auto
00269     numeric_traits<qd_real>::
00270     pi() -> qd_real
00271     { return qd_real::_pi; }
00272
00273     template<>
00274     inline
00275     auto
00276     numeric_traits<qd_real>::
00277     ln_2() -> qd_real
00278     { return qd_real::_log2; }
00279
00280     _GLUCAT_QD_F(qd_real, exp)
00281
00282     _GLUCAT_QD_F(qd_real, log)
00283
00284     _GLUCAT_QD_F(qd_real, cos)
00285
00286     _GLUCAT_QD_F(qd_real, acos)
00287
00288     _GLUCAT_QD_F(qd_real, cosh)
00289
00290     _GLUCAT_QD_F(qd_real, sinh)
00291
00292     _GLUCAT_QD_F(qd_real, sin)
00293
00294     _GLUCAT_QD_F(qd_real, asin)
00295
00296     _GLUCAT_QD_F(qd_real, tanh)
00297
00298     _GLUCAT_QD_F(qd_real, tan)
00299
00300     _GLUCAT_QD_F(qd_real, atan)
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314

```



```

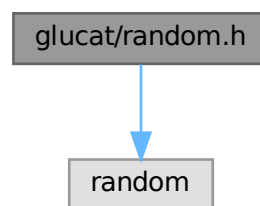
00315  _GLUCAT_QD_F(qd_real, tanh)
00316
00317  #endif // !defined(_GLUCAT_USE_QD) || !defined(QD_API)
00318
00319  } // namespace glucat
00320
00321  #endif // _GLUCAT_QD_H

```

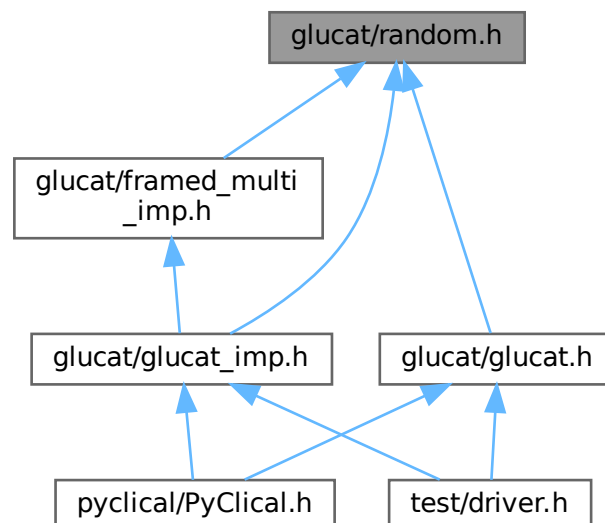
## 7.45 glucat/random.h File Reference

```
#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.](#)

```

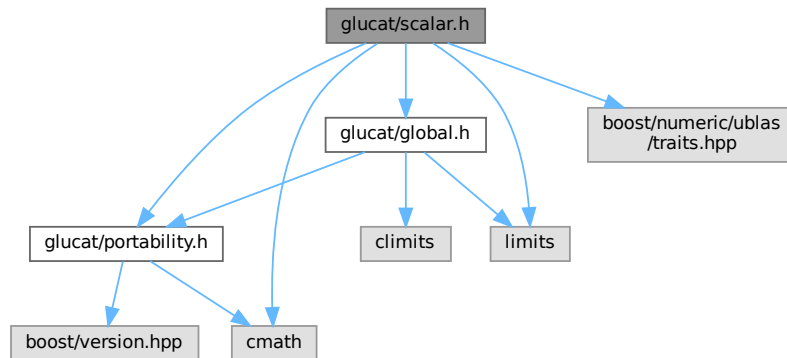
00001 #ifndef _GLUCAT_RANDOM_H
00002 #define _GLUCAT_RANDOM_H
00003 /*****
00004     GluCat : Generic library of universal Clifford algebra templates
00005     random.h : Random number generator with single instance per Scalar_T
00006     -----
00007     begin                : 2010-03-28
00008     copyright            : (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

```

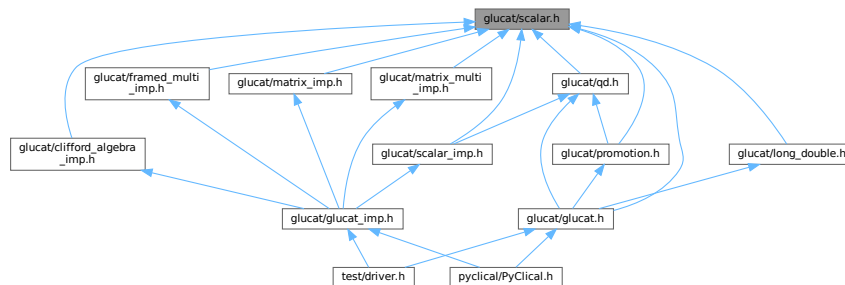
## 7.47 glucat/scalar.h File Reference

```
#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

```

```

00075     isNaN(const Scalar_T& val, bool_to_type<true>) -> bool
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

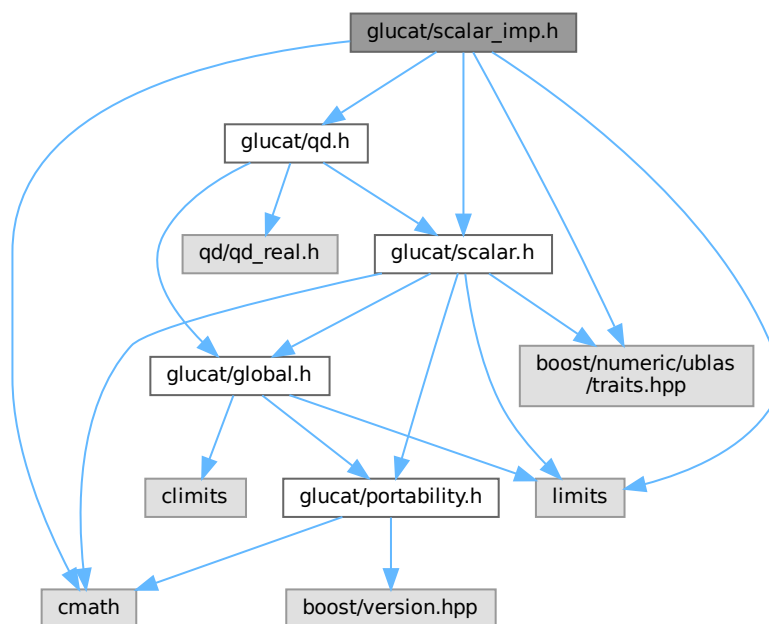
```

## 7.49 glucat/scalar\_imp.h File Reference

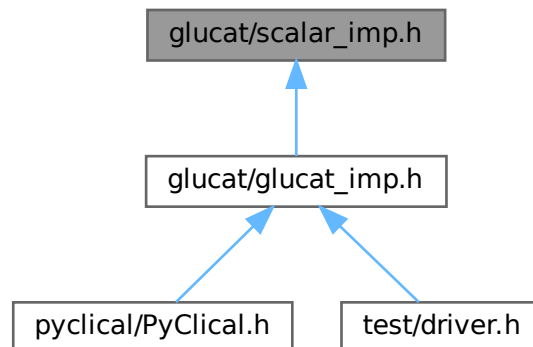
```

#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

```

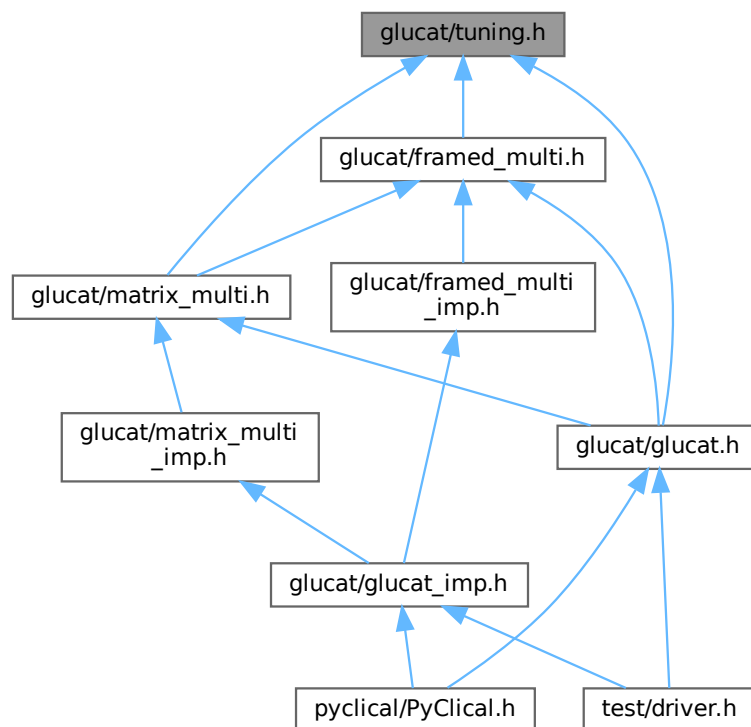
```

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         <

```

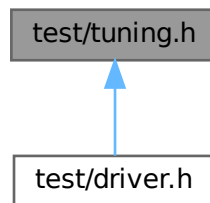
```

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(Clifford Clf)
00099         scalar_t error_squared(Clifford Lhs, Clifford Rhs, scalar_t threshold)
00100         bint approx_equal(Clifford Lhs, Clifford Rhs, scalar_t threshold, scalar_t tol)
00101         scalar_t scalar(Clifford Clf)
00102         scalar_t real(Clifford Clf)
00103         scalar_t imag(Clifford Clf)
00104         Clifford pure(Clifford Clf)
00105         Clifford even(Clifford Clf)
00106         Clifford odd(Clifford Clf)
00107         Clifford involute(Clifford Clf)
00108         Clifford reverse(Clifford Clf)
00109         Clifford conj(Clifford Clf)
00110         scalar_t norm(Clifford Clf)
00111         scalar_t abs(Clifford Clf)
00112         scalar_t max_abs(Clifford Clf)
00113         scalar_t quad(Clifford Clf)
00114         Clifford inv(Clifford Clf)
00115         Clifford pow(Clifford Clf, int m)
00116         Clifford outer_pow(Clifford Clf, int m)
00117
00118         Clifford complexifier(Clifford Clf)
00119         Clifford sqrt(Clifford 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)

```

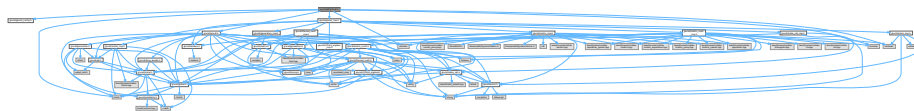
## 7.57 pyclical/PyClical.h File Reference

```

#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](#).

References [glucat::numeric\\_traits< Scalar\\_T >::to\\_double\(\)](#).

### 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 [PyClicl.h](#).

## 7.58 PyClicl.h

[Go to the documentation of this file.](#)

```
00001 /*****
00002     GluCat : Generic library of universal Clifford algebra templates
00003     PyClicl.h : C++ definitions needed by PyClicl
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 PyClicl 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>
```

```

00075 inline String clifford_to_repr(const Multivector_T& mv)
00076 {
00077     using scalar_t = typename Multivector_T::scalar_t;
00078     std::ostringstream os;
00079     os << std::setprecision(std::numeric_limits<scalar_t>::digits10 + 1);
00080     os << "clifford(\"" << mv << "\")";
00081     return os.str();
00082 }
00083
00085 template<typename Multivector_T>
00086 inline String clifford_to_str(const Multivector_T& mv)
00087 {
00088     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 namespace cga3
00098 {
00099     template<typename Multivector_T>
00100     inline Multivector_T cga3(const Multivector_T& x)
00101     {
00102         using cl = Multivector_T;
00103         using ist = typename cl::index_set_t;
00104         static const cl ninf3 = cl(ist(4)) + cl(ist(-1));
00105
00106         return (cl(ist(4)) - x) * ninf3 * (x - cl(ist(4)));
00107     }
00108
00109     template<typename Multivector_T>
00110     inline Multivector_T cga3std(const Multivector_T& X)
00111     {
00112         using cl = Multivector_T;
00113         using ist = typename cl::index_set_t;
00114         using scalar_t = typename cl::scalar_t;
00115         static const cl ninf3 = cl(ist(4)) + cl(ist(-1));
00116
00117         return scalar_t(-2.0) * X / (X & ninf3);
00118     }
00119
00120     template<typename Multivector_T>
00121     inline Multivector_T agc3(const Multivector_T& X)
00122     {
00123         using cl = Multivector_T;
00124         using ist = typename cl::index_set_t;
00125         using scalar_t = typename cl::scalar_t;
00126
00127         const cl& cga3stdX = cga3std(X);
00128         return (cl(ist(1))*cga3stdX[ist(1)] +
00129             cl(ist(2))*cga3stdX[ist(2)] +
00130             cl(ist(3))*cga3stdX[ist(3)]) / scalar_t(2.0);
00131     }
00132 }
00133
00134 // Specifications of the IndexSet and Clifford C++ classes for use with PyClical.
00135
00136 using namespace glucat;
00137 const index_t lo_ndx = DEFAULT_LO;
00138 const index_t hi_ndx = DEFAULT_HI;
00139 using IndexSet = index_set<lo_ndx, hi_ndx>;
00140
00141 using scalar_t = double;
00142 using Clifford = matrix_multi<scalar_t, lo_ndx, hi_ndx, tuning_promoted>;
00143
00144 const scalar_t epsilon = std::numeric_limits<scalar_t>::epsilon();
00145
00146 // Do not warn about unused values. This affects clang++ as well as g++.
00147
00148 #pragma GCC diagnostic ignored "-Wunused-value"
00149
00150 #if defined(__clang__)
00151 // Do not warn about unused functions. The affects clang++ only.
00152
00153 #pragma clang diagnostic ignored "-Wunused-function"
00154
00155 // Do not warn about unneeded internal declarations. The affects clang++ only.
00156
00157 #pragma clang diagnostic ignored "-Wunneeded-internal-declaration"
00158 #endif

```

## 7.59 pyclical/PyClical.pxd File Reference

### Namespaces

- namespace [PyClical](#)

## 7.60 PyClical.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 # PyClical.pxd: Basic Cython definitions for PyClical
00008 #         corresponding to C++ definitions from PyClical.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 "PyClical.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 pyclical/PyClical.pyx File Reference

### Classes

- class [PyClical.index\\_set](#)
- class [PyClical.clifford](#)



## Namespaces

- namespace [PyClical](#)

## Functions

- [PyClical.index\\_set\\_hidden\\_doctests](#) ()
- [PyClical.clifford\\_hidden\\_doctests](#) ()
- [PyClical.e](#) (obj)
- [PyClical.istpq](#) (p, q)
- [PyClical.\\_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 PyClicical 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.instance[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.instance[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.instance = new IndexSet((<index_set>other).unwrap())
00096         elif isinstance(other, numbers.Integral):
00097             self.instance = new IndexSet(<int>other)
00098         elif isinstance(other, (set, frozenset)):
00099             try:
00100                 self.instance = 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.instance = 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.instance
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.instance.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.instance.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.instance.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.instance.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.

```

```

00285
00286     >> x = index_set({1}); x &= index_set({2}); print(x)
00287     {}
00288     >> x = index_set({1,2}); x &= index_set({2}); print(x)
00289     {2}
00290     """
00291     return self.wrap( self.unwrap() & toIndexSet(rhs) )
00292
00293 def __or__(lhs, rhs):
00294     """
00295     Set union: or.
00296
00297     >> print(index_set({1}) | index_set({2}))
00298     {1,2}
00299     >> print(index_set({1,2}) | index_set({2}))
00300     {1,2}
00301     """
00302     return index_set().wrap( toIndexSet(lhs) | toIndexSet(rhs) )
00303
00304 def __ior__(self, rhs):
00305     """
00306     Set union: or.
00307
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.instance.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.instance.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.instance.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.instance.min()
00350
00351 def max(self):
00352     """
00353     Maximum member.
00354
00355     >> index_set({-1,1,2}).max()
00356     2
00357     """
00358     return self.instance.max()
00359
00360 def hash_fn(self):
00361     """
00362     Hash function.
00363     """
00364     return self.instance.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.instance.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.instance.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 cdef 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 cdef 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 cdef 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 __radd__(rhs, lhs):
00752         """
00753         Geometric sum.
00754
00755         >> print(1 + clifford("{2}"))
00756         1+{2}
00757         """
00758         return clifford().wrap( toClifford(lhs) + toClifford(rhs) )
00759
00760     def __iadd__(self, rhs):
00761         """
00762         Geometric sum.
00763
00764         >> x = clifford(1); x += clifford("{2}"); print(x)
00765         1+{2}
00766         """
00767         return self.wrap( self.unwrap() + toClifford(rhs) )
00768
00769     def __sub__(lhs, rhs):
00770         """
00771         Geometric difference.
00772
00773         >> print(clifford(1) - clifford("{2}"))
00774         1-{2}
00775         >> print(clifford("{1}") - clifford("{2}"))
00776         {1}-{2}
00777         """
00778         return clifford().wrap( toClifford(lhs) - toClifford(rhs) )
00779
00780     def __rsub__(rhs, lhs):
00781         """
00782         Geometric difference.
00783
00784         >> print(1 - clifford("{2}"))
00785         1-{2}
00786         """
00787         return clifford().wrap( toClifford(lhs) - toClifford(rhs) )
00788
00789     def __isub__(self, rhs):
00790         """
00791         Geometric difference.
00792
00793         >> x = clifford(1); x -= clifford("{2}"); print(x)
00794         1-{2}
00795         """
00796         return self.wrap( self.unwrap() - toClifford(rhs) )
00797
00798     def __mul__(lhs, rhs):
00799         """
00800         Geometric product.
00801
00802         >> print(clifford("{1}") * clifford("{2}"))
00803         {1,2}
00804         >> print(clifford(2) * clifford("{2}"))
00805         2{2}

```

```

00806         >> print(clifford("{1}") * clifford("{1,2}"))
00807         {2}
00808         """
00809         return clifford().wrap( toClifford(lhs) * toClifford(rhs) )
00810
00811     def __rmul__(rhs, lhs):
00812         """
00813         Geometric product.
00814
00815         >> print(2 * clifford("{2}"))
00816         2{2}
00817         """
00818         return clifford().wrap( toClifford(lhs) * toClifford(rhs) )
00819
00820     def __imul__(self, rhs):
00821         """
00822         Geometric product.
00823
00824         >> x = clifford(2); x *= clifford("{2}"); print(x)
00825         2{2}
00826         >> x = clifford("{1}"); x *= clifford("{2}"); print(x)
00827         {1,2}
00828         >> x = clifford("{1}"); x *= clifford("{1,2}"); print(x)
00829         {2}
00830         """
00831         return self.wrap( self.unwrap() * toClifford(rhs) )
00832
00833     def __mod__(lhs, rhs):
00834         """
00835         Contraction.
00836
00837         >> print(clifford("{1}") % clifford("{2}"))
00838         0
00839         >> print(clifford(2) % clifford("{2}"))
00840         2{2}
00841         >> print(clifford("{1}") % clifford("{1}"))
00842         1
00843         >> print(clifford("{1}") % clifford("{1,2}"))
00844         {2}
00845         """
00846         return clifford().wrap( toClifford(lhs) % toClifford(rhs) )
00847
00848     def __rmod__(rhs, lhs):
00849         """
00850         Contraction.
00851
00852         >> print(2 % clifford("{2}"))
00853         2{2}
00854         """
00855         return clifford().wrap( toClifford(lhs) % toClifford(rhs) )
00856
00857     def __imod__(self, rhs):
00858         """
00859         Contraction.
00860
00861         >> x = clifford("{1}"); x %= clifford("{2}"); print(x)
00862         0
00863         >> x = clifford(2); x %= clifford("{2}"); print(x)
00864         2{2}
00865         >> x = clifford("{1}"); x %= clifford("{1}"); print(x)
00866         1
00867         >> x = clifford("{1}"); x %= clifford("{1,2}"); print(x)
00868         {2}
00869         """
00870         return self.wrap( self.unwrap() % toClifford(rhs) )
00871
00872     def __and__(lhs, rhs):
00873         """
00874         Inner product.
00875
00876         >> print(clifford("{1}") & clifford("{2}"))
00877         0
00878         >> print(clifford(2) & clifford("{2}"))
00879         0
00880         >> print(clifford("{1}") & clifford("{1}"))
00881         1
00882         >> print(clifford("{1}") & clifford("{1,2}"))
00883         {2}
00884         """
00885         return clifford().wrap( toClifford(lhs) & toClifford(rhs) )
00886
00887     def __rand__(rhs, lhs):
00888         """
00889         Inner product.
00890
00891         >> print(2 & clifford("{2}"))
00892         0

```

```

00893         """
00894         return clifford().wrap( toClifford(lhs) & toClifford(rhs) )
00895
00896     def __iand__(self, rhs):
00897         """
00898         Inner product.
00899
00900         >> x = clifford("{1}"); x &= clifford("{2}"); print(x)
00901         0
00902         >> x = clifford(2); x &= clifford("{2}"); print(x)
00903         0
00904         >> x = clifford("{1}"); x &= clifford("{1}"); print(x)
00905         1
00906         >> x = clifford("{1}"); x &= clifford("{1,2}"); print(x)
00907         {2}
00908         """
00909         return self.wrap( self.unwrap() & toClifford(rhs) )
00910
00911     def __xor__(lhs, rhs):
00912         """
00913         Outer product.
00914
00915         >> print(clifford("{1}") ^ clifford("{2}"))
00916         {1,2}
00917         >> print(clifford(2) ^ clifford("{2}"))
00918         2{2}
00919         >> print(clifford("{1}") ^ clifford("{1}"))
00920         0
00921         >> print(clifford("{1}") ^ clifford("{1,2}"))
00922         0
00923         """
00924         return clifford().wrap( toClifford(lhs) ^ toClifford(rhs) )
00925
00926     def __rxor__(rhs, lhs):
00927         """
00928         Outer product.
00929
00930         >> print(2 ^ clifford("{2}"))
00931         2{2}
00932         """
00933         return clifford().wrap( toClifford(lhs) ^ toClifford(rhs) )
00934
00935     def __ixor__(self, rhs):
00936         """
00937         Outer product.
00938
00939         >> x = clifford("{1}"); x ^= clifford("{2}"); print(x)
00940         {1,2}
00941         >> x = clifford(2); x ^= clifford("{2}"); print(x)
00942         2{2}
00943         >> x = clifford("{1}"); x ^= clifford("{1}"); print(x)
00944         0
00945         >> x = clifford("{1}"); x ^= clifford("{1,2}"); print(x)
00946         0
00947         """
00948         return self.wrap( self.unwrap() ^ toClifford(rhs) )
00949
00950     def __truediv__(lhs, rhs):
00951         """
00952         Geometric quotient.
00953
00954         >> print(clifford("{1}") / clifford("{2}"))
00955         {1,2}
00956         >> print(clifford(2) / clifford("{2}"))
00957         2{2}
00958         >> print(clifford("{1}") / clifford("{1}"))
00959         1
00960         >> print(clifford("{1}") / clifford("{1,2}"))
00961         -{2}
00962         """
00963         return clifford().wrap( toClifford(lhs) / toClifford(rhs) )
00964
00965     def __rtruediv__(rhs, lhs):
00966         """
00967         Geometric quotient.
00968
00969         >> print(2 / clifford("{2}"))
00970         2{2}
00971         """
00972         return clifford().wrap( toClifford(lhs) / toClifford(rhs) )
00973
00974     def __idiv__(self, rhs):
00975         """
00976         Geometric quotient.
00977
00978         >> x = clifford("{1}"); x /= clifford("{2}"); print(x)
00979         {1,2}

```

```

00980         >> x = clifford(2); x /= clifford("{2}"); print(x)
00981         2{2}
00982         >> x = clifford("{1}"); x /= clifford("{1}"); print(x)
00983         1
00984         >> x = clifford("{1}"); x /= clifford("{1,2}"); print(x)
00985         -{2}
00986         """
00987         return self.wrap( self.unwrap() / toClifford(rhs) )
00988
00989     def inv(self):
00990         """
00991         Geometric multiplicative inverse.
00992
00993         >> x = clifford("{1}"); print(x.inv())
00994         {1}
00995         >> x = clifford(2); print(x.inv())
00996         0.5
00997         >> x = clifford("{1,2}"); print(x.inv())
00998         -{1,2}
00999         """
01000         return clifford().wrap( self.instance.inv() )
01001
01002     def __or__(lhs, rhs):
01003         """
01004         Transform left hand side, using right hand side as a transformation.
01005
01006         >> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); print(y|x)
01007         -{1}
01008         >> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); print(y|exp(x))
01009         -{1}
01010         """
01011         return clifford().wrap( toClifford(lhs) | toClifford(rhs) )
01012
01013     def __ior__(self, rhs):
01014         """
01015         Transform left hand side, using right hand side as a transformation.
01016
01017         >> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); y|=x; print(y)
01018         -{1}
01019         >> x=clifford("{1,2}") * pi/2; y=clifford("{1}"); y|=exp(x); print(y)
01020         -{1}
01021         """
01022         return self.wrap( self.unwrap() | toClifford(rhs) )
01023
01024     def __pow__(self, m, dummy):
01025         """
01026         Power: self to the m.
01027
01028         >> x=clifford("{1}"); print(x ** 2)
01029         1
01030         >> x=clifford("2"); print(x ** 2)
01031         4
01032         >> x=clifford("2+{1}"); print(x ** 0)
01033         1
01034         >> x=clifford("2+{1}"); print(x ** 1)
01035         2+{1}
01036         >> x=clifford("2+{1}"); print(x ** 2)
01037         5+4{1}
01038         >> i=clifford("{1,2}"); print(exp(pi/2) * (i ** i))
01039         1
01040         """
01041         return pow(self, m)
01042
01043     def pow(self, m):
01044         """
01045         Power: self to the m.
01046
01047         >> x=clifford("{1}"); print(x.pow(2))
01048         1
01049         >> x=clifford("2"); print(x.pow(2))
01050         4
01051         >> x=clifford("2+{1}"); print(x.pow(0))
01052         1
01053         >> x=clifford("2+{1}"); print(x.pow(1))
01054         2+{1}
01055         >> x=clifford("2+{1}"); print(x.pow(2))
01056         5+4{1}
01057         >> print(clifford("1+{1}+{1,2}").pow(3))
01058         1+3{1}+3{1,2}
01059         >> i=clifford("{1,2}"); print(exp(pi/2) * i.pow(i))
01060         1
01061         """
01062         if isinstance(m, numbers.Integral):
01063             return clifford().wrap( self.instance.pow(m) )
01064         else:
01065             return exp(m * log(self))
01066

```

```

01067     def outer_pow(self, m):
01068         """
01069         Outer product power.
01070
01071         >> x=clifford("2+{1}"); print(x.outer_pow(0))
01072         1
01073         >> x=clifford("2+{1}"); print(x.outer_pow(1))
01074         2+{1}
01075         >> x=clifford("2+{1}"); print(x.outer_pow(2))
01076         4+4{1}
01077         >> print(clifford("1+{1}+{1,2}").outer_pow(3))
01078         1+3{1}+3{1,2}
01079
01080         """
01081         return clifford().wrap( self.instance.outer_pow(m) )
01082
01083     def __call__(self, grade):
01084         """
01085         Pure grade-vector part.
01086
01087         >> print(clifford("{1}") (1))
01088         {1}
01089         >> print(clifford("{1}") (0))
01090         0
01091         >> print(clifford("1+{1}+{1,2}") (0))
01092         1
01093         >> print(clifford("1+{1}+{1,2}") (1))
01094         {1}
01095         >> print(clifford("1+{1}+{1,2}") (2))
01096         {1,2}
01097         >> print(clifford("1+{1}+{1,2}") (3))
01098         0
01099         """
01100         return clifford().wrap( self.instance.call(grade) )
01101
01102     def scalar(self):
01103         """
01104         Scalar part.
01105
01106         >> clifford("1+{1}+{1,2}").scalar()
01107         1.0
01108         >> clifford("{1,2}").scalar()
01109         0.0
01110         """
01111         return self.instance.scalar()
01112
01113     def pure(self):
01114         """
01115         Pure part.
01116
01117         >> print(clifford("1+{1}+{1,2}").pure())
01118         {1}+{1,2}
01119         >> print(clifford("{1,2}").pure())
01120         {1,2}
01121         """
01122         return clifford().wrap( self.instance.pure() )
01123
01124     def even(self):
01125         """
01126         Even part of multivector, sum of even grade terms.
01127
01128         >> print(clifford("1+{1}+{1,2}").even())
01129         1+{1,2}
01130         """
01131         return clifford().wrap( self.instance.even() )
01132
01133     def odd(self):
01134         """
01135         Odd part of multivector, sum of odd grade terms.
01136
01137         >> print(clifford("1+{1}+{1,2}").odd())
01138         {1}
01139         """
01140         return clifford().wrap( self.instance.odd() )
01141
01142     def vector_part(self, frm = None):
01143         """
01144         Vector part of multivector, as a Python list, with respect to frm.
01145
01146         >> print(clifford("1+2{1}+3{2}+4{1,2}").vector_part())
01147         [2.0, 3.0]
01148         >> print(clifford("1+2{1}+3{2}+4{1,2}").vector_part(index_set((-1,1,2))))
01149         [0.0, 2.0, 3.0]
01150         """
01151         error_msg_prefix = "Cannot take vector part of "
01152         cdef vector[scalar_t] vec
01153         cdef int n

```

```

01154         cdef int i
01155     try:
01156         if frm is None:
01157             vec = self.instance.vector_part()
01158         else:
01159             vec = self.instance.vector_part((<index_set>frm).unwrap())
01160         n = vec.size()
01161         lst = [0.0]*n
01162         for i in xrange(n):
01163             lst[i] = vec[i]
01164         return lst
01165     except RuntimeError as err:
01166         raise ValueError(error_msg_prefix + str(self) + " using invalid "
01167                         + repr(frm) + " as frame:\n\t"
01168                         + str(err))
01169
01170 def involute(self):
01171     """
01172     Main involution, each {i} is replaced by -{i} in each term,
01173     eg. clifford("{1}") -> -clifford("{1}").
01174
01175     >> print(clifford("{1}").involute())
01176     -{1}
01177     >> print((clifford("{2}") * clifford("{1}")).involute())
01178     -{1,2}
01179     >> print((clifford("{1}") * clifford("{2}")).involute())
01180     {1,2}
01181     >> print(clifford("1+{1}+{1,2}").involute())
01182     1-{1}+{1,2}
01183     """
01184     return clifford().wrap( self.instance.involute() )
01185
01186 def reverse(self):
01187     """
01188     Reversion, eg. clifford("{1}")*clifford("{2}") -> clifford("{2}")*clifford("{1}").
01189
01190     >> print(clifford("{1}").reverse())
01191     {1}
01192     >> print((clifford("{2}") * clifford("{1}")).reverse())
01193     {1,2}
01194     >> print((clifford("{1}") * clifford("{2}")).reverse())
01195     -{1,2}
01196     >> print(clifford("1+{1}+{1,2}").reverse())
01197     1+{1}-{1,2}
01198     """
01199     return clifford().wrap( self.instance.reverse() )
01200
01201 def conj(self):
01202     """
01203     Conjugation, reverse o involute == involute o reverse.
01204
01205     >> print((clifford("{1}")).conj())
01206     -{1}
01207     >> print((clifford("{2}") * clifford("{1}")).conj())
01208     {1,2}
01209     >> print((clifford("{1}") * clifford("{2}")).conj())
01210     -{1,2}
01211     >> print(clifford("1+{1}+{1,2}").conj())
01212     1-{1}-{1,2}
01213     """
01214     return clifford().wrap( self.instance.conj() )
01215
01216 def quad(self):
01217     """
01218     Quadratic form == (rev(x)*x)(0).
01219
01220     >> print(clifford("1+{1}+{1,2}").quad())
01221     3.0
01222     >> print(clifford("1+{-1}+{1,2}+{1,2,3}").quad())
01223     2.0
01224     """
01225     return self.instance.quad()
01226
01227 def norm(self):
01228     """
01229     Norm == sum of squares of coordinates.
01230
01231     >> clifford("1+{1}+{1,2}").norm()
01232     3.0
01233     >> clifford("1+{-1}+{1,2}+{1,2,3}").norm()
01234     4.0
01235     """
01236     return self.instance.norm()
01237
01238 def abs(self):
01239     """
01240     Absolute value: square root of norm.

```

```

01241
01242     >> clifford("1+{-1}+{1,2}+{1,2,3}").abs()
01243     2.0
01244     """
01245     return glucat.abs( self.unwrap() )
01246
01247 def max_abs(self):
01248     """
01249     Maximum of absolute values of components of multivector: multivector infinity norm.
01250
01251     >> clifford("1+{-1}+{1,2}+{1,2,3}").max_abs()
01252     1.0
01253     >> clifford("3+2{1}+{1,2}").max_abs()
01254     3.0
01255     """
01256     return self.instance.max_abs()
01257
01258 def truncated(self, limit):
01259     """
01260     Remove all terms of self with relative size smaller than limit.
01261
01262     >> clifford("1e8+{1}+1e-8{1,2}").truncated(1.0e-6)
01263     clifford("100000000")
01264     >> clifford("1e4+{1}+1e-4{1,2}").truncated(1.0e-6)
01265     clifford("10000+{1}")
01266     """
01267     return clifford().wrap( self.instance.truncated(limit) )
01268
01269 def isinf(self):
01270     """
01271     Check if a multivector contains any infinite values.
01272
01273     >> clifford().isinf()
01274     False
01275     """
01276     return self.instance.isnan()
01277
01278 def isnan(self):
01279     """
01280     Check if a multivector contains any IEEE NaN values.
01281
01282     >> clifford().isnan()
01283     False
01284     """
01285     return self.instance.isnan()
01286
01287 def frame(self):
01288     """
01289     Subalgebra generated by all generators of terms of given multivector.
01290
01291     >> print(clifford("1+3{-1}+2{1,2}+4{-2,7}").frame())
01292     {-2,-1,1,2,7}
01293     >> s=clifford("1+3{-1}+2{1,2}+4{-2,7}").frame(); type(s)
01294     <class 'PyClical.index_set'>
01295     """
01296     return index_set().wrap( self.instance.frame() )
01297
01298 def __repr__(self):
01299     """
01300     The "official" string representation of self.
01301
01302     >> clifford("1+3{-1}+2{1,2}+4{-2,7}").__repr__()
01303     'clifford("1+3{-1}+2{1,2}+4{-2,7}")'
01304     """
01305     return clifford_to_repr( self.unwrap() ).decode()
01306
01307 def __str__(self):
01308     """
01309     The "informal" string representation of self.
01310
01311     >> clifford("1+3{-1}+2{1,2}+4{-2,7}").__str__()
01312     '1+3{-1}+2{1,2}+4{-2,7}'
01313     """
01314     return clifford_to_str( self.unwrap() ).decode()
01315
01316 def clifford_hidden_doctests():
01317     """
01318     Tests for functions that Doctest cannot see.
01319
01320     For clifford.__cinit__: Construct an object of type clifford.
01321
01322     >> print(clifford(2))
01323     2
01324     >> print(clifford(2.0))
01325     2
01326     >> print(clifford(1.0e-1))
01327     0.1

```



```

01328     >> print(clifford("2"))
01329     2
01330     >> print(clifford("2{1,2,3}"))
01331     2{1,2,3}
01332     >> print(clifford(clifford("2{1,2,3}")))
01333     2{1,2,3}
01334     >> print(clifford("-{1}"))
01335     -{1}
01336     >> print(clifford(2,index_set({1,2})))
01337     2{1,2}
01338     >> print(clifford([2,3],index_set({1,2})))
01339     2{1}+3{2}
01340     >> print(clifford([1,2]))
01341     Traceback (most recent call last):
01342     ...
01343     TypeError: Cannot initialize clifford object from <class 'list'>.
01344     >> print(clifford(None))
01345     Traceback (most recent call last):
01346     ...
01347     TypeError: Cannot initialize clifford object from <class 'NoneType'>.
01348     >> print(clifford(None,[1,2]))
01349     Traceback (most recent call last):
01350     ...
01351     TypeError: Cannot initialize clifford object from (<class 'NoneType'>, <class 'list'>).
01352     >> print(clifford([1,2],[1,2]))
01353     Traceback (most recent call last):
01354     ...
01355     TypeError: Cannot initialize clifford object from (<class 'list'>, <class 'list'>).
01356     >> print(clifford(""))
01357     Traceback (most recent call last):
01358     ...
01359     ValueError: Cannot initialize clifford object from invalid string "".
01360     >> print(clifford("{}"))
01361     Traceback (most recent call last):
01362     ...
01363     ValueError: Cannot initialize clifford object from invalid string '{}'.
01364     >> print(clifford("{1}"))
01365     Traceback (most recent call last):
01366     ...
01367     ValueError: Cannot initialize clifford object from invalid string '{1'.
01368     >> print(clifford("{1}"))
01369     Traceback (most recent call last):
01370     ...
01371     ValueError: Cannot initialize clifford object from invalid string '+'.
01372     >> print(clifford("-"))
01373     Traceback (most recent call last):
01374     ...
01375     ValueError: Cannot initialize clifford object from invalid string '-'.
01376     >> print(clifford("{1}+"))
01377     Traceback (most recent call last):
01378     ...
01379     ValueError: Cannot initialize clifford object from invalid string '{1}+'.
01380
01381     For clifford.__richcmp__: Compare objects of type clifford.
01382
01383     >> clifford("{1}") == clifford("1{1}")
01384     True
01385     >> clifford("{1}") != clifford("1.0{1}")
01386     False
01387     >> clifford("{1}") != clifford("1.0")
01388     True
01389     >> clifford("{1,2}") == None
01390     False
01391     >> clifford("{1,2}") != None
01392     True
01393     >> None == clifford("{1,2}")
01394     False
01395     >> None != clifford("{1,2}")
01396     True
01397     """
01398     return
01399
01400 cpdef inline error_squared_tol(obj):
01401     """
01402     Quadratic norm error tolerance relative to a specific multivector.
01403
01404     >> print(error_squared_tol(clifford("{1}")) * 3.0 - error_squared_tol(clifford("1{1}-2{2}+3{3}")))
01405     0.0
01406     """
01407     return glucat.error_squared_tol(toClifford(obj))
01408
01409 cpdef inline error_squared(lhs, rhs, threshold):
01410     """
01411     Relative or absolute error using the quadratic norm.
01412
01413     >> err2=scalar_epsilon*scalar_epsilon
01414

```

```

01415     >> print(error_squared(clifford("{1}"), clifford("1{1}"), err2))
01416     0.0
01417     >> print(error_squared(clifford("1{1}-3{2}+4{3}"), clifford("{1}"), err2))
01418     25.0
01419     """
01420     return glucat.error_squared(toClifford(lhs), toClifford(rhs), <scalar_t>threshold)
01421
01422 cpdef inline approx_equal(lhs, rhs, threshold=None, tol=None):
01423     """
01424     Test for approximate equality of multivectors.
01425
01426     >> err2=scalar_epsilon*scalar_epsilon
01427
01428     >> print(approx_equal(clifford("{1}"), clifford("1{1}")))
01429     True
01430     >> print(approx_equal(clifford("1{1}-3{2}+4{3}"), clifford("{1}")))
01431     False
01432     >> print(approx_equal(clifford("1{1}-3{2}+4{3}+0.001"), clifford("1{1}-3{2}+4{3}"), err2, err2))
01433     False
01434     >> print(approx_equal(clifford("1{1}-3{2}+4{3}+1.0e-30"), clifford("1{1}-3{2}+4{3}"), err2, err2))
01435     True
01436     """
01437     threshold = error_squared_tol(rhs) if threshold is None else threshold
01438     tol = error_squared_tol(rhs) if tol is None else tol
01439     return glucat.approx_equal(toClifford(lhs), toClifford(rhs), <scalar_t>threshold, <scalar_t>tol)
01440
01441 cpdef inline inv(obj):
01442     """
01443     Geometric multiplicative inverse.
01444
01445     >> print(inv(clifford("{1}")))
01446     {1}
01447     >> print(inv(clifford("{-1}")))
01448     -{-1}
01449     >> print(inv(clifford("{-2,-1}")))
01450     -{-2,-1}
01451     >> print(inv(clifford("{-1}+{1}")))
01452     nan
01453     """
01454     return clifford(obj).inv()
01455
01456 cpdef inline scalar(obj):
01457     """
01458     Scalar part.
01459
01460     >> scalar(clifford("1+{1}+{1,2}"))
01461     1.0
01462     >> scalar(clifford("{1,2}"))
01463     0.0
01464     """
01465     return clifford(obj).scalar()
01466
01467 cpdef inline real(obj):
01468     """
01469     Real part: synonym for scalar part.
01470
01471     >> real(clifford("1+{1}+{1,2}"))
01472     1.0
01473     >> real(clifford("{1,2}"))
01474     0.0
01475     """
01476     return clifford(obj).scalar()
01477
01478 cpdef inline imag(obj):
01479     """
01480     Imaginary part: deprecated (always 0).
01481
01482     >> imag(clifford("1+{1}+{1,2}"))
01483     0.0
01484     >> imag(clifford("{1,2}"))
01485     0.0
01486     """
01487     return 0.0
01488
01489 cpdef inline pure(obj):
01490     """
01491     Pure part
01492
01493     >> print(pure(clifford("1+{1}+{1,2}")))
01494     {1}+{1,2}
01495     >> print(pure(clifford("{1,2}")))
01496     {1,2}
01497     """
01498     return clifford(obj).pure()
01499
01500 cpdef inline even(obj):
01501     """

```

```

01502     Even part of multivector, sum of even grade terms.
01503
01504     >> print(even(clifford("1+{1}+{1,2}")))
01505     1+{1,2}
01506     """
01507     return clifford(obj).even()
01508
01509 cpdef inline odd(obj):
01510     """
01511     Odd part of multivector, sum of odd grade terms.
01512
01513     >> print(odd(clifford("1+{1}+{1,2}")))
01514     {1}
01515     """
01516     return clifford(obj).odd()
01517
01518 cpdef inline involute(obj):
01519     """
01520     Main involution, each {i} is replaced by -{i} in each term, eg. {1}*{2} -> (-{2})*(-{1})
01521
01522     >> print(involute(clifford("{1}")))
01523     -{1}
01524     >> print(involute(clifford("{2}") * clifford("{1}")))
01525     -{1,2}
01526     >> print(involute(clifford("{1}") * clifford("{2}")))
01527     {1,2}
01528     >> print(involute(clifford("1+{1}+{1,2}")))
01529     1-{1}+{1,2}
01530     """
01531     return clifford(obj).involute()
01532
01533 cpdef inline reverse(obj):
01534     """
01535     Reversion, eg. {1}*{2} -> {2}*{1}
01536
01537     >> print(reverse(clifford("{1}")))
01538     {1}
01539     >> print(reverse(clifford("{2}") * clifford("{1}")))
01540     {1,2}
01541     >> print(reverse(clifford("{1}") * clifford("{2}")))
01542     -{1,2}
01543     >> print(reverse(clifford("1+{1}+{1,2}")))
01544     1+{1}-{1,2}
01545     """
01546     return clifford(obj).reverse()
01547
01548 cpdef inline conj(obj):
01549     """
01550     Conjugation, reverse o involute == involute o reverse.
01551
01552     >> print(conj(clifford("{1}")))
01553     -{1}
01554     >> print(conj(clifford("{2}") * clifford("{1}")))
01555     {1,2}
01556     >> print(conj(clifford("{1}") * clifford("{2}")))
01557     -{1,2}
01558     >> print(conj(clifford("1+{1}+{1,2}")))
01559     1-{1}-{1,2}
01560     """
01561     return clifford(obj).conj()
01562
01563 cpdef inline quad(obj):
01564     """
01565     Quadratic form == (rev(x)*x)(0).
01566
01567     >> print(quad(clifford("1+{1}+{1,2}")))
01568     3.0
01569     >> print(quad(clifford("1+{-1}+{1,2}+{1,2,3}")))
01570     2.0
01571     """
01572     return clifford(obj).quad()
01573
01574 cpdef inline norm(obj):
01575     """
01576     norm == sum of squares of coordinates.
01577
01578     >> norm(clifford("1+{1}+{1,2}"))
01579     3.0
01580     >> norm(clifford("1+{-1}+{1,2}+{1,2,3}"))
01581     4.0
01582     """
01583     return clifford(obj).norm()
01584
01585 cpdef inline abs(obj):
01586     """
01587     Absolute value of multivector: multivector 2-norm.
01588

```

```

01589     >> abs(clifford("1+{-1}+{1,2}+{1,2,3}"))
01590     2.0
01591     """
01592     return glucat.abs(toClifford(obj))
01593
01594 cpdef inline max_abs(obj):
01595     """
01596     Maximum absolute value of coordinates multivector: multivector infinity-norm.
01597
01598     >> max_abs(clifford("1+{-1}+{1,2}+{1,2,3}"))
01599     1.0
01600     >> max_abs(clifford("3+2{1}+{1,2}"))
01601     3.0
01602     """
01603
01604     return glucat.max_abs(toClifford(obj))
01605
01606 cpdef inline pow(obj, m):
01607     """
01608     Integer power of multivector: obj to the m.
01609
01610     >> x=clifford("{1}"); print(pow(x,2))
01611     1
01612     >> x=clifford("2"); print(pow(x,2))
01613     4
01614     >> x=clifford("2+{1}"); print(pow(x,0))
01615     1
01616     >> x=clifford("2+{1}"); print(pow(x,1))
01617     2+{1}
01618     >> x=clifford("2+{1}"); print(pow(x,2))
01619     5+4{1}
01620     >> print(pow(clifford("1+{1}+{1,2}"),3))
01621     1+3{1}+3{1,2}
01622     >> i=clifford("{1,2}"); print(exp(pi/2) * pow(i, i))
01623     1
01624     """
01625     try:
01626         math.pow(obj, m)
01627     except:
01628         return clifford(obj).pow(m)
01629
01630 cpdef inline outer_pow(obj, m):
01631     """
01632     Outer product power of multivector.
01633
01634     >> print(outer_pow(clifford("1+{1}+{1,2}"),3))
01635     1+3{1}+3{1,2}
01636     """
01637     return clifford(obj).outer_pow(m)
01638
01639 cpdef inline complexifier(obj):
01640     """
01641     Square root of -1 which commutes with all members of the frame of the given multivector.
01642
01643     >> print(complexifier(clifford(index_set({1}))))
01644     {1,2,3}
01645     >> print(complexifier(clifford(index_set({-1}))))
01646     {-1}
01647     >> print(complexifier(index_set({1})))
01648     {1,2,3}
01649     >> print(complexifier(index_set({-1})))
01650     {-1}
01651     """
01652     return clifford().wrap( glucat.complexifier(toClifford(obj)) )
01653
01654 cpdef inline sqrt(obj, i = None):
01655     """
01656     Square root of multivector with optional complexifier.
01657
01658     >> print(sqrt(-1))
01659     {-1}
01660     >> print(sqrt(clifford("2{-1}")))
01661     1+{-1}
01662     >> j=sqrt(-1,complexifier(index_set({1}))); print(j); print(j*j)
01663     {1,2,3}
01664     -1
01665     >> j=sqrt(-1,"{1,2,3}"); print(j); print(j*j)
01666     {1,2,3}
01667     -1
01668     """
01669     if not (i is None):
01670         return clifford().wrap( glucat.sqrt(toClifford(obj), toClifford(i)) )
01671     else:
01672         try:
01673             return math.sqrt(obj)
01674         except:
01675             return clifford().wrap( glucat.sqrt(toClifford(obj)) )

```

```

01676
01677 cpdef inline exp(obj):
01678     """
01679     Exponential of multivector.
01680
01681     >> x=clifford("{1,2}") * pi/4; print(exp(x))
01682     0.7071+0.7071{1,2}
01683     >> x=clifford("{1,2}") * pi/2; print(exp(x))
01684     {1,2}
01685     """
01686     try:
01687         return math.exp(obj)
01688     except:
01689         return clifford().wrap( glucat.exp(toClifford(obj)) )
01690
01691 cpdef inline log(obj,i = None):
01692     """
01693     Natural logarithm of multivector with optional complexifier.
01694
01695     >> x=clifford("{-1}"); print((log(x,"{-1}") * 2/pi))
01696     {-1}
01697     >> x=clifford("{1,2}"); print((log(x,"{1,2,3}") * 2/pi))
01698     {1,2}
01699     >> x=clifford("{1,2}"); print((log(x) * 2/pi))
01700     {1,2}
01701     >> x=clifford("{1,2}"); print((log(x,"{1,2}") * 2/pi))
01702     Traceback (most recent call last):
01703     ...
01704     RuntimeError: check_complex(val, i): i is not a valid complexifier for val
01705     """
01706     if not (i is None):
01707         return clifford().wrap( glucat.log(toClifford(obj), toClifford(i)) )
01708     else:
01709         try:
01710             return math.log(obj)
01711         except:
01712             return clifford().wrap( glucat.log(toClifford(obj)) )
01713
01714 cpdef inline cos(obj,i = None):
01715     """
01716     Cosine of multivector with optional complexifier.
01717
01718     >> x=clifford("{1,2}"); print(cos(acos(x),"{1,2,3}"))
01719     {1,2}
01720     >> x=clifford("{1,2}"); print(cos(acos(x)))
01721     {1,2}
01722     """
01723     if not (i is None):
01724         return clifford().wrap( glucat.cos(toClifford(obj), toClifford(i)) )
01725     else:
01726         try:
01727             return math.cos(obj)
01728         except:
01729             return clifford().wrap( glucat.cos(toClifford(obj)) )
01730
01731 cpdef inline acos(obj,i = None):
01732     """
01733     Inverse cosine of multivector with optional complexifier.
01734
01735     >> x=clifford("{1,2}"); print(cos(acos(x),"{1,2,3}"))
01736     {1,2}
01737     >> x=clifford("{1,2}"); print(cos(acos(x),"{-1,1,2,3,4}"))
01738     {1,2}
01739     >> print(acos(0) / pi)
01740     0.5
01741     >> x=clifford("{1,2}"); print(cos(acos(x)))
01742     {1,2}
01743     """
01744     if not (i is None):
01745         return clifford().wrap( glucat.acos(toClifford(obj), toClifford(i)) )
01746     else:
01747         try:
01748             return math.acos(obj)
01749         except:
01750             return clifford().wrap( glucat.acos(toClifford(obj)) )
01751
01752 cpdef inline cosh(obj):
01753     """
01754     Hyperbolic cosine of multivector.
01755
01756     >> x=clifford("{1,2}") * pi; print(cosh(x))
01757     -1
01758     >> x=clifford("{1,2,3}"); print(cosh(acosh(x)))
01759     {1,2,3}
01760     >> x=clifford("{1,2}"); print(cosh(acosh(x)))
01761     {1,2}
01762     """

```

```

01763     try:
01764         return math.cosh(obj)
01765     except:
01766         return clifford().wrap( glucat.cosh(toClifford(obj)) )
01767
01768 cpdef inline acosh(obj,i = None):
01769     """
01770     Inverse hyperbolic cosine of multivector with optional complexifier.
01771
01772     >> print(acosh(0,"{-2,-1,1}"))
01773     1.571{-2,-1,1}
01774     >> x=clifford("{1,2,3}"); print(cosh(acosh(x,"{-1,1,2,3,4}")))
01775     {1,2,3}
01776     >> print(acosh(0))
01777     1.571{-1}
01778     >> x=clifford("{1,2,3}"); print(cosh(acosh(x)))
01779     {1,2,3}
01780     >> x=clifford("{1,2}"); print(cosh(acosh(x)))
01781     {1,2}
01782     """
01783     if not (i is None):
01784         return clifford().wrap( glucat.acosh(toClifford(obj), toClifford(i)) )
01785     else:
01786         try:
01787             return math.acosh(obj)
01788         except:
01789             return clifford().wrap( glucat.acosh(toClifford(obj)) )
01790
01791 cpdef inline sin(obj,i = None):
01792     """
01793     Sine of multivector with optional complexifier.
01794
01795     >> s="{1}"; x=clifford(s); print(asin(sin(x,s),s))
01796     {-1}
01797     >> s="{1}"; x=clifford(s); print(asin(sin(x,s),"{-2,-1,1}"))
01798     {-1}
01799     >> x=clifford("{1,2,3}"); print(asin(sin(x)))
01800     {1,2,3}
01801     """
01802     if not (i is None):
01803         return clifford().wrap( glucat.sin(toClifford(obj), toClifford(i)) )
01804     else:
01805         try:
01806             return math.sin(obj)
01807         except:
01808             return clifford().wrap( glucat.sin(toClifford(obj)) )
01809
01810 cpdef inline asin(obj,i = None):
01811     """
01812     Inverse sine of multivector with optional complexifier.
01813
01814     >> s="{1}"; x=clifford(s); print(asin(sin(x,s),s))
01815     {-1}
01816     >> s="{1}"; x=clifford(s); print(asin(sin(x,s),"{-2,-1,1}"))
01817     {-1}
01818     >> print(asin(1) / pi)
01819     0.5
01820     >> x=clifford("{1,2,3}"); print(asin(sin(x)))
01821     {1,2,3}
01822     """
01823     if not (i is None):
01824         return clifford().wrap( glucat.asin(toClifford(obj), toClifford(i)) )
01825     else:
01826         try:
01827             return math.asin(obj)
01828         except:
01829             return clifford().wrap( glucat.asin(toClifford(obj)) )
01830
01831 cpdef inline sinh(obj):
01832     """
01833     Hyperbolic sine of multivector.
01834
01835     >> x=clifford("{1,2}") * pi/2; print(sinh(x))
01836     {1,2}
01837     >> x=clifford("{1,2}") * pi/6; print(sinh(x))
01838     0.5{1,2}
01839     """
01840     try:
01841         return math.sinh(obj)
01842     except:
01843         return clifford().wrap( glucat.sinh(toClifford(obj)) )
01844
01845 cpdef inline asinh(obj,i = None):
01846     """
01847     Inverse hyperbolic sine of multivector with optional complexifier.
01848
01849     >> x=clifford("{1,2}"); print(asinh(x,"{1,2,3}") * 2/pi)

```

```

01850     {1,2}
01851     >> x=clifford("{1,2}"); print(asinh(x) * 2/pi)
01852     {1,2}
01853     >> x=clifford("{1,2}") / 2; print(asinh(x) * 6/pi)
01854     {1,2}
01855     """
01856     if not (i is None):
01857         return clifford().wrap( glucat.asinh(toClifford(obj), toClifford(i)) )
01858     else:
01859         try:
01860             return math.asinh(obj)
01861         except:
01862             return clifford().wrap( glucat.asinh(toClifford(obj)) )
01863
01864 cpdef inline tan(obj,i = None):
01865     """
01866     Tangent of multivector with optional complexifier.
01867
01868     >> x=clifford("{1,2}"); print(tan(x,"{1,2,3}"))
01869     0.7616{1,2}
01870     >> x=clifford("{1,2}"); print(tan(x))
01871     0.7616{1,2}
01872     """
01873     if not (i is None):
01874         return clifford().wrap( glucat.tan(toClifford(obj), toClifford(i)) )
01875     else:
01876         try:
01877             return math.tan(obj)
01878         except:
01879             return clifford().wrap( glucat.tan(toClifford(obj)) )
01880
01881 cpdef inline atan(obj,i = None):
01882     """
01883     Inverse tangent of multivector with optional complexifier.
01884
01885     >> s=index_set({1,2,3}); x=clifford("{1}"); print(tan(atan(x,s),s))
01886     {1}
01887     >> x=clifford("{1}"); print(tan(atan(x)))
01888     {1}
01889     """
01890     if not (i is None):
01891         return clifford().wrap( glucat.atan(toClifford(obj), toClifford(i)) )
01892     else:
01893         try:
01894             return math.atan(obj)
01895         except:
01896             return clifford().wrap( glucat.atan(toClifford(obj)) )
01897
01898 cpdef inline tanh(obj):
01899     """
01900     Hyperbolic tangent of multivector.
01901
01902     >> x=clifford("{1,2}") * pi/4; print(tanh(x))
01903     {1,2}
01904     """
01905     try:
01906         return math.tanh(obj)
01907     except:
01908         return clifford().wrap( glucat.tanh(toClifford(obj)) )
01909
01910 cpdef inline atanh(obj,i = None):
01911     """
01912     Inverse hyperbolic tangent of multivector with optional complexifier.
01913
01914     >> s=index_set({1,2,3}); x=clifford("{1,2}"); print(tanh(atanh(x,s)))
01915     {1,2}
01916     >> x=clifford("{1,2}"); print(tanh(atanh(x)))
01917     {1,2}
01918     """
01919     if not (i is None):
01920         return clifford().wrap( glucat.atanh(toClifford(obj), toClifford(i)) )
01921     else:
01922         try:
01923             return math.atanh(obj)
01924         except:
01925             return clifford().wrap( glucat.atanh(toClifford(obj)) )
01926
01927 cpdef inline random_clifford(index_set ixt, fill = 1.0):
01928     """
01929     Random multivector within a frame.
01930
01931     >> print(random_clifford(index_set({-3,-1,2})).frame())
01932     {-3,-1,2}
01933     """
01934     return clifford().wrap( clifford().instance.random(ixt.unwrap(), <scalar_t>fill) )
01935
01936 cpdef inline cga3(obj):

```

```

01937     """
01938     Convert Euclidean 3D multivector to Conformal Geometric Algebra using Doran and Lasenby
definition.
01939
01940     >> x=clifford("2{1}+9{2}+{3}"); print(cga3(x))
01941     87{-1}+4{1}+18{2}+2{3}+85{4}
01942     """
01943     return clifford().wrap( glucat.cga3(toClifford(obj)) )
01944
01945 cpdef inline cga3std(obj):
01946     """
01947     Convert CGA3 null vector to standard conformal null vector using Doran and Lasenby definition.
01948
01949     >> x=clifford("2{1}+9{2}+{3}"); print(cga3std(cga3(x)))
01950     87{-1}+4{1}+18{2}+2{3}+85{4}
01951     >> x=clifford("2{1}+9{2}+{3}"); print(cga3std(cga3(x))-cga3(x))
01952     0
01953     """
01954     return clifford().wrap( glucat.cga3std(toClifford(obj)) )
01955
01956 cpdef inline agc3(obj):
01957     """
01958     Convert CGA3 null vector to Euclidean 3D vector using Doran and Lasenby definition.
01959
01960     >> x=clifford("2{1}+9{2}+{3}"); print(agc3(cga3(x)))
01961     2{1}+9{2}+{3}
01962     >> x=clifford("2{1}+9{2}+{3}"); print(agc3(cga3(x))-x)
01963     0
01964     """
01965     return clifford().wrap( glucat.agc3(toClifford(obj)) )
01966
01967 # Some abbreviations.
01968 scalar_epsilon = epsilon
01969
01970 pi = atan(clifford(1.0)) * 4.0
01971 tau = atan(clifford(1.0)) * 8.0
01972
01973 cl = clifford
01974 """
01975 Abbreviation for clifford.
01976
01977 >> print(cl(2))
01978 2
01979 >> print(cl(2.0))
01980 2
01981 >> print(cl(5.0e-1))
01982 0.5
01983 >> print(cl("2"))
01984 2
01985 >> print(cl("2{1,2,3}"))
01986 2{1,2,3}
01987 >> print(cl(cl("2{1,2,3}")))
01988 2{1,2,3}
01989 """
01990
01991 ist = index_set
01992 """
01993 Abbreviation for index_set.
01994
01995 >> print(ist("{1,2,3}"))
01996 {1,2,3}
01997 """
01998
01999 def e(obj):
02000     """
02001     Abbreviation for clifford(index_set(obj)).
02002
02003     >> print(e(1))
02004     {1}
02005     >> print(e(-1))
02006     {-1}
02007     >> print(e(0))
02008     1
02009     """
02010     return clifford(index_set(obj))
02011
02012 def istpq(p, q):
02013     """
02014     Abbreviation for index_set({-q,...p}).
02015
02016     >> print(istpq(2,3))
02017     {-3,-2,-1,1,2}
02018     """
02019     return index_set(set(range(-q,p+1)))
02020
02021 ninf3 = e(4) + e(-1) # Null infinity point in 3D Conformal Geometric Algebra [DL].
02022 nbar3 = e(4) - e(-1) # Null bar point in 3D Conformal Geometric Algebra [DL].

```



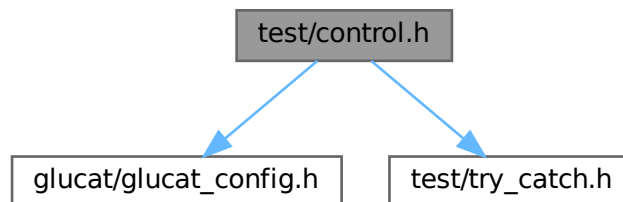
```
02023
02024 # Doctest interface.
02025 def _test():
02026     import PyClical, doctest
02027     return doctest.testmod(PyClical)
02028
02029 if __name__ == "__main__":
02030     _test()
```

## 7.63 test/control.h File Reference

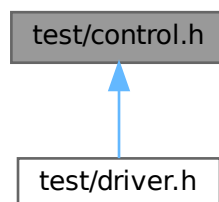
```
#include "glucat/glucat_config.h"
```

```
#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`

## 7.64 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())
? try_catch(f)
: (*f)();
00140     else
00141         return 1;
00142 }
00143
00144 inline
00145 int
00146 control_t::
00147 call(intintfn f, int arg) const
00148 {
00149     if (valid())
00150         return (catch_exceptions())
? try_catch(f, arg)
: (*f)(arg);
00151     else
00152         return 1;
00153 }
00154 }
00155 }
00156 #endif // _GLUCAT_CONTROL_H

```

## 7.65 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:



## 7.66 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

```

## 7.67 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.*

## 7.68 timing.h

[Go to the documentation of this file.](#)

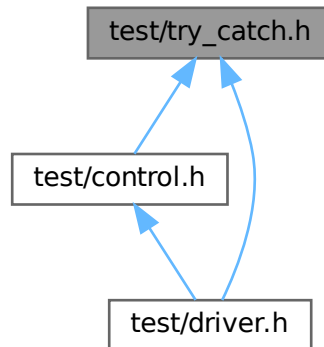
```

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         const double MS_PER_SEC = 1000.0;
00039
00040         const double MS_PER_CLOCK = MS_PER_SEC / double(CLOCKS_PER_SEC);
00041
00042         const int EXTRA_TRIALS = 2;
00043
00044         inline
00045         static
00046         double
00047         elapsed(clock_t cpu_time)
00048         { return double(clock() - cpu_time) * MS_PER_CLOCK; }
00049     }
00050 }
00051 #endif // GLUCAT_TEST_TIMING_H

```

## 7.69 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.*

## 7.70 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

```

## 7.71 test/undefine.h File Reference

## 7.72 undefine.h

[Go to the documentation of this file.](#)

```

00001 #ifndef GLUCAT_TEST_UNDEFINE_H
00002 #define GLUCAT_TEST_UNDEFINE_H
00003 /*****
00004     GluCat : Generic library of universal Clifford algebra templates
00005     undefine.h : Undefine preprocessor macro names that control test tuning
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 // Undefine tuning policy constants
00035 #undef _GLUCAT_TEST_TUNING_SLOW
00036 #undef _GLUCAT_TEST_TUNING_NAIVE
00037 #undef _GLUCAT_TEST_TUNING_FAST
00038 #undef _GLUCAT_TEST_TUNING_PROMOTED
00039 #undef _GLUCAT_TEST_TUNING_DEMOTED
00040
00041 #endif // GLUCAT_TEST_UNDEFINE_H
```



# Index

`_GLUCAT_CLIFFORD_ALGEBRA_OPERATIONS`  
    `clifford_algebra.h`, [257](#)

`_GLUCAT_CTAssert`  
    `global.h`, [327](#)  
    `glucat`, [24](#)  
    `tuning.h`, [423](#)

`_GLUCAT_HASH_N`  
    `framed_multi_imp.h`, [298](#)

`_GLUCAT_HASH_SIZE_T`  
    `framed_multi_imp.h`, [298](#)

`_GLUCAT_ISINF`  
    `portability.h`, [403](#)

`_GLUCAT_ISNAN`  
    `portability.h`, [403](#)

`__add__`  
    `PyClical.clifford`, [90](#)

`__and__`  
    `PyClical.clifford`, [90](#)  
    `PyClical.index_set`, [176](#)

`__call__`  
    `PyClical.clifford`, [91](#)

`__cinit__`  
    `PyClical.clifford`, [91](#)  
    `PyClical.index_set`, [176](#)

`__contains__`  
    `PyClical.clifford`, [92](#)  
    `PyClical.index_set`, [176](#)

`__dealloc__`  
    `PyClical.clifford`, [92](#)  
    `PyClical.index_set`, [177](#)

`__getitem__`  
    `PyClical.clifford`, [92](#)  
    `PyClical.index_set`, [177](#)

`__iadd__`  
    `PyClical.clifford`, [93](#)

`__iand__`  
    `PyClical.clifford`, [93](#)  
    `PyClical.index_set`, [177](#)

`__idiv__`  
    `PyClical.clifford`, [93](#)

`__imod__`  
    `PyClical.clifford`, [94](#)

`__imul__`  
    `PyClical.clifford`, [94](#)

`__invert__`  
    `PyClical.index_set`, [178](#)

`__ior__`  
    `PyClical.clifford`, [94](#)  
    `PyClical.index_set`, [178](#)

`__isub__`  
    `PyClical.clifford`, [95](#)

`__iter__`  
    `PyClical.clifford`, [95](#)  
    `PyClical.index_set`, [178](#)

`__ixor__`  
    `PyClical.clifford`, [95](#)  
    `PyClical.index_set`, [178](#)

`__mod__`  
    `PyClical.clifford`, [95](#)

`__mul__`  
    `PyClical.clifford`, [96](#)

`__neg__`  
    `PyClical.clifford`, [96](#)

`__or__`  
    `PyClical.clifford`, [96](#)  
    `PyClical.index_set`, [179](#)

`__pos__`  
    `PyClical.clifford`, [97](#)

`__pow__`  
    `PyClical.clifford`, [97](#)

`__radd__`  
    `PyClical.clifford`, [97](#)

`__rand__`  
    `PyClical.clifford`, [98](#)

`__repr__`  
    `PyClical.clifford`, [98](#)  
    `PyClical.index_set`, [179](#)

`__richcmp__`  
    `PyClical.clifford`, [98](#)  
    `PyClical.index_set`, [179](#)

`__rmod__`  
    `PyClical.clifford`, [99](#)

`__rmul__`  
    `PyClical.clifford`, [99](#)

`__rsub__`  
    `PyClical.clifford`, [99](#)

`__rtruediv__`  
    `PyClical.clifford`, [100](#)

`__rxor__`  
    `PyClical.clifford`, [100](#)

`__setitem__`  
    `PyClical.index_set`, [180](#)

`__str__`  
    `PyClical.clifford`, [100](#)  
    `PyClical.index_set`, [180](#)

`__sub__`  
    `PyClical.clifford`, [100](#)

`__truediv__`

- PyClical.clifford, 101
- \_\_version\_\_
  - PyClical, 81
- \_\_xor\_\_
  - PyClical.clifford, 101
  - PyClical.index\_set, 180
- \_test
  - PyClical, 77
- ~basis\_table
  - glucat::basis\_table< Scalar\_T, LO, HI, Matrix\_T >, 86
- ~clifford\_algebra
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 112
- ~control\_t
  - glucat::control\_t, 125
- ~framed\_multi
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 142
- ~generator\_table
  - glucat::gen::generator\_table< Matrix\_T >, 153
- ~glucat\_error
  - glucat::glucat\_error, 158
- ~matrix\_multi
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 192
- ~random\_generator
  - glucat::random\_generator< Scalar\_T >, 235
- ~reference
  - glucat::index\_set< LO, HI >::reference, 238
- ~var\_term
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::var\_term, 247
- abs
  - glucat, 24
  - glucat::numeric\_traits< Scalar\_T >, 206
  - PyClical.clifford, 101
- acos
  - glucat, 25
  - glucat::numeric\_traits< Scalar\_T >, 206
- acosh
  - glucat, 25
- agc3
  - cga3, 9
- approx\_equal
  - glucat, 26
- are\_same
  - glucat::compare\_types< LHS\_T, RHS\_T >, 123
  - glucat::compare\_types< T, T >, 123
- array
  - pade::pade\_log\_denom< dd\_real >, 217
  - pade::pade\_log\_denom< float >, 218
  - pade::pade\_log\_denom< long double >, 218
  - pade::pade\_log\_denom< qd\_real >, 219
  - pade::pade\_log\_denom< Scalar\_T >, 216
  - pade::pade\_log\_number< dd\_real >, 221
  - pade::pade\_log\_number< float >, 222
  - pade::pade\_log\_number< long double >, 223
  - pade::pade\_log\_number< qd\_real >, 223
  - pade::pade\_log\_number< Scalar\_T >, 220
  - pade::pade\_sqrt\_denom< dd\_real >, 225
  - pade::pade\_sqrt\_denom< float >, 226
  - pade::pade\_sqrt\_denom< long double >, 227
  - pade::pade\_sqrt\_denom< qd\_real >, 228
  - pade::pade\_sqrt\_denom< Scalar\_T >, 224
  - pade::pade\_sqrt\_number< dd\_real >, 229
  - pade::pade\_sqrt\_number< float >, 230
  - pade::pade\_sqrt\_number< long double >, 231
  - pade::pade\_sqrt\_number< qd\_real >, 232
  - pade::pade\_sqrt\_number< Scalar\_T >, 229
- asin
  - glucat, 26, 27
  - glucat::numeric\_traits< Scalar\_T >, 206
- asinh
  - glucat, 27
- atan
  - glucat, 27, 28
  - glucat::numeric\_traits< Scalar\_T >, 206
- atanh
  - glucat, 28
- basis
  - glucat::basis\_table< Scalar\_T, LO, HI, Matrix\_T >, 87
- basis\_element
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 197
- basis\_matrix\_t
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 190
- basis\_table
  - glucat::basis\_table< Scalar\_T, LO, HI, Matrix\_T >, 86
- BITS\_PER\_SET\_VALUE
  - glucat, 57
- bitset\_t
  - glucat::index\_set< LO, HI >, 164
- BOOST\_STATIC\_ASSERT
  - glucat::index\_set< LO, HI >, 166
- call
  - glucat::control\_t, 125
- cascade\_log
  - glucat, 28
- catch\_exceptions
  - glucat::control\_t, 126
- centre\_pm4\_qp4
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 146
- centre\_pp4\_qm4
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 146
- centre\_qp1\_pm1
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 146
- cga3, 9
  - agc3, 9

- cga3, [9](#)
- cga3std, [10](#)
- cga3std
  - cga3, [10](#)
- check\_complex
  - glucat, [29](#)
- cl
  - PyClical, [81](#)
- classify\_eigenvalues
  - glucat::matrix, [64](#)
- classname
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [112](#)
  - glucat::error< Class\_T >, [133](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [147](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::var\_term, [247](#)
  - glucat::glucat\_error, [158](#)
  - glucat::index\_set< LO, HI >, [166](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [197](#)
- Clifford
  - PyClical.h, [431](#)
- clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >::default\_truncation
  - glucat, [57](#)
- clifford\_algebra.h
  - \_GLUCAT\_CLIFFORD\_ALGEBRA\_OPERATIONS, [257](#)
- clifford\_exp
  - glucat, [29](#)
- clifford\_hidden\_doctests
  - PyClical, [77](#)
- clifford\_to\_repr
  - PyClical.h, [432](#)
- clifford\_to\_str
  - PyClical.h, [432](#)
- compare
  - glucat, [29](#)
  - glucat::index\_set< LO, HI >, [173](#)
- complexifier
  - glucat, [30](#)
- conj
  - glucat, [30](#)
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [112](#)
  - glucat::numeric\_traits< Scalar\_T >, [207](#)
  - PyClical.clifford, [102](#)
- const\_iterator
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [139](#)
- control
  - glucat::control\_t, [126](#)
- control\_t
  - glucat::control\_t, [125](#)
- cos
  - glucat, [30](#)
- glucat::numeric\_traits< Scalar\_T >, [207](#)
- cosh
  - glucat, [31](#)
  - glucat::numeric\_traits< Scalar\_T >, [207](#)
- count
  - glucat::index\_set< LO, HI >, [166](#)
  - PyClical.index\_set, [181](#)
- count\_neg
  - glucat::index\_set< LO, HI >, [166](#)
  - PyClical.index\_set, [181](#)
- count\_pos
  - glucat::index\_set< LO, HI >, [167](#)
  - PyClical.index\_set, [181](#)
- 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 >, [121](#)
- denom
  - pade::pade\_log\_denom< dd\_real >, [217](#)
  - pade::pade\_log\_denom< float >, [218](#)
  - pade::pade\_log\_denom< long double >, [219](#)
  - pade::pade\_log\_denom< qd\_real >, [219](#)
  - pade::pade\_log\_denom< Scalar\_T >, [216](#)
  - pade::pade\_sqrt\_denom< dd\_real >, [225](#)
  - pade::pade\_sqrt\_denom< float >, [226](#)
  - pade::pade\_sqrt\_denom< long double >, [227](#)
  - pade::pade\_sqrt\_denom< qd\_real >, [228](#)
  - pade::pade\_sqrt\_denom< Scalar\_T >, [225](#)
- divide
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [147](#)
- e
  - PyClical, [79](#)
- eig\_case\_t
  - glucat::matrix, [63](#)
- eigenvalues
  - glucat::matrix, [64](#)
- elapsed
  - glucat::timing, [68](#)
- elliptic
  - glucat, [32](#)
- epsilon
  - PyClical.h, [433](#)
- error
  - glucat::error< Class\_T >, [133](#)
- error\_squared
  - glucat, [33](#)
- error\_squared\_tol

- glucat, [33](#)
- error\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [139](#)
  - glucat::index\_set< LO, HI >, [164](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [190](#)
- even
  - glucat, [33](#)
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [112](#)
  - PyClical.clifford, [102](#)
- exp
  - glucat, [34](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [149](#)
  - glucat::numeric\_traits< Scalar\_T >, [207](#)
- EXTRA\_TRIALS
  - glucat::timing, [69](#)
- fast
  - glucat, [34](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [147](#)
- fast\_framed\_multi
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [147](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [197](#)
- fast\_matrix\_multi
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [148](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [197](#)
- fill
  - PyClical, [81](#)
- flip
  - glucat::index\_set< LO, HI >, [167](#)
  - glucat::index\_set< LO, HI >::reference, [239](#)
- fmod
  - glucat::numeric\_traits< Scalar\_T >, [207](#)
- fold
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [148](#)
  - glucat::index\_set< LO, HI >, [167](#)
- folded\_dim
  - glucat, [34](#)
- frame
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [113](#)
  - PyClical.clifford, [102](#)
- framed\_multi
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [142–145](#), [149](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [198](#)
- framed\_multi\_imp.h
  - \_GLUCAT\_HASH\_N, [298](#)
  - \_GLUCAT\_HASH\_SIZE\_T, [298](#)
- framed\_multi\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [139](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [190](#)
- framed\_pair\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [139](#)
- friend\_for\_private\_destructor
  - glucat::basis\_table< Scalar\_T, LO, HI, Matrix\_T >, [87](#)
  - glucat::control\_t, [127](#)
  - glucat::gen::generator\_table< Matrix\_T >, [156](#)
  - glucat::random\_generator< Scalar\_T >, [236](#)
- gen\_from\_pm1\_qm1
  - glucat::gen::generator\_table< Matrix\_T >, [154](#)
- gen\_from\_pm4\_qp4
  - glucat::gen::generator\_table< Matrix\_T >, [154](#)
- gen\_from\_pp4\_qm4
  - glucat::gen::generator\_table< Matrix\_T >, [154](#)
- gen\_from\_qp1\_pm1
  - glucat::gen::generator\_table< Matrix\_T >, [155](#)
- gen\_vector
  - glucat::gen::generator\_table< Matrix\_T >, [155](#)
- generator
  - glucat::gen::generator\_table< Matrix\_T >, [155](#)
  - glucat::random\_generator< Scalar\_T >, [235](#)
- generator\_table
  - glucat::gen::generator\_table< Matrix\_T >, [153](#), [154](#)
- global.h
  - \_GLUCAT\_CTAssert, [327](#)
- 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\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >::default\_truncation, [57](#)
  - 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](#)
- l\_ln2, [58](#)
- l\_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, [23](#)
- 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, [59](#)
- Tuning\_Fast\_Fast\_Size\_Threshold, [59](#)
- 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, [60](#)
- tuning\_naive, [23](#)
- Tuning\_Naive\_Basis\_Max\_Count, [60](#)
- 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, [61](#)
- Tuning\_Slow\_Products\_Size\_Threshold, [61](#)
- vector\_part, [57](#)
- glucat/clifford\_algebra.h, [249](#), [257](#)
- glucat/clifford\_algebra\_imp.h, [266](#), [273](#)
- glucat/errors.h, [286](#), [287](#)
- glucat/errors\_imp.h, [287](#), [288](#)
- glucat/framed\_multi.h, [289](#), [292](#)
- glucat/framed\_multi\_imp.h, [296](#), [299](#)
- glucat/generation.h, [319](#), [320](#)
- glucat/generation\_imp.h, [321](#), [322](#)
- glucat/global.h, [325](#), [327](#)
- glucat/glucat.h, [328](#), [329](#)
- glucat/glucat\_config.h, [330](#), [334](#)
- glucat/glucat\_imp.h, [335](#), [336](#)
- glucat/index\_set.h, [337](#), [338](#)
- glucat/index\_set\_imp.h, [341](#), [342](#)
- glucat/long\_double.h, [354](#), [355](#)
- glucat/matrix.h, [356](#), [358](#)
- glucat/matrix\_imp.h, [359](#), [361](#)
- glucat/matrix\_multi.h, [368](#), [370](#)
- glucat/matrix\_multi\_imp.h, [374](#), [378](#)
- glucat/portability.h, [402](#), [404](#)
- glucat/promotion.h, [404](#), [406](#)
- glucat/qd.h, [408](#), [409](#)
- glucat/random.h, [413](#), [414](#)
- glucat/scalar.h, [415](#), [416](#)
- glucat/scalar\_imp.h, [419](#), [420](#)
- glucat/tuning.h, [422](#), [424](#)
- glucat::basis\_table< Scalar\_T, LO, HI, Matrix\_T >, [85](#)
  - ~basis\_table, [86](#)
  - basis, [87](#)
  - basis\_table, [86](#)

- friend\_for\_private\_destructor, [87](#)
- operator=, [87](#)
- glucat::bool\_to\_type< truth\_value >, [87](#)
- value, [88](#)
- glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [109](#)
- ~clifford\_algebra, [112](#)
- classname, [112](#)
- conj, [112](#)
- default\_truncation, [121](#)
- even, [112](#)
- frame, [113](#)
- grade, [113](#)
- index\_set\_t, [111](#)
- inv, [113](#)
- involute, [113](#)
- isinf, [114](#)
- isnan, [114](#)
- max\_abs, [114](#)
- multivector\_t, [111](#)
- norm, [114](#)
- odd, [115](#)
- operator(), [115](#)
- operator+=, [116](#)
- operator-, [116](#)
- operator=, [116](#)
- operator/=, [117](#)
- operator==, [117](#)
- operator%=: [115](#)
- operator&=: [115](#)
- operator[], [117](#)
- operator\*=, [115](#), [116](#)
- operator^=: [117](#)
- operator|=: [118](#)
- outer\_pow, [118](#)
- pair\_t, [111](#)
- pow, [118](#)
- pure, [118](#)
- quad, [118](#)
- reverse, [119](#)
- scalar, [119](#)
- scalar\_t, [112](#)
- truncated, [119](#)
- v\_hi, [121](#)
- v\_lo, [121](#)
- vector\_part, [119](#), [120](#)
- vector\_t, [112](#)
- write, [120](#)
- glucat::compare\_types< LHS\_T, RHS\_T >, [121](#)
- are\_same, [123](#)
- glucat::compare\_types< T, T >, [123](#)
- are\_same, [123](#)
- glucat::control\_t, [124](#)
- ~control\_t, [125](#)
- call, [125](#)
- catch\_exceptions, [126](#)
- control, [126](#)
- control\_t, [125](#)
- friend\_for\_private\_destructor, [127](#)
- m\_catch\_exceptions, [127](#)
- m\_valid, [127](#)
- m\_verbose\_output, [127](#)
- operator=, [126](#)
- valid, [126](#)
- verbose, [126](#)
- glucat::CTAssertion< bool >, [128](#)
- glucat::CTAssertion< true >, [128](#)
- glucat::error< Class\_T >, [131](#)
- classname, [133](#)
- error, [133](#)
- heading, [133](#)
- print\_error\_msg, [133](#)
- glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [134](#)
- ~framed\_multi, [142](#)
- centre\_pm4\_qp4, [146](#)
- centre\_pp4\_qm4, [146](#)
- centre\_qp1\_pm1, [146](#)
- classname, [147](#)
- const\_iterator, [139](#)
- divide, [147](#)
- error\_t, [139](#)
- exp, [149](#)
- fast, [147](#)
- fast\_framed\_multi, [147](#)
- fast\_matrix\_multi, [148](#)
- fold, [148](#)
- framed\_multi, [142–145](#), [149](#)
- framed\_multi\_t, [139](#)
- framed\_pair\_t, [139](#)
- index\_set\_t, [140](#)
- iterator, [140](#)
- map\_t, [140](#)
- matrix\_multi, [150](#)
- matrix\_multi\_t, [140](#)
- matrix\_t, [140](#)
- multivector\_t, [140](#)
- nbr\_terms, [148](#)
- operator<<, [151](#)
- operator>>, [151](#)
- operator+=, [148](#)
- operator/, [150](#)
- operator%, [150](#)
- operator&, [150](#)
- operator\*, [150](#)
- operator^, [151](#)
- operator|, [151](#)
- random, [149](#)
- scalar\_t, [141](#)
- size\_type, [141](#)
- sorted\_map\_t, [141](#)
- star, [151](#)
- term\_t, [141](#)
- tune\_p, [141](#)
- unfold, [149](#)
- var\_term\_t, [141](#)
- vector\_t, [142](#)

glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P  
     >::hash\_size\_t, 159  
     hash\_size\_t, 160  
     n, 160  
     operator(), 160  
 glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P  
     >::var\_term, 245  
     ~var\_term, 247  
     classname, 247  
     operator\*=: 247  
     var\_pair\_t, 247  
     var\_term, 247  
 glucat::gen, 61  
     offset\_to\_super, 62  
     signature\_t, 61  
 glucat::gen::generator\_table< Matrix\_T >, 152  
     ~generator\_table, 153  
     friend\_for\_private\_destructor, 156  
     gen\_from\_pm1\_qm1, 154  
     gen\_from\_pm4\_qp4, 154  
     gen\_from\_pp4\_qm4, 154  
     gen\_from\_qp1\_pm1, 155  
     gen\_vector, 155  
     generator, 155  
     generator\_table, 153, 154  
     operator(), 155  
     operator=, 156  
 glucat::glucat\_error, 157  
     ~glucat\_error, 158  
     classname, 158  
     glucat\_error, 158  
     heading, 158  
     name, 159  
     print\_error\_msg, 158  
 glucat::index\_set< LO, HI >, 161  
     bitset\_t, 164  
     BOOST\_STATIC\_ASSERT, 166  
     classname, 166  
     compare, 173  
     count, 166  
     count\_neg, 166  
     count\_pos, 167  
     error\_t, 164  
     flip, 167  
     fold, 167  
     hash\_fn, 168  
     index\_pair\_t, 164  
     index\_set, 165, 166  
     index\_set\_t, 164  
     is\_contiguous, 168  
     lex\_less\_than, 168  
     max, 168  
     min, 169  
     operator!=, 169  
     operator<, 169  
     operator==, 170  
     operator&, 173  
     operator&=, 169  
     operator[], 170  
     operator~, 171  
     operator^, 173  
     operator^=, 170  
     operator|, 173  
     operator|=, 170  
     reference, 174  
     reset, 171  
     set, 171, 172  
     sign\_of\_mult, 172  
     sign\_of\_square, 172  
     test, 172  
     unfold, 172  
     v\_hi, 174  
     v\_lo, 174  
     value\_of\_fold, 173  
 glucat::index\_set< LO, HI >::reference, 237  
     ~reference, 238  
     flip, 239  
     index\_set, 240  
     m\_idx, 241  
     m\_pst, 241  
     operator bool, 239  
     operator=, 239  
     operator==, 240  
     operator~, 240  
     reference, 238  
 glucat::index\_set\_hash< LO, HI >, 184  
     index\_set\_t, 184  
     operator(), 185  
 glucat::matrix, 62  
     classify\_eigenvalues, 64  
     eig\_case\_t, 63  
     eigenvalues, 64  
     inner, 64  
     isinf, 64  
     isnan, 64  
     kron, 65  
     mono\_kron, 65  
     mono\_prod, 65  
     nnz, 65  
     nork, 66  
     nork\_range, 66  
     norm\_frob2, 66  
     prod, 66  
     signed\_perm\_nork, 67  
     sparse\_prod, 67  
     to\_blaze, 67  
     trace, 67  
     unit, 68  
 glucat::matrix::eig\_genus< Matrix\_T >, 129  
     m\_eig\_case, 130  
     m\_is\_singular, 130  
     m\_safe\_arg, 131  
     Scalar\_T, 130  
 glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 185  
     ~matrix\_multi, 192  
     basis\_element, 197

- basis\_matrix\_t, 190
- classname, 197
- error\_t, 190
- fast\_framed\_multi, 197
- fast\_matrix\_multi, 197
- framed\_multi, 198
- framed\_multi\_t, 190
- index\_set\_t, 190
- m\_frame, 201
- m\_matrix, 201
- matrix\_index\_t, 191
- matrix\_log, 198
- matrix\_multi, 192–196, 199
- matrix\_multi\_t, 191
- matrix\_sqrt, 199
- matrix\_t, 191
- multivector\_t, 191
- operator<<, 200
- operator>>, 200
- operator+&, 197
- operator/, 200
- operator=, 198
- operator%, 199
- operator&, 199
- operator\*, 199
- operator^, 200
- operator|, 200
- orientation\_t, 191
- random, 198
- reframe, 200
- scalar\_t, 191
- star, 201
- term\_t, 192
- tune\_p, 192
- vector\_t, 192
- glucat::numeric\_traits< Scalar\_T >, 204
  - abs, 206
  - acos, 206
  - asin, 206
  - atan, 206
  - conj, 207
  - cos, 207
  - cosh, 207
  - exp, 207
  - fmod, 207
  - imag, 208
  - isInf, 208
  - isNaN, 209
  - isNaN\_or\_isInf, 209
  - ln\_2, 210
  - log, 210
  - log2, 210
  - NaN, 211
  - pi, 211
  - pow, 211
  - real, 212
  - sin, 212
  - sinh, 212
  - sqrt, 212
  - tan, 212
  - tanh, 213
  - to\_double, 213
  - to\_int, 213
  - to\_scalar\_t, 213–215
- glucat::numeric\_traits< Scalar\_T >::demoted, 129
  - type, 129
- glucat::numeric\_traits< Scalar\_T >::promoted, 232
  - type, 233
- glucat::random\_generator< Scalar\_T >, 233
  - ~random\_generator, 235
  - friend\_for\_private\_destructor, 236
  - generator, 235
  - normal, 235
  - normal\_dist, 236
  - operator=, 235
  - random\_generator, 234
  - seed, 236
  - uint\_gen, 236
  - uniform, 235
  - uniform\_dist, 236
- glucat::sorted\_range< Map\_T, Sorted\_Map\_T >, 241
  - map\_t, 242
  - sorted\_begin, 243
  - sorted\_end, 243
  - sorted\_iterator, 242
  - sorted\_map\_t, 242
  - sorted\_range, 242
- glucat::sorted\_range< Sorted\_Map\_T, Sorted\_Map\_T >, 243
  - map\_t, 244
  - sorted\_begin, 245
  - sorted\_end, 245
  - sorted\_iterator, 244
  - sorted\_map\_t, 244
  - sorted\_range, 244
- glucat::timing, 68
  - elapsed, 68
  - EXTRA\_TRIALS, 69
  - MS\_PER\_CLOCK, 69
  - MS\_PER\_SEC, 69
- glucat\_config.h
  - GLUCAT\_HAVE\_CXX11, 331
  - GLUCAT\_HAVE\_INTTYPES\_H, 331
  - GLUCAT\_HAVE\_STDINT\_H, 331
  - GLUCAT\_HAVE\_STDIO\_H, 331
  - GLUCAT\_HAVE\_STDLIB\_H, 331
  - GLUCAT\_HAVE\_STRING\_H, 332
  - GLUCAT\_HAVE\_STRINGS\_H, 332
  - GLUCAT\_HAVE\_SYS\_STAT\_H, 332
  - GLUCAT\_HAVE\_SYS\_TYPES\_H, 332
  - GLUCAT\_HAVE\_UNISTD\_H, 332
  - GLUCAT\_PACKAGE, 332
  - GLUCAT\_PACKAGE\_BUGREPORT, 332
  - GLUCAT\_PACKAGE\_NAME, 332
  - GLUCAT\_PACKAGE\_STRING, 333
  - GLUCAT\_PACKAGE\_TARNAME, 333



- GLUCAT\_PACKAGE\_URL, 333
- GLUCAT\_PACKAGE\_VERSION, 333
- GLUCAT\_STDC\_HEADERS, 333
- GLUCAT\_VERSION, 333
- glucat\_error
  - glucat::glucat\_error, 158
- GLUCAT\_HAVE\_CXX11
  - glucat\_config.h, 331
- GLUCAT\_HAVE\_INTTYPES\_H
  - glucat\_config.h, 331
- GLUCAT\_HAVE\_STDINT\_H
  - glucat\_config.h, 331
- GLUCAT\_HAVE\_STDIO\_H
  - glucat\_config.h, 331
- GLUCAT\_HAVE\_STDLIB\_H
  - glucat\_config.h, 331
- GLUCAT\_HAVE\_STRING\_H
  - glucat\_config.h, 332
- GLUCAT\_HAVE\_STRINGS\_H
  - glucat\_config.h, 332
- GLUCAT\_HAVE\_SYS\_STAT\_H
  - glucat\_config.h, 332
- GLUCAT\_HAVE\_SYS\_TYPES\_H
  - glucat\_config.h, 332
- GLUCAT\_HAVE\_UNISTD\_H
  - glucat\_config.h, 332
- GLUCAT\_PACKAGE
  - glucat\_config.h, 332
- GLUCAT\_PACKAGE\_BUGREPORT
  - glucat\_config.h, 332
- GLUCAT\_PACKAGE\_NAME
  - glucat\_config.h, 332
- GLUCAT\_PACKAGE\_STRING
  - glucat\_config.h, 333
- GLUCAT\_PACKAGE\_TARNAME
  - glucat\_config.h, 333
- GLUCAT\_PACKAGE\_URL
  - glucat\_config.h, 333
- GLUCAT\_PACKAGE\_VERSION
  - glucat\_config.h, 333
- glucat\_package\_version
  - PyClical.h, 433
- GLUCAT\_STDC\_HEADERS
  - glucat\_config.h, 333
- GLUCAT\_VERSION
  - glucat\_config.h, 333
- grade
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 113
- hash\_fn
  - glucat::index\_set< LO, HI >, 168
  - PyClical.index\_set, 182
- hash\_size\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::hash\_size\_t, 160
- heading
  - glucat::error< Class\_T >, 133
  - glucat::glucat\_error, 158
- hi\_ndx
  - PyClical.h, 433
- i
  - PyClical, 81
- imag
  - glucat, 35
  - glucat::numeric\_traits< Scalar\_T >, 208
- index\_pair\_t
  - glucat::index\_set< LO, HI >, 164
- index\_set
  - glucat::index\_set< LO, HI >, 165, 166
  - glucat::index\_set< LO, HI >::reference, 240
- index\_set\_hidden\_doctests
  - PyClical, 79
- index\_set\_t
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 111
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 140
  - glucat::index\_set< LO, HI >, 164
  - glucat::index\_set\_hash< LO, HI >, 184
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 190
- index\_set\_to\_repr
  - PyClical.h, 432
- index\_set\_to\_str
  - PyClical.h, 432
- index\_t
  - glucat, 22
- IndexSet
  - PyClical.h, 431
- inner
  - glucat::matrix, 64
- instance
  - PyClical.clifford, 108
  - PyClical.index\_set, 184
- intfn
  - glucat, 22
- intintfn
  - glucat, 22
- inv
  - glucat, 35
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 113
  - PyClical.clifford, 103
- inverse\_gray
  - glucat, 35
- inverse\_reversed\_gray
  - glucat, 35
- involute
  - glucat, 36
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 113
  - PyClical.clifford, 103
- is\_contiguous
  - glucat::index\_set< LO, HI >, 168
- isInf
  - glucat::numeric\_traits< Scalar\_T >, 208

- isinf
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 114
  - glucat::matrix, 64
  - PyClical.clifford, 103
- isNaN
  - glucat::numeric\_traits< Scalar\_T >, 209
- isnan
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 114
  - glucat::matrix, 64
  - PyClical.clifford, 104
- isNaN\_or\_isInf
  - glucat::numeric\_traits< Scalar\_T >, 209
- ist
  - PyClical, 81
- istpq
  - PyClical, 80
- iterator
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 140
- ixt
  - PyClical, 81
- kron
  - glucat::matrix, 65
- l\_in2
  - glucat, 58
- l\_pi
  - glucat, 58
- lex\_less\_than
  - glucat::index\_set< LO, HI >, 168
- lhs
  - PyClical, 81
- ln\_2
  - glucat::numeric\_traits< Scalar\_T >, 210
- lo\_ndx
  - PyClical.h, 433
- log
  - glucat, 36, 37
  - glucat::numeric\_traits< Scalar\_T >, 210
- log2
  - glucat, 37
  - glucat::numeric\_traits< Scalar\_T >, 210
- m\_catch\_exceptions
  - glucat::control\_t, 127
- m\_eig\_case
  - glucat::matrix::eig\_genus< Matrix\_T >, 130
- m\_frame
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 201
- m\_idx
  - glucat::index\_set< LO, HI >::reference, 241
- m\_is\_singular
  - glucat::matrix::eig\_genus< Matrix\_T >, 130
- m\_matrix
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 201
- m\_pst
  - glucat::index\_set< LO, HI >::reference, 241
- m\_safe\_arg
  - glucat::matrix::eig\_genus< Matrix\_T >, 131
- m\_valid
  - glucat::control\_t, 127
- m\_verbose\_output
  - glucat::control\_t, 127
- map\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 140
  - glucat::sorted\_range< Map\_T, Sorted\_Map\_T >, 242
  - glucat::sorted\_range< Sorted\_Map\_T, Sorted\_Map\_T >, 244
- matrix\_index\_t
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 191
- matrix\_log
  - glucat, 37
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 198
- matrix\_multi
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 150
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 192–196, 199
- matrix\_multi\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 140
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 191
- matrix\_sqrt
  - glucat, 37
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 199
- matrix\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 140
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 191
- max
  - glucat::index\_set< LO, HI >, 168
  - PyClical.index\_set, 182
- max\_abs
  - glucat, 38
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 114
  - PyClical.clifford, 104
- max\_pos
  - glucat, 38
- min
  - glucat::index\_set< LO, HI >, 169
  - PyClical.index\_set, 182
- min\_neg
  - glucat, 38

- mono\_kron
  - glucat::matrix, [65](#)
- mono\_prod
  - glucat::matrix, [65](#)
- MS\_PER\_CLOCK
  - glucat::timing, [69](#)
- MS\_PER\_S
  - glucat, [58](#)
- MS\_PER\_SEC
  - glucat::timing, [69](#)
- multivector\_t
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [111](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [140](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [191](#)
- n
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::hash\_size\_t, [160](#)
- name
  - glucat::glucat\_error, [159](#)
- NaN
  - glucat::numeric\_traits< Scalar\_T >, [211](#)
- nbar3
  - PyClical, [81](#)
- nbr\_terms
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [148](#)
- ninf3
  - PyClical, [82](#)
- nnz
  - glucat::matrix, [65](#)
- None
  - PyClical, [82](#)
- nork
  - glucat::matrix, [66](#)
- nork\_range
  - glucat::matrix, [66](#)
- norm
  - glucat, [38](#)
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [114](#)
  - PyClical.clifford, [104](#)
- norm\_frob2
  - glucat::matrix, [66](#)
- normal
  - glucat::random\_generator< Scalar\_T >, [235](#)
- normal\_dist
  - glucat::random\_generator< Scalar\_T >, [236](#)
- numer
  - pade::pade\_log\_numer< dd\_real >, [221](#)
  - pade::pade\_log\_numer< float >, [222](#)
  - pade::pade\_log\_numer< long double >, [223](#)
  - pade::pade\_log\_numer< qd\_real >, [224](#)
  - pade::pade\_log\_numer< Scalar\_T >, [220](#)
  - pade::pade\_sqrt\_numer< dd\_real >, [230](#)
  - pade::pade\_sqrt\_numer< float >, [230](#)
  - pade::pade\_sqrt\_numer< long double >, [231](#)
  - pade::pade\_sqrt\_numer< qd\_real >, [232](#)
  - pade::pade\_sqrt\_numer< Scalar\_T >, [229](#)
- obj
  - PyClical, [82](#)
- odd
  - glucat, [39](#)
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [115](#)
  - PyClical.clifford, [105](#)
- offset\_level
  - glucat, [39](#)
- offset\_to\_super
  - glucat::gen, [62](#)
- operator bool
  - glucat::index\_set< LO, HI >::reference, [239](#)
- operator!=
  - glucat, [39](#), [40](#)
  - glucat::index\_set< LO, HI >, [169](#)
- operator<
  - glucat::index\_set< LO, HI >, [169](#)
- operator<<
  - glucat, [46](#), [47](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [151](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [200](#)
- operator>>
  - glucat, [47](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [151](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [200](#)
- operator()
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [115](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::hash\_size\_t, [160](#)
  - glucat::gen::generator\_table< Matrix\_T >, [155](#)
  - glucat::index\_set\_hash< LO, HI >, [185](#)
- operator+
  - glucat, [43](#), [44](#)
- operator+=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [116](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [148](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [197](#)
- operator-
  - glucat, [44](#)
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [116](#)
- operator-=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [116](#)
- operator/
  - glucat, [45](#), [46](#)

- glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 150
- glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 200
- operator/=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 117
- operator=
  - glucat::basis\_table< Scalar\_T, LO, HI, Matrix\_T >, 87
  - glucat::control\_t, 126
  - glucat::gen::generator\_table< Matrix\_T >, 156
  - glucat::index\_set< LO, HI >::reference, 239
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 198
  - glucat::random\_generator< Scalar\_T >, 235
- operator==
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 117
  - glucat::index\_set< LO, HI >, 170
  - glucat::index\_set< LO, HI >::reference, 240
- operator%
  - glucat, 40
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 150
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 199
- operator%=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 115
- operator&
  - glucat, 41
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 150
  - glucat::index\_set< LO, HI >, 173
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 199
- operator&=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 115
  - glucat::index\_set< LO, HI >, 169
- operator[]
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 117
  - glucat::index\_set< LO, HI >, 170
- operator\*
  - glucat, 42, 43
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 150
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 199
- operator\*=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 115, 116
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::var\_term, 247
- operator~
  - glucat::index\_set< LO, HI >, 171
- glucat::index\_set< LO, HI >::reference, 240
- operator^
  - glucat, 47, 48
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 151
  - glucat::index\_set< LO, HI >, 173
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 200
- operator^=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 117
  - glucat::index\_set< LO, HI >, 170
- operator|
  - glucat, 48, 49
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 151
  - glucat::index\_set< LO, HI >, 173
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 200
- operator|=
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 118
  - glucat::index\_set< LO, HI >, 170
- orientation\_t
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 191
- outer\_pow
  - glucat, 49
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 118
  - PyClical.clifford, 105
- pade, 69
  - pade\_log\_denom< dd\_real >::denom, 70
  - pade\_log\_denom< float >::denom, 70
  - pade\_log\_denom< longdouble >::denom, 70
  - pade\_log\_denom< qd\_real >::denom, 71
  - pade\_log\_denom< Scalar\_T >::denom, 71
  - pade\_log\_numer< dd\_real >::numer, 72
  - pade\_log\_numer< float >::numer, 72
  - pade\_log\_numer< longdouble >::numer, 72
  - pade\_log\_numer< qd\_real >::numer, 72
  - pade\_log\_numer< Scalar\_T >::numer, 73
  - pade\_sqrt\_denom< dd\_real >::denom, 73
  - pade\_sqrt\_denom< float >::denom, 74
  - pade\_sqrt\_denom< longdouble >::denom, 74
  - pade\_sqrt\_denom< qd\_real >::denom, 74
  - pade\_sqrt\_denom< Scalar\_T >::denom, 75
  - pade\_sqrt\_numer< dd\_real >::numer, 75
  - pade\_sqrt\_numer< float >::numer, 75
  - pade\_sqrt\_numer< longdouble >::numer, 75
  - pade\_sqrt\_numer< qd\_real >::numer, 76
  - pade\_sqrt\_numer< Scalar\_T >::numer, 76
- pade::pade\_log\_denom< dd\_real >, 216
  - array, 217
  - denom, 217
- pade::pade\_log\_denom< float >, 217
  - array, 218
  - denom, 218

- pade::pade\_log\_denom< long double >, 218
  - array, 218
  - denom, 219
- pade::pade\_log\_denom< qd\_real >, 219
  - array, 219
  - denom, 219
- pade::pade\_log\_denom< Scalar\_T >, 215
  - array, 216
  - denom, 216
- pade::pade\_log\_numer< dd\_real >, 221
  - array, 221
  - numer, 221
- pade::pade\_log\_numer< float >, 221
  - array, 222
  - numer, 222
- pade::pade\_log\_numer< long double >, 222
  - array, 223
  - numer, 223
- pade::pade\_log\_numer< qd\_real >, 223
  - array, 223
  - numer, 224
- pade::pade\_log\_numer< Scalar\_T >, 220
  - array, 220
  - numer, 220
- pade::pade\_sqrt\_denom< dd\_real >, 225
  - array, 225
  - denom, 225
- pade::pade\_sqrt\_denom< float >, 226
  - array, 226
  - denom, 226
- pade::pade\_sqrt\_denom< long double >, 226
  - array, 227
  - denom, 227
- pade::pade\_sqrt\_denom< qd\_real >, 227
  - array, 228
  - denom, 228
- pade::pade\_sqrt\_denom< Scalar\_T >, 224
  - array, 224
  - denom, 225
- pade::pade\_sqrt\_numer< dd\_real >, 229
  - array, 229
  - numer, 230
- pade::pade\_sqrt\_numer< float >, 230
  - array, 230
  - numer, 230
- pade::pade\_sqrt\_numer< long double >, 231
  - array, 231
  - numer, 231
- pade::pade\_sqrt\_numer< qd\_real >, 231
  - array, 232
  - numer, 232
- pade::pade\_sqrt\_numer< Scalar\_T >, 228
  - array, 229
  - numer, 229
- pade\_approx
  - glucat, 50
- pade\_log
  - glucat, 50
- pade\_log\_denom< dd\_real >::denom
  - pade, 70
- pade\_log\_denom< float >::denom
  - pade, 70
- pade\_log\_denom< longdouble >::denom
  - pade, 70
- pade\_log\_denom< qd\_real >::denom
  - pade, 71
- pade\_log\_denom< Scalar\_T >::denom
  - pade, 71
- pade\_log\_numer< dd\_real >::numer
  - pade, 72
- pade\_log\_numer< float >::numer
  - pade, 72
- pade\_log\_numer< longdouble >::numer
  - pade, 72
- pade\_log\_numer< qd\_real >::numer
  - pade, 72
- pade\_log\_numer< Scalar\_T >::numer
  - pade, 73
- pade\_sqrt\_denom< dd\_real >::denom
  - pade, 73
- pade\_sqrt\_denom< float >::denom
  - pade, 74
- pade\_sqrt\_denom< longdouble >::denom
  - pade, 74
- pade\_sqrt\_denom< qd\_real >::denom
  - pade, 74
- pade\_sqrt\_denom< Scalar\_T >::denom
  - pade, 75
- pade\_sqrt\_numer< dd\_real >::numer
  - pade, 75
- pade\_sqrt\_numer< float >::numer
  - pade, 75
- pade\_sqrt\_numer< longdouble >::numer
  - pade, 75
- pade\_sqrt\_numer< qd\_real >::numer
  - pade, 76
- pade\_sqrt\_numer< Scalar\_T >::numer
  - pade, 76
- pair\_t
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 111
- pi
  - glucat::numeric\_traits< Scalar\_T >, 211
  - PyClical, 82
- portability.h
  - \_GLUCAT\_ISINF, 403
  - \_GLUCAT\_ISNAN, 403
  - UBLAS\_ABS, 403
  - UBLAS\_SQRT, 403
- pos\_mod
  - glucat, 50
- pow
  - glucat, 50, 51
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 118
  - glucat::numeric\_traits< Scalar\_T >, 211

- PyClical.clifford, 105
- print\_error\_msg
  - glucat::error< Class\_T >, 133
  - glucat::glucat\_error, 158
- prod
  - glucat::matrix, 66
- pure
  - glucat, 51
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 118
  - PyClical.clifford, 106
- PyClical, 77
  - \_\_version\_\_, 81
  - \_test, 77
  - cl, 81
  - clifford\_hidden\_doctests, 77
  - e, 79
  - fill, 81
  - i, 81
  - index\_set\_hidden\_doctests, 79
  - ist, 81
  - istpq, 80
  - iht, 81
  - lhs, 81
  - nbar3, 81
  - ninf3, 82
  - None, 82
  - obj, 82
  - pi, 82
  - rhs, 82
  - scalar\_epsilon, 82
  - tau, 82
  - threshold, 82
  - tol, 83
- PyClical.clifford, 88
  - \_\_add\_\_, 90
  - \_\_and\_\_, 90
  - \_\_call\_\_, 91
  - \_\_cinit\_\_, 91
  - \_\_contains\_\_, 92
  - \_\_dealloc\_\_, 92
  - \_\_getitem\_\_, 92
  - \_\_iadd\_\_, 93
  - \_\_iand\_\_, 93
  - \_\_idiv\_\_, 93
  - \_\_imod\_\_, 94
  - \_\_imul\_\_, 94
  - \_\_ior\_\_, 94
  - \_\_isub\_\_, 95
  - \_\_iter\_\_, 95
  - \_\_ixor\_\_, 95
  - \_\_mod\_\_, 95
  - \_\_mul\_\_, 96
  - \_\_neg\_\_, 96
  - \_\_or\_\_, 96
  - \_\_pos\_\_, 97
  - \_\_pow\_\_, 97
  - \_\_radd\_\_, 97
  - \_\_rand\_\_, 98
  - \_\_repr\_\_, 98
  - \_\_richcmp\_\_, 98
  - \_\_rmod\_\_, 99
  - \_\_rmul\_\_, 99
  - \_\_rsub\_\_, 99
  - \_\_rtruediv\_\_, 100
  - \_\_rxor\_\_, 100
  - \_\_str\_\_, 100
  - \_\_sub\_\_, 100
  - \_\_truediv\_\_, 101
  - \_\_xor\_\_, 101
  - abs, 101
  - conj, 102
  - even, 102
  - frame, 102
  - instance, 108
  - inv, 103
  - involute, 103
  - isinf, 103
  - isnan, 104
  - max\_abs, 104
  - norm, 104
  - odd, 105
  - outer\_pow, 105
  - pow, 105
  - pure, 106
  - quad, 106
  - reframe, 106
  - reverse, 107
  - scalar, 107
  - truncated, 107
  - vector\_part, 108
- PyClical.h
  - Clifford, 431
  - clifford\_to\_repr, 432
  - clifford\_to\_str, 432
  - epsilon, 433
  - glucat\_package\_version, 433
  - hi\_ndx, 433
  - index\_set\_to\_repr, 432
  - index\_set\_to\_str, 432
  - IndexSet, 431
  - lo\_ndx, 433
  - PyFloat\_FromDouble, 433
  - scalar\_t, 431
  - String, 432
- PyClical.index\_set, 175
  - \_\_and\_\_, 176
  - \_\_cinit\_\_, 176
  - \_\_contains\_\_, 176
  - \_\_dealloc\_\_, 177
  - \_\_getitem\_\_, 177
  - \_\_iand\_\_, 177
  - \_\_invert\_\_, 178
  - \_\_ior\_\_, 178
  - \_\_iter\_\_, 178
  - \_\_ixor\_\_, 178

- `__or__`, 179
- `__repr__`, 179
- `__richcmp__`, 179
- `__setitem__`, 180
- `__str__`, 180
- `__xor__`, 180
- `count`, 181
- `count_neg`, 181
- `count_pos`, 181
- `hash_fn`, 182
- `instance`, 184
- `max`, 182
- `min`, 182
- `sign_of_mult`, 183
- `sign_of_square`, 183
- `pyclical/glucat.pxd`, 428
- `pyclical/PyClical.h`, 430, 434
- `pyclical/PyClical.pxd`, 436
- `pyclical/PyClical.pyx`, 436, 437
- `PyFloat_FromDouble`
  - `PyClical.h`, 433
- `quad`
  - `glucat`, 51
  - `glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >`, 118
  - `PyClical.clifford`, 106
- `random`
  - `glucat::framed_multi< Scalar_T, LO, HI, Tune_P >`, 149
  - `glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >`, 198
- `random_generator`
  - `glucat::random_generator< Scalar_T >`, 234
- `real`
  - `glucat`, 51
  - `glucat::numeric_traits< Scalar_T >`, 212
- `reference`
  - `glucat::index_set< LO, HI >`, 174
  - `glucat::index_set< LO, HI >::reference`, 238
- `reframe`
  - `glucat`, 52
  - `glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >`, 200
  - `PyClical.clifford`, 106
- `reset`
  - `glucat::index_set< LO, HI >`, 171
- `reverse`
  - `glucat`, 52
  - `glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >`, 119
  - `PyClical.clifford`, 107
- `rhs`
  - `PyClical`, 82
- `scalar`
  - `glucat`, 52
- `glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >`, 119
- `PyClical.clifford`, 107
- `scalar_epsilon`
  - `PyClical`, 82
- `Scalar_T`
  - `glucat::matrix::eig_genus< Matrix_T >`, 130
- `scalar_t`
  - `glucat::clifford_algebra< Scalar_T, Index_Set_T, Multivector_T >`, 112
  - `glucat::framed_multi< Scalar_T, LO, HI, Tune_P >`, 141
  - `glucat::matrix_multi< Scalar_T, LO, HI, Tune_P >`, 191
  - `PyClical.h`, 431
- `seed`
  - `glucat::random_generator< Scalar_T >`, 236
- `set`
  - `glucat::index_set< LO, HI >`, 171, 172
- `set_value_t`
  - `glucat`, 23
- `sign_of_mult`
  - `glucat::index_set< LO, HI >`, 172
  - `PyClical.index_set`, 183
- `sign_of_square`
  - `glucat`, 52
  - `glucat::index_set< LO, HI >`, 172
  - `PyClical.index_set`, 183
- `signature_t`
  - `glucat::gen`, 61
- `signed_perm_nork`
  - `glucat::matrix`, 67
- `sin`
  - `glucat`, 53
  - `glucat::numeric_traits< Scalar_T >`, 212
- `sinh`
  - `glucat`, 53
  - `glucat::numeric_traits< Scalar_T >`, 212
- `size_type`
  - `glucat::framed_multi< Scalar_T, LO, HI, Tune_P >`, 141
- `sorted_begin`
  - `glucat::sorted_range< Map_T, Sorted_Map_T >`, 243
  - `glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >`, 245
- `sorted_end`
  - `glucat::sorted_range< Map_T, Sorted_Map_T >`, 243
  - `glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >`, 245
- `sorted_iterator`
  - `glucat::sorted_range< Map_T, Sorted_Map_T >`, 242
  - `glucat::sorted_range< Sorted_Map_T, Sorted_Map_T >`, 244
- `sorted_map_t`
  - `glucat::framed_multi< Scalar_T, LO, HI, Tune_P >`,

- 141
- glucat::sorted\_range< Map\_T, Sorted\_Map\_T >, 242
- glucat::sorted\_range< Sorted\_Map\_T, Sorted\_Map\_T >, 244
- sorted\_range
  - glucat::sorted\_range< Map\_T, Sorted\_Map\_T >, 242
  - glucat::sorted\_range< Sorted\_Map\_T, Sorted\_Map\_T >, 244
- sparse\_prod
  - glucat::matrix, 67
- sqrt
  - glucat, 53, 54
  - glucat::numeric\_traits< Scalar\_T >, 212
- star
  - glucat, 54, 55
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 151
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 201
- std, 83
- std::numeric\_limits< glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P > >, 202
- std::numeric\_limits< glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P > >, 203
- String
  - PyClical.h, 432
- tan
  - glucat, 55
  - glucat::numeric\_traits< Scalar\_T >, 212
- tanh
  - glucat, 56
  - glucat::numeric\_traits< Scalar\_T >, 213
- tau
  - PyClical, 82
- term\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 141
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 192
- test
  - glucat::index\_set< LO, HI >, 172
- test/control.h, 461, 462
- test/driver.h, 463, 464
- test/timing.h, 464, 465
- test/try\_catch.h, 466
- test/tuning.h, 426, 427
- test/undefine.h, 467
- threshold
  - PyClical, 82
- to\_blaze
  - glucat::matrix, 67
- to\_demote
  - glucat, 56
- to\_double
  - glucat::numeric\_traits< Scalar\_T >, 213
- to\_int
  - glucat::numeric\_traits< Scalar\_T >, 213
  - to\_promote
    - glucat, 56
  - to\_scalar\_t
    - glucat::numeric\_traits< Scalar\_T >, 213–215
  - tol
    - PyClical, 83
  - trace
    - glucat::matrix, 67
  - truncated
    - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, 119
    - PyClical.clifford, 107
  - try\_catch
    - glucat, 56, 57
  - tune\_p
    - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, 141
    - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, 192
  - tuning.h
    - \_GLUCAT\_CTAssert, 423
  - 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, 59
  - Tuning\_Fast\_Fast\_Size\_Threshold
    - glucat, 59
  - 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, 60
  - tuning\_naive
    - glucat, 23
  - Tuning\_Naive\_Basis\_Max\_Count
    - glucat, 60
  - 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, [61](#)
- Tuning\_Slow\_Products\_Size\_Threshold
  - glucat, [61](#)
- type
  - glucat::numeric\_traits< Scalar\_T >::demoted, [129](#)
  - glucat::numeric\_traits< Scalar\_T >::promoted, [233](#)
- UBLAS\_ABS
  - portability.h, [403](#)
- UBLAS\_SQRT
  - portability.h, [403](#)
- uint\_gen
  - glucat::random\_generator< Scalar\_T >, [236](#)
- unfold
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [149](#)
  - glucat::index\_set< LO, HI >, [172](#)
- uniform
  - glucat::random\_generator< Scalar\_T >, [235](#)
- uniform\_dist
  - glucat::random\_generator< Scalar\_T >, [236](#)
- unit
  - glucat::matrix, [68](#)
- v\_hi
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [121](#)
  - glucat::index\_set< LO, HI >, [174](#)
- v\_lo
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [121](#)
  - glucat::index\_set< LO, HI >, [174](#)
- valid
  - glucat::control\_t, [126](#)
- value
  - glucat::bool\_to\_type< truth\_value >, [88](#)
- value\_of\_fold
  - glucat::index\_set< LO, HI >, [173](#)
- var\_pair\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::var\_term, [247](#)
- var\_term
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >::var\_term, [247](#)
- var\_term\_t
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [141](#)
- vector\_part
  - glucat, [57](#)
- glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [119](#), [120](#)
- PyClical.clifford, [108](#)
- vector\_t
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [112](#)
  - glucat::framed\_multi< Scalar\_T, LO, HI, Tune\_P >, [142](#)
  - glucat::matrix\_multi< Scalar\_T, LO, HI, Tune\_P >, [192](#)
- verbose
  - glucat::control\_t, [126](#)
- write
  - glucat::clifford\_algebra< Scalar\_T, Index\_Set\_T, Multivector\_T >, [120](#)