

quantum++

0.1

Generated by Doxygen 1.8.7

Thu Nov 6 2014 12:44:47



# Contents

<b>1</b>	<b>quantum++ - A C++11 quantum computing library</b>	<b>1</b>
<b>2</b>	<b>Namespace Index</b>	<b>5</b>
2.1	Namespace List . . . . .	5
<b>3</b>	<b>Hierarchical Index</b>	<b>7</b>
3.1	Class Hierarchy . . . . .	7
<b>4</b>	<b>Class Index</b>	<b>9</b>
4.1	Class List . . . . .	9
<b>5</b>	<b>File Index</b>	<b>11</b>
5.1	File List . . . . .	11
<b>6</b>	<b>Namespace Documentation</b>	<b>13</b>
6.1	qpp Namespace Reference . . . . .	13
6.1.1	Typedef Documentation . . . . .	21
6.1.1.1	bra . . . . .	21
6.1.1.2	cmat . . . . .	21
6.1.1.3	cplx . . . . .	21
6.1.1.4	dmat . . . . .	21
6.1.1.5	DynColVect . . . . .	22
6.1.1.6	DynMat . . . . .	22
6.1.1.7	DynRowVect . . . . .	22
6.1.1.8	ket . . . . .	22
6.1.2	Function Documentation . . . . .	22
6.1.2.1	absm . . . . .	22
6.1.2.2	abssq . . . . .	23
6.1.2.3	abssq . . . . .	23
6.1.2.4	adjoint . . . . .	24
6.1.2.5	anticomm . . . . .	24
6.1.2.6	apply . . . . .	25
6.1.2.7	applyCTRL . . . . .	26

6.1.2.8	channel	27
6.1.2.9	channel	28
6.1.2.10	choi	29
6.1.2.11	choi2kraus	30
6.1.2.12	comm	31
6.1.2.13	compperm	32
6.1.2.14	concurrence	33
6.1.2.15	conjugate	34
6.1.2.16	cosm	35
6.1.2.17	cwise	35
6.1.2.18	det	36
6.1.2.19	disp	36
6.1.2.20	disp	37
6.1.2.21	disp	37
6.1.2.22	disp	38
6.1.2.23	disp	38
6.1.2.24	displn	39
6.1.2.25	displn	39
6.1.2.26	displn	40
6.1.2.27	displn	41
6.1.2.28	displn	41
6.1.2.29	entanglement	42
6.1.2.30	evals	43
6.1.2.31	evects	44
6.1.2.32	expm	44
6.1.2.33	funm	45
6.1.2.34	gconcurrence	45
6.1.2.35	grams	46
6.1.2.36	grams	47
6.1.2.37	grams	47
6.1.2.38	hevals	48
6.1.2.39	hevects	48
6.1.2.40	inverse	49
6.1.2.41	invperm	49
6.1.2.42	kron	50
6.1.2.43	kron	50
6.1.2.44	kron	51
6.1.2.45	kron	51
6.1.2.46	kronpow	52
6.1.2.47	load	52

6.1.2.48	<a href="#">loadMATLABmatrix</a>	53
6.1.2.49	<a href="#">loadMATLABmatrix</a>	53
6.1.2.50	<a href="#">loadMATLABmatrix</a>	53
6.1.2.51	<a href="#">logdet</a>	54
6.1.2.52	<a href="#">logm</a>	54
6.1.2.53	<a href="#">lognegativity</a>	55
6.1.2.54	<a href="#">measure</a>	56
6.1.2.55	<a href="#">measure</a>	56
6.1.2.56	<a href="#">measure</a>	57
6.1.2.57	<a href="#">mket</a>	58
6.1.2.58	<a href="#">mket</a>	58
6.1.2.59	<a href="#">mprj</a>	59
6.1.2.60	<a href="#">mprj</a>	59
6.1.2.61	<a href="#">multiidx2n</a>	60
6.1.2.62	<a href="#">n2multiidx</a>	60
6.1.2.63	<a href="#">negativity</a>	61
6.1.2.64	<a href="#">norm</a>	62
6.1.2.65	<a href="#">omega</a>	62
6.1.2.66	<a href="#">operator""_i</a>	63
6.1.2.67	<a href="#">operator""_i</a>	63
6.1.2.68	<a href="#">powm</a>	63
6.1.2.69	<a href="#">prj</a>	64
6.1.2.70	<a href="#">prod</a>	64
6.1.2.71	<a href="#">prod</a>	65
6.1.2.72	<a href="#">ptrace</a>	65
6.1.2.73	<a href="#">ptrace1</a>	66
6.1.2.74	<a href="#">ptrace2</a>	67
6.1.2.75	<a href="#">ptranspose</a>	68
6.1.2.76	<a href="#">qmutualinfo</a>	69
6.1.2.77	<a href="#">rand</a>	70
6.1.2.78	<a href="#">rand</a>	70
6.1.2.79	<a href="#">rand</a>	71
6.1.2.80	<a href="#">rand</a>	71
6.1.2.81	<a href="#">randH</a>	72
6.1.2.82	<a href="#">randint</a>	72
6.1.2.83	<a href="#">randket</a>	73
6.1.2.84	<a href="#">randkraus</a>	73
6.1.2.85	<a href="#">randn</a>	74
6.1.2.86	<a href="#">randn</a>	74
6.1.2.87	<a href="#">randn</a>	75

6.1.2.88	randn	75
6.1.2.89	randperm	76
6.1.2.90	randrho	76
6.1.2.91	randU	77
6.1.2.92	randV	77
6.1.2.93	renyi	77
6.1.2.94	reshape	78
6.1.2.95	save	79
6.1.2.96	saveMATLABmatrix	79
6.1.2.97	saveMATLABmatrix	79
6.1.2.98	saveMATLABmatrix	79
6.1.2.99	schatten	80
6.1.2.100	schmidtcoeff	81
6.1.2.101	schmidtprob	81
6.1.2.102	schmidtU	82
6.1.2.103	schmidtV	83
6.1.2.104	shannon	84
6.1.2.105	sinm	85
6.1.2.106	spectralpowm	85
6.1.2.107	sqrtn	86
6.1.2.108	sum	86
6.1.2.109	sum	87
6.1.2.110	super	87
6.1.2.111	svals	88
6.1.2.112	svdU	88
6.1.2.113	svdV	89
6.1.2.114	syspermute	89
6.1.2.115	trace	90
6.1.2.116	transpose	91
6.1.2.117	tsallis	91
6.1.3	Variable Documentation	92
6.1.3.1	chop	92
6.1.3.2	codes	92
6.1.3.3	ee	92
6.1.3.4	eps	92
6.1.3.5	gt	92
6.1.3.6	infty	93
6.1.3.7	init	93
6.1.3.8	maxn	93
6.1.3.9	pi	93

6.1.3.10	<a href="#">rdevs</a>	93
6.1.3.11	<a href="#">st</a>	93
6.2	<a href="#">qpp::experimental Namespace Reference</a>	93
6.2.1	<a href="#">Detailed Description</a>	94
6.2.2	<a href="#">Function Documentation</a>	94
6.2.2.1	<a href="#">apply</a>	94
6.2.2.2	<a href="#">channel</a>	95
6.2.2.3	<a href="#">choi</a>	96
6.2.2.4	<a href="#">CTRL</a>	97
6.2.2.5	<a href="#">randkraus</a>	98
6.2.2.6	<a href="#">renyi_inf</a>	99
6.2.2.7	<a href="#">super</a>	100
6.3	<a href="#">qpp::internal Namespace Reference</a>	101
6.3.1	<a href="#">Detailed Description</a>	101
6.3.2	<a href="#">Function Documentation</a>	101
6.3.2.1	<a href="#">_check_col_vector</a>	101
6.3.2.2	<a href="#">_check_dims</a>	101
6.3.2.3	<a href="#">_check_dims_match_cvect</a>	102
6.3.2.4	<a href="#">_check_dims_match_mat</a>	102
6.3.2.5	<a href="#">_check_dims_match_rvect</a>	102
6.3.2.6	<a href="#">_check_eq_dims</a>	102
6.3.2.7	<a href="#">_check_nonzero_size</a>	102
6.3.2.8	<a href="#">_check_perm</a>	102
6.3.2.9	<a href="#">_check_row_vector</a>	102
6.3.2.10	<a href="#">_check_square_mat</a>	102
6.3.2.11	<a href="#">_check_subsys_match_dims</a>	102
6.3.2.12	<a href="#">_check_vector</a>	102
6.3.2.13	<a href="#">_kron2</a>	102
6.3.2.14	<a href="#">_multiidx2n</a>	102
6.3.2.15	<a href="#">_n2multiidx</a>	102
6.3.2.16	<a href="#">variadic_vector_emplace</a>	102
6.3.2.17	<a href="#">variadic_vector_emplace</a>	103
<b>7</b>	<b><a href="#">Class Documentation</a></b>	<b>105</b>
7.1	<a href="#">qpp::Codes Class Reference</a>	105
7.1.1	<a href="#">Detailed Description</a>	106
7.1.2	<a href="#">Member Enumeration Documentation</a>	106
7.1.2.1	<a href="#">Type</a>	106
7.1.3	<a href="#">Constructor &amp; Destructor Documentation</a>	106
7.1.3.1	<a href="#">Codes</a>	106

7.1.4	Member Function Documentation	106
7.1.4.1	codeword	106
7.1.5	Friends And Related Function Documentation	107
7.1.5.1	internal::Singleton< const Codes >	107
7.2	qpp::Exception Class Reference	107
7.2.1	Detailed Description	109
7.2.2	Member Enumeration Documentation	109
7.2.2.1	Type	109
7.2.3	Constructor & Destructor Documentation	110
7.2.3.1	Exception	110
7.2.3.2	Exception	110
7.2.4	Member Function Documentation	111
7.2.4.1	_construct_exception_msg	111
7.2.4.2	what	111
7.2.5	Member Data Documentation	111
7.2.5.1	_custom	111
7.2.5.2	_msg	111
7.2.5.3	_type	111
7.2.5.4	_where	111
7.3	qpp::Gates Class Reference	111
7.3.1	Detailed Description	113
7.3.2	Constructor & Destructor Documentation	113
7.3.2.1	Gates	113
7.3.3	Member Function Documentation	114
7.3.3.1	CTRL	114
7.3.3.2	expandout	114
7.3.3.3	Fd	115
7.3.3.4	Id	116
7.3.3.5	Rn	116
7.3.3.6	Xd	116
7.3.3.7	Zd	117
7.3.4	Friends And Related Function Documentation	117
7.3.4.1	internal::Singleton< const Gates >	117
7.3.5	Member Data Documentation	117
7.3.5.1	CNOTab	118
7.3.5.2	CNOTba	118
7.3.5.3	CZ	118
7.3.5.4	FRED	118
7.3.5.5	H	118
7.3.5.6	Id2	118



7.3.5.7	S	118
7.3.5.8	SWAP	118
7.3.5.9	T	118
7.3.5.10	TOF	118
7.3.5.11	X	118
7.3.5.12	Y	118
7.3.5.13	Z	119
7.4	qpp::Init Class Reference	119
7.4.1	Detailed Description	120
7.4.2	Constructor & Destructor Documentation	120
7.4.2.1	Init	120
7.4.2.2	~Init	120
7.4.3	Friends And Related Function Documentation	120
7.4.3.1	internal::Singleton< const Init >	120
7.5	qpp::experimental::Qudit Class Reference	120
7.5.1	Constructor & Destructor Documentation	121
7.5.1.1	Qudit	121
7.5.2	Member Function Documentation	121
7.5.2.1	getD	121
7.5.2.2	getRho	121
7.5.2.3	measure	121
7.5.2.4	measure	122
7.5.3	Member Data Documentation	122
7.5.3.1	_D	122
7.5.3.2	_rho	122
7.6	qpp::RandomDevices Class Reference	122
7.6.1	Detailed Description	123
7.6.2	Constructor & Destructor Documentation	123
7.6.2.1	RandomDevices	123
7.6.3	Friends And Related Function Documentation	124
7.6.3.1	internal::Singleton< RandomDevices >	124
7.6.4	Member Data Documentation	124
7.6.4.1	_rd	124
7.6.4.2	_rng	124
7.7	qpp::internal::Singleton< T > Class Template Reference	124
7.7.1	Detailed Description	124
7.7.2	Constructor & Destructor Documentation	125
7.7.2.1	Singleton	125
7.7.2.2	~Singleton	125
7.7.2.3	Singleton	125

7.7.3	Member Function Documentation	125
7.7.3.1	get_instance	125
7.7.3.2	operator=	125
7.8	qpp::States Class Reference	125
7.8.1	Detailed Description	127
7.8.2	Constructor & Destructor Documentation	127
7.8.2.1	States	127
7.8.3	Friends And Related Function Documentation	127
7.8.3.1	internal::Singleton< const States >	127
7.8.4	Member Data Documentation	127
7.8.4.1	b00	127
7.8.4.2	b01	127
7.8.4.3	b10	128
7.8.4.4	b11	128
7.8.4.5	GHZ	128
7.8.4.6	pb00	128
7.8.4.7	pb01	128
7.8.4.8	pb10	128
7.8.4.9	pb11	128
7.8.4.10	pGHZ	128
7.8.4.11	pW	128
7.8.4.12	px0	128
7.8.4.13	px1	128
7.8.4.14	py0	128
7.8.4.15	py1	129
7.8.4.16	pz0	129
7.8.4.17	pz1	129
7.8.4.18	W	129
7.8.4.19	x0	129
7.8.4.20	x1	129
7.8.4.21	y0	129
7.8.4.22	y1	129
7.8.4.23	z0	129
7.8.4.24	z1	129
7.9	qpp::Timer Class Reference	129
7.9.1	Detailed Description	130
7.9.2	Constructor & Destructor Documentation	130
7.9.2.1	Timer	130
7.9.3	Member Function Documentation	130
7.9.3.1	seconds	130

7.9.3.2	<a href="#">tic</a>	130
7.9.3.3	<a href="#">toc</a>	131
7.9.4	<a href="#">Friends And Related Function Documentation</a>	131
7.9.4.1	<a href="#">operator&lt;&lt;</a>	131
7.9.5	<a href="#">Member Data Documentation</a>	131
7.9.5.1	<a href="#">_end</a>	131
7.9.5.2	<a href="#">_start</a>	131
<b>8</b>	<b><a href="#">File Documentation</a></b>	<b>133</b>
8.1	<a href="#">include/classes/codes.h File Reference</a>	133
8.2	<a href="#">include/classes/exception.h File Reference</a>	134
8.3	<a href="#">include/classes/gates.h File Reference</a>	134
8.4	<a href="#">include/classes/init.h File Reference</a>	135
8.5	<a href="#">include/classes/randevs.h File Reference</a>	135
8.6	<a href="#">include/classes/singleton.h File Reference</a>	136
8.7	<a href="#">include/classes/states.h File Reference</a>	137
8.8	<a href="#">include/classes/timer.h File Reference</a>	137
8.9	<a href="#">include/constants.h File Reference</a>	138
8.10	<a href="#">include/entanglement.h File Reference</a>	139
8.11	<a href="#">include/entropies.h File Reference</a>	140
8.12	<a href="#">include/experimental/classes/qudit.h File Reference</a>	141
8.13	<a href="#">include/experimental/test.h File Reference</a>	141
8.14	<a href="#">include/functions.h File Reference</a>	142
8.15	<a href="#">include/internal/functions.h File Reference</a>	146
8.16	<a href="#">include/instruments.h File Reference</a>	147
8.17	<a href="#">include/io.h File Reference</a>	148
8.18	<a href="#">include/MATLAB/matlab.h File Reference</a>	149
8.19	<a href="#">include/operations.h File Reference</a>	150
8.20	<a href="#">include/qpp.h File Reference</a>	152
8.21	<a href="#">include/random.h File Reference</a>	153
8.22	<a href="#">include/types.h File Reference</a>	154
8.23	<a href="#">mainpage.dox File Reference</a>	155
	<b><a href="#">Index</a></b>	<b>156</b>



# Chapter 1

## quantum++ - A C++11 quantum computing library

### Version

0.1

### Author

Vlad Gheorghiu ([vgheorgh@gmail.com](mailto:vgheorgh@gmail.com))

### Copyright

(c) 2013 - 2014 Vlad Gheorghiu ([vgheorgh@gmail.com](mailto:vgheorgh@gmail.com))

An example is worth more than one thousand words :)

```
#include "qpp.h"

// #include "MATLAB/matlab.h" // support for MATLAB

using namespace qpp;

int main()
{
    // Qudit Teleportation
    {
        std::size_t D = 3; // size of the system
        std::cout << std::endl << "**** Qudit Teleportation, D = " << D
            << " ****" << std::endl;
        ket mes_AB = ket::Zero(D * D); // maximally entangled state resource
        for (std::size_t i = 0; i < D; i++)
            mes_AB += mket( { i, i }, D);
        mes_AB /= std::sqrt((double) D);
        cmat Bell_aA = adjoint( // circuit that measures in the qudit Bell basis
            gt.CTRL(gt.Xd(D), { 0 }, { 1 }, 2, D)
            kron(gt.Fd(D), gt.Id(D)));
        ket psi_a = randket(D); // random state as input on a
        std::cout << ">> Initial state:" << std::endl;
        displn(psi_a);
        ket input_aAB = kron(psi_a, mes_AB); // joint input state aAB
        // output before measurement
        ket output_aAB = apply(input_aAB, Bell_aA, { 0, 1 }, 3, D);
        auto measured_aA = measure(ptrace2(prj(output_aAB), { D * D, D })),
            gt.Id(D * D)); // measure on aA
        std::discrete_distribution<std::size_t> dd(measured_aA.first.begin(),
            measured_aA.first.end());
        std::cout << ">> Measurement probabilities: ";
        displn(measured_aA.first, " ");
        std::size_t m = dd(rdevs._rng); // sample
        auto midx = n2multiidx(m, { D, D });
        std::cout << ">> Measurement result: ";
        displn(midx, " ");
        // conditional result on B before correction
        ket output_m_aAB = apply(output_aAB, prj(mket(midx, D)), { 0, 1 }, 3, D)
            / std::sqrt(measured_aA.first[m]);
        cmat correction_B = powm(gt.Zd(D), midx[0])
            powm(adjoint(gt.Xd(D)), midx[1]); // correction operator
```

```

        // apply correction on B
        output_aAB = apply(output_m_aAB, correction_B, { 2 }, 3, D);
        cmat rho_B = ptracel(prj(output_aAB), { D * D, D });
        std::cout << ">> Bob's density operator: " << std::endl;
        displn(rho_B);
        std::cout << ">> Norm difference: " << norm(rho_B - prj(psi_a))
            << std::endl; // verification
    }

// Qudit Dense Coding
{
    std::size_t D = 3; // size of the system
    std::cout << std::endl << "**** Qudit Dense Coding, D = " << D
        << " ****" << std::endl;
    ket mes_AB = ket::Zero(D * D); // maximally entangled state resource
    for (std::size_t i = 0; i < D; i++)
        mes_AB += mket({ i, i }, D);
    mes_AB /= std::sqrt((double) D);
    cmat Bell_AB = adjoint( // circuit that measures in the qudit Bell basis
        gt.CTRL(gt.Xd(D), { 0 }, { 1 }, 2, D)
        kron(gt.Fd(D), gt.Id(D)));
    // equal probabilities of choosing a message
    std::uniform_int_distribution<std::size_t> uid(0, D * D - 1);
    std::size_t m_A = uid(rdevs._rng); // sample, obtain the message index
    auto midx = n2multiidx(m_A, { D, D });
    std::cout << ">> Alice sent: ";
    displn(midx, " ");
    // Alice's operation
    cmat U_A = powm(gt.Zd(D), midx[0]) * powm(adjoint(
        gt.Xd(D)), midx[1]);
    // Alice encodes the message
    ket psi_AB = apply(mes_AB, U_A, { 0 }, 2, D);
    // Bob measures the joint system in the qudit Bell basis
    psi_AB = apply(psi_AB, Bell_AB, { 0, 1 }, 2, D);
    auto measured = measure(psi_AB, gt.Id(D * D));
    std::cout << ">> Bob measurement probabilities: ";
    displn(measured.first, " ");
    // Bob samples according to the measurement probabilities
    std::discrete_distribution<std::size_t> dd(measured.first.begin(),
        measured.first.end());
    std::size_t m_B = dd(rdevs._rng);
    std::cout << ">> Bob received: ";
    displn(n2multiidx(m_B, { D, D }), " ");
}

// Grover's search algorithm, we time it
{
    Timer t; // set a timer
    std::size_t n = 4; // number of qubits
    std::cout << std::endl << "**** Grover on n = " << n << " qubits ****"
        << std::endl;
    std::vector<std::size_t> dims(n, 2); // local dimensions
    std::size_t N = std::pow(2, n); // number of elements in the database
    std::cout << ">> Database size: " << N << std::endl;
    // mark an element randomly
    std::uniform_int_distribution<std::size_t> uid(0, N - 1);
    std::size_t marked = uid(rdevs._rng);
    std::cout << ">> Marked state: " << marked << " -> ";
    displn(n2multiidx(marked, dims), " ");
    ket psi = mket(n2multiidx(0, dims)); // computational |0>^{\otimes n}
    psi = (kronpow(gt.H, n) * psi).eval(); // apply H^{\otimes n}, no aliasing
    cmat G = 2 * prj(psi) - gt.Id(N); // Diffusion operator
    // number of queries
    std::size_t nqueries = std::ceil(pi * std::sqrt(N) / 4.);
    std::cout << ">> We run " << nqueries << " queries" << std::endl;
    for (std::size_t i = 0; i < nqueries; i++)
    {
        psi(marked) = -psi(marked); // apply the oracle first, no aliasing
        psi = (G * psi).eval(); // then the diffusion operator, no aliasing
    }
    // we now measure the state in the computational basis
    auto measured = measure(psi, gt.Id(N));
    std::cout << ">> Probability of the marked state: "
        << measured.first[marked] << std::endl;
    std::cout << ">> Probability of all results: ";
    displn(measured.first, " ");
    std::cout << ">> Let's sample..." << std::endl;
    std::discrete_distribution<std::size_t> dd(measured.first.begin(),
        measured.first.end());
    std::size_t result = dd(rdevs._rng);
    if (result == marked)
        std::cout << ">> Hooray, we obtained the correct result: ";
    else
        std::cout << ">> Not there yet... we obtained: ";
    std::cout << result << " -> ";
    displn(n2multiidx(result, dims), " ");
    // stop the timer and display it
}

```

```

std::cout << ">> It took " << t.toc()
    << " seconds to simulate Grover on " << n << " qubits."
    << std::endl;
}

// Entanglement
{
    std::cout << std::endl << "**** Entanglement ****" << std::endl;
    cmat rho = 0.2 * st.pb00 + 0.8 * st.pb11;
    std::cout << ">> rho: " << std::endl;
    displn(rho);
    std::cout << ">> Concurrence of rho: " << concurrence(rho) << std::endl;
    std::cout << ">> Negativity of rho: " << negativity(rho, { 2, 2 })
        << std::endl;
    std::cout << ">> Logarithmic negativity of rho: "
        << lognegativity(rho, { 2, 2 }) << std::endl;
    ket psi = 0.8 * mket({ 0, 0 }) + 0.6 * mket({ 1, 1 });
    // apply some local random unitaries
    psi = kron(randU(2), randU(2)) * psi;
    std::cout << ">> psi: " << std::endl;
    displn(psi);
    std::cout << ">> Entanglement of psi: " << entanglement(psi, { 2, 2 })
        << std::endl;
    std::cout << ">> Concurrence of psi: " << concurrence(prj(psi))
        << std::endl;
    std::cout << ">> G-Concurrence of psi: " << gconcurrence(psi)
        << std::endl;
    std::cout << ">> Schmidt coefficients of psi: " << std::endl;
    displn(schmidtcoeff(psi, { 2, 2 }));
    std::cout << ">> Schmidt probabilities of psi: " << std::endl;
    displn(schmidtprob(psi, { 2, 2 }));
    cmat U = schmidtU(psi, { 2, 2 });
    cmat V = schmidtV(psi, { 2, 2 });
    std::cout << ">> Schmidt vectors on Alice's side: " << std::endl;
    displn(U);
    std::cout << ">> Schmidt vectors on Bob's side: " << std::endl;
    displn(V);
    std::cout << ">> State psi in the Schmidt basis: " << std::endl;
    displn(adjoint(kron(U, V)) * psi);
    // reconstructed state
    ket psi_from_schmidt = schmidtcoeff(psi, { 2, 2 })(0)
        kron(U.col(0), V.col(0))
        + schmidtcoeff(psi, { 2, 2 })(1) * kron(U.col(1), V.col(1));
    std::cout
        << ">> State psi reconstructed from the Schmidt decomposition: "
        << std::endl;
    displn(psi_from_schmidt);
    std::cout << ">> Norm difference: " << norm(psi - psi_from_schmidt)
        << std::endl;
}

// Quantum error correcting codes
{
    std::cout << std::endl << "**** Quantum error correcting codes ****"
        << std::endl;
    ket a0 = codes.codeword(Codes::Type::FIVE_QUBIT, 0);
    ket a1 = codes.codeword(Codes::Type::FIVE_QUBIT, 1);
    ket b0 = codes.codeword(Codes::Type::SEVEN_QUBIT_STEANE
, 0);
    ket b1 = codes.codeword(Codes::Type::SEVEN_QUBIT_STEANE
, 1);
    ket c0 = codes.codeword(Codes::Type::NINE_QUBIT_SHOR, 0
);
    ket c1 = codes.codeword(Codes::Type::NINE_QUBIT_SHOR, 1
);
    std::cout << ">> Five qubit [[5,1,3]] code. ";
    std::cout << "Checking codeword orthogonality." << std::endl;
    std::cout << ">> <0L | 1L> = ";
    displn(adjoint(a0) * a1);
    std::cout << ">> Seven qubit [[7,1,3]] Steane code. ";
    std::cout << "Checking codeword orthogonality." << std::endl;
    std::cout << ">> <0L | 1L> = ";
    displn(adjoint(b0) * b1);
    std::cout << ">> Nine qubit [[9,1,3]] Shor code. ";
    std::cout << "Checking codeword orthogonality." << std::endl;
    std::cout << ">> <0L | 1L> = ";
    displn(adjoint(c0) * c1);
}
}

```





## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">qpp</a> . . . . .	13
<a href="#">qpp::experimental</a> . . . . .	93
<a href="#">qpp::internal</a> . . . . .	101



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

std::exception	
qpp::Exception . . . . .	107
qpp::experimental::Qudit . . . . .	120
qpp::internal::Singleton< T > . . . . .	124
qpp::internal::Singleton< const Codes > . . . . .	124
qpp::Codes . . . . .	105
qpp::internal::Singleton< const Gates > . . . . .	124
qpp::Gates . . . . .	111
qpp::internal::Singleton< const Init > . . . . .	124
qpp::Init . . . . .	119
qpp::internal::Singleton< const States > . . . . .	124
qpp::States . . . . .	125
qpp::internal::Singleton< RandomDevices > . . . . .	124
qpp::RandomDevices . . . . .	122
qpp::Timer . . . . .	129



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">qpp::Codes</a>	Const Singleton class that defines quantum error correcting codes . . . . .	105
<a href="#">qpp::Exception</a>	Generates custom exceptions, used when validating function parameters . . . . .	107
<a href="#">qpp::Gates</a>	Const Singleton class that implements most commonly used gates . . . . .	111
<a href="#">qpp::Init</a>	Const Singleton class that performs additional initializations/cleanups . . . . .	119
<a href="#">qpp::experimental::Qudit</a>	. . . . .	120
<a href="#">qpp::RandomDevices</a>	Singleton class that manages the source of randomness in the library . . . . .	122
<a href="#">qpp::internal::Singleton&lt; T &gt;</a>	<a href="#">Singleton</a> policy class, used internally to implement the singleton pattern via CRTP (Curiously recurring template pattern) . . . . .	124
<a href="#">qpp::States</a>	Const Singleton class that implements most commonly used states . . . . .	125
<a href="#">qpp::Timer</a>	Measures time . . . . .	129



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

include/	constants.h	138
include/	entanglement.h	139
include/	entropies.h	140
include/	functions.h	142
include/	instruments.h	147
include/	io.h	148
include/	operations.h	150
include/	qpp.h	152
include/	random.h	153
include/	types.h	154
include/classes/	codes.h	133
include/classes/	exception.h	134
include/classes/	gates.h	134
include/classes/	init.h	135
include/classes/	randevs.h	135
include/classes/	singleton.h	136
include/classes/	states.h	137
include/classes/	timer.h	137
include/experimental/	test.h	141
include/experimental/classes/	qudit.h	141
include/internal/	functions.h	146
include/MATLAB/	matlab.h	149





## Chapter 6

# Namespace Documentation

### 6.1 qpp Namespace Reference

#### Namespaces

- [experimental](#)
- [internal](#)

#### Classes

- class [Codes](#)  
*const Singleton class that defines quantum error correcting codes*
- class [Exception](#)  
*Generates custom exceptions, used when validating function parameters.*
- class [Gates](#)  
*const Singleton class that implements most commonly used gates*
- class [Init](#)  
*const Singleton class that performs additional initializations/cleanups*
- class [RandomDevices](#)  
*Singleton class that manages the source of randomness in the library.*
- class [States](#)  
*const Singleton class that implements most commonly used states*
- class [Timer](#)  
*Measures time.*

#### Typedefs

- using [cplx](#) = std::complex< double >  
*Complex number in double precision.*
- template<typename Scalar >  
using [DynMat](#) = Eigen::Matrix< Scalar, Eigen::Dynamic, Eigen::Dynamic >  
*Dynamic Eigen matrix over the field specified by Scalar.*
- template<typename Scalar >  
using [DynColVect](#) = Eigen::Matrix< Scalar, Eigen::Dynamic, 1 >  
*Dynamic Eigen column vector over the field specified by Scalar.*
- template<typename Scalar >  
using [DynRowVect](#) = Eigen::Matrix< Scalar, 1, Eigen::Dynamic >

- *Dynamic Eigen row vector over the field specified by Scalar.*  
using `ket = DynColVect< cplx >`  
*Complex (double precision) dynamic Eigen column vector.*
- using `bra = DynRowVect< cplx >`  
*Complex (double precision) dynamic Eigen row vector.*
- using `cmat = DynMat< cplx >`  
*Complex (double precision) dynamic Eigen matrix.*
- using `dmat = DynMat< double >`  
*Real (double precision) dynamic Eigen matrix.*

## Functions

- `constexpr std::complex< double > operator""_i (unsigned long long int x)`  
*User-defined literal for complex  $i = \sqrt{-1}$  (integer overload)*
- `constexpr std::complex< double > operator""_i (long double x)`  
*User-defined literal for complex  $i = \sqrt{-1}$  (real overload)*
- `std::complex< double > omega (std::size_t D)`  
*D-th root of unity.*
- `template<typename Derived >`  
`DynColVect< cplx > schmidtcoeff (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Schmidt coefficients of the bi-partite pure state A.*
- `template<typename Derived >`  
`cmat schmidtU (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Schmidt basis on Alice's side.*
- `template<typename Derived >`  
`cmat schmidtV (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Schmidt basis on Bob's side.*
- `template<typename Derived >`  
`DynColVect< double > schmidtprob (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Schmidt probabilities of the bi-partite pure state A.*
- `template<typename Derived >`  
`double entanglement (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Entanglement of the bi-partite pure state A.*
- `template<typename Derived >`  
`double gconcurrence (const Eigen::MatrixBase< Derived > &A)`  
*G-concurrence of the bi-partite pure state A.*
- `template<typename Derived >`  
`double negativity (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Negativity of the bi-partite mixed state A.*
- `template<typename Derived >`  
`double lognegativity (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Logarithmic negativity of the bi-partite mixed state A.*
- `template<typename Derived >`  
`double concurrence (const Eigen::MatrixBase< Derived > &A)`  
*Wootters concurrence of the bi-partite qubit mixed state A.*
- `template<typename Derived >`  
`double shannon (const Eigen::MatrixBase< Derived > &A)`  
*Shannon/von-Neumann entropy of the probability distribution/density matrix A.*
- `template<typename Derived >`  
`double renyi (const Eigen::MatrixBase< Derived > &A, double alpha)`

- Renyi-  $\alpha$  entropy of the probability distribution/density matrix A, for  $\alpha \geq 0$ .*
- `template<typename Derived >`  
`double tsallis (const Eigen::MatrixBase< Derived > &A, double alpha)`  
*Tsallis-  $\alpha$  entropy of the probability distribution/density matrix A, for  $\alpha \geq 0$*
  - `template<typename Derived >`  
`double qmutualinfo (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &subsysA, const std::vector< std::size_t > &subsysB, const std::vector< std::size_t > &dims)`  
*Quantum mutual information between 2 subsystems of a composite system.*
  - `template<typename Derived >`  
`DynMat< typename Derived::Scalar > transpose (const Eigen::MatrixBase< Derived > &A)`  
*Transpose.*
  - `template<typename Derived >`  
`DynMat< typename Derived::Scalar > conjugate (const Eigen::MatrixBase< Derived > &A)`  
*Complex conjugate.*
  - `template<typename Derived >`  
`DynMat< typename Derived::Scalar > adjoint (const Eigen::MatrixBase< Derived > &A)`  
*Adjoint.*
  - `template<typename Derived >`  
`DynMat< typename Derived::Scalar > inverse (const Eigen::MatrixBase< Derived > &A)`  
*Inverse.*
  - `template<typename Derived >`  
`Derived::Scalar trace (const Eigen::MatrixBase< Derived > &A)`  
*Trace.*
  - `template<typename Derived >`  
`Derived::Scalar det (const Eigen::MatrixBase< Derived > &A)`  
*Determinant.*
  - `template<typename Derived >`  
`Derived::Scalar logdet (const Eigen::MatrixBase< Derived > &A)`  
*Logarithm of the determinant.*
  - `template<typename Derived >`  
`Derived::Scalar sum (const Eigen::MatrixBase< Derived > &A)`  
*Element-wise sum of A.*
  - `template<typename Derived >`  
`Derived::Scalar prod (const Eigen::MatrixBase< Derived > &A)`  
*Element-wise product of A.*
  - `template<typename Derived >`  
`double norm (const Eigen::MatrixBase< Derived > &A)`  
*Frobenius norm.*
  - `template<typename Derived >`  
`DynColVect< cplx > evals (const Eigen::MatrixBase< Derived > &A)`  
*Eigenvalues.*
  - `template<typename Derived >`  
`cmat evects (const Eigen::MatrixBase< Derived > &A)`  
*Eigenvectors.*
  - `template<typename Derived >`  
`DynColVect< double > hevals (const Eigen::MatrixBase< Derived > &A)`  
*Hermitian eigenvalues.*
  - `template<typename Derived >`  
`cmat hevects (const Eigen::MatrixBase< Derived > &A)`  
*Hermitian eigenvectors.*
  - `template<typename Derived >`  
`DynColVect< double > svals (const Eigen::MatrixBase< Derived > &A)`

*Singular values.*

- `template<typename Derived >`  
`cmat svdU` (const Eigen::MatrixBase< Derived > &A)

*Left singular vectors.*

- `template<typename Derived >`  
`cmat svdV` (const Eigen::MatrixBase< Derived > &A)

*Right singular vectors.*

- `template<typename Derived >`  
`cmat funm` (const Eigen::MatrixBase< Derived > &A, `cplx`(\*f)(const `cplx` &))

*Functional calculus  $f(A)$*

- `template<typename Derived >`  
`cmat sqrtm` (const Eigen::MatrixBase< Derived > &A)

*Matrix square root.*

- `template<typename Derived >`  
`cmat absm` (const Eigen::MatrixBase< Derived > &A)

*Matrix absolut value.*

- `template<typename Derived >`  
`cmat expm` (const Eigen::MatrixBase< Derived > &A)

*Matrix exponential.*

- `template<typename Derived >`  
`cmat logm` (const Eigen::MatrixBase< Derived > &A)

*Matrix logarithm.*

- `template<typename Derived >`  
`cmat sinm` (const Eigen::MatrixBase< Derived > &A)

*Matrix sin.*

- `template<typename Derived >`  
`cmat cosm` (const Eigen::MatrixBase< Derived > &A)

*Matrix cos.*

- `template<typename Derived >`  
`cmat spectralpowm` (const Eigen::MatrixBase< Derived > &A, const `cplx` z)

*Matrix power.*

- `template<typename Derived >`  
`DynMat`< typename Derived::Scalar > `powm` (const Eigen::MatrixBase< Derived > &A, std::size\_t n)

*Matrix power.*

- `template<typename Derived >`  
double `schatten` (const Eigen::MatrixBase< Derived > &A, std::size\_t p)

*Schatten norm.*

- `template<typename OutputScalar , typename Derived >`  
`DynMat`< OutputScalar > `cwise` (const Eigen::MatrixBase< Derived > &A, OutputScalar(\*f)(const typename Derived::Scalar &))

*Functor.*

- `template<typename T >`  
`DynMat`< typename T::Scalar > `kron` (const T &head)

*Kronecker product (variadic overload)*

- `template<typename T , typename... Args>`  
`DynMat`< typename T::Scalar > `kron` (const T &head, const Args &...tail)

*Kronecker product (variadic overload)*

- `template<typename Derived >`  
`DynMat`< typename Derived::Scalar > `kron` (const std::vector< Derived > &As)

*Kronecker product (std::vector overload)*

- `template<typename Derived >`  
`DynMat`< typename Derived::Scalar > `kron` (const std::initializer\_list< Derived > &As)

*Kronecker product (std::initializer\_list overload)*

- template<typename Derived >  
**DynMat**< typename Derived::Scalar > **kronpow** (const Eigen::MatrixBase< Derived > &A, std::size\_t n)  
*Kronecker power.*
- template<typename Derived >  
**DynMat**< typename Derived::Scalar > **reshape** (const Eigen::MatrixBase< Derived > &A, std::size\_t rows, std::size\_t cols)  
*Reshape.*
- template<typename Derived1 , typename Derived2 >  
**DynMat**< typename Derived1::Scalar > **comm** (const Eigen::MatrixBase< Derived1 > &A, const Eigen::MatrixBase< Derived2 > &B)  
*Commutator.*
- template<typename Derived1 , typename Derived2 >  
**DynMat**< typename Derived1::Scalar > **anticomm** (const Eigen::MatrixBase< Derived1 > &A, const Eigen::MatrixBase< Derived2 > &B)  
*Anti-commutator.*
- template<typename Derived >  
**DynMat**< typename Derived::Scalar > **prj** (const Eigen::MatrixBase< Derived > &V)  
*Projector.*
- template<typename Derived >  
**DynMat**< typename Derived::Scalar > **grams** (const std::vector< Derived > &Vs)  
*Gram-Schmidt orthogonalization (std::vector overload)*
- template<typename Derived >  
**DynMat**< typename Derived::Scalar > **grams** (const std::initializer\_list< Derived > &Vs)  
*Gram-Schmidt orthogonalization (std::initializer\_list overload)*
- template<typename Derived >  
**DynMat**< typename Derived::Scalar > **grams** (const Eigen::MatrixBase< Derived > &A)  
*Gram-Schmidt orthogonalization (Eigen expression (matrix) overload)*
- std::vector< std::size\_t > **n2multiidx** (std::size\_t n, const std::vector< std::size\_t > &dims)  
*Non-negative integer index to multi-index.*
- std::size\_t **multiidx2n** (const std::vector< std::size\_t > &midx, const std::vector< std::size\_t > &dims)  
*Multi-index to non-negative integer index.*
- **ket mket** (const std::vector< std::size\_t > &mask, const std::vector< std::size\_t > &dims)  
*Multi-partite qudit ket (different dimensions overload)*
- **ket mket** (const std::vector< std::size\_t > &mask, std::size\_t d=2)  
*Multi-partite qudit ket (same dimensions overload)*
- **cmat mprj** (const std::vector< std::size\_t > &mask, const std::vector< std::size\_t > &dims)  
*Projector onto multi-partite qudit ket (different dimensions overload)*
- **cmat mprj** (const std::vector< std::size\_t > &mask, std::size\_t d=2)  
*Projector onto multi-partite qudit ket (same dimensions overload)*
- std::vector< std::size\_t > **invperm** (const std::vector< std::size\_t > &perm)  
*Inverse permutation.*
- std::vector< std::size\_t > **compperm** (const std::vector< std::size\_t > &perm, const std::vector< std::size\_t > &sigma)  
*Compose permutations.*
- template<typename InputIterator >  
std::vector< double > **abssq** (InputIterator first, InputIterator last)  
*Computes the absolut values squared of a range of complex numbers.*
- template<typename Derived >  
std::vector< double > **abssq** (const Eigen::MatrixBase< Derived > &V)  
*Computes the absolut values squared of a column vector.*
- template<typename InputIterator >  
auto **sum** (InputIterator first, InputIterator last) -> typename InputIterator::value\_type  
*Element-wise sum of a range.*

- `template<typename InputIterator >`  
`auto prod (InputIterator first, InputIterator last) -> typename InputIterator::value_type`  
*Element-wise product of a range.*
- `template<typename Derived >`  
`std::pair< std::vector< double >`  
`, std::vector< cmat > > measure (const Eigen::MatrixBase< Derived > &A, const std::vector< cmat > &Ks)`  
*Measures the state A using the set of Kraus operators Ks.*
- `template<typename Derived >`  
`std::pair< std::vector< double >`  
`, std::vector< cmat > > measure (const Eigen::MatrixBase< Derived > &A, const std::initializer_list< cmat`  
`> &Ks)`  
*Measures the state A using the set of Kraus operators Ks (std::initializer\_list overload)*
- `template<typename Derived >`  
`std::pair< std::vector< double >`  
`, std::vector< cmat > > measure (const Eigen::MatrixBase< Derived > &A, const cmat &M)`  
*Measures the state A in the orthonormal basis specified by the eigenvectors of M.*
- `template<typename InputIterator >`  
`void disp (const InputIterator &first, const InputIterator &last, const std::string &separator, const std::string`  
`&start="["`, const std::string &end=""]", std::ostream &os=std::cout)  
*Displays a range. Does not add a newline.*
- `template<typename InputIterator >`  
`std::ostream & displn (const InputIterator &first, const InputIterator &last, const std::string &separator, const`  
`std::string &start="["`, const std::string &end=""]", std::ostream &os=std::cout)  
*Displays a range. Adds a newline.*
- `template<typename T >`  
`std::ostream & disp (const T &x, const std::string &separator, const std::string &start="["`, const std::string  
`&end=""]", std::ostream &os=std::cout)`  
*Displays a standard container that supports std::begin, std::end and forward iteration. Does not add a newline.*
- `template<typename T >`  
`std::ostream & displn (const T &x, const std::string &separator, const std::string &start="["`, const std::string  
`&end=""]", std::ostream &os=std::cout)`  
*Displays a standard container that supports std::begin, std::end and forward iteration. Adds a newline.*
- `template<typename T >`  
`std::ostream & disp (const T *x, const std::size_t n, const std::string &separator, const std::string &start="["`,  
`const std::string &end=""]", std::ostream &os=std::cout)`  
*Displays a C-style array. Does not add a newline.*
- `template<typename T >`  
`std::ostream & displn (const T *x, const std::size_t n, const std::string &separator, const std::string &start="["`,  
`const std::string &end=""]", std::ostream &os=std::cout)`  
*Displays a C-style array. Adds a newline.*
- `template<typename Derived >`  
`std::ostream & disp (const Eigen::MatrixBase< Derived > &A, double chop=qpp::chop, std::ostream`  
`&os=std::cout)`  
*Displays an Eigen expression in matrix friendly form. Does not add a new line.*
- `template<typename Derived >`  
`std::ostream & displn (const Eigen::MatrixBase< Derived > &A, double chop=qpp::chop, std::ostream`  
`&os=std::cout)`  
*Displays an Eigen expression in matrix friendly form. Adds a newline.*
- `std::ostream & disp (const cplx z, double chop=qpp::chop, std::ostream &os=std::cout)`  
*Displays a number (implicitly converted to std::complex<double>) in friendly form. Does not add a new line.*
- `std::ostream & displn (const cplx z, double chop=qpp::chop, std::ostream &os=std::cout)`  
*Displays a number (implicitly converted to std::complex<double>) in friendly form. Adds a new line.*
- `template<typename Derived >`  
`void save (const Eigen::MatrixBase< Derived > &A, const std::string &fname)`

- Saves Eigen expression to a binary file (internal format) in double precision.*

  - `template<typename Derived >`  
`DynMat< typename Derived::Scalar > load` (const std::string &fname)  
*Loads Eigen matrix from a binary file (internal format) in double precision.*
- `template<typename Derived >`  
`Derived loadMATLABmatrix` (const std::string &mat\_file, const std::string &var\_name)  
*Loads an Eigen dynamic matrix from a MATLAB .mat file, generic version.*
- `template<>`  
`dmat loadMATLABmatrix` (const std::string &mat\_file, const std::string &var\_name)  
*Loads an Eigen dynamic matrix from a MATLAB .mat file, specialization for double matrices ([qpp::dmat](#))*
- `template<>`  
`cmat loadMATLABmatrix` (const std::string &mat\_file, const std::string &var\_name)  
*Loads an Eigen dynamic matrix from a MATLAB .mat file, specialization for complex matrices ([qpp::cmat](#))*
- `template<typename Derived >`  
`void saveMATLABmatrix` (const Eigen::MatrixBase< Derived > &A, const std::string &mat\_file, const std::string &var\_name, const std::string &mode)  
*Saves an Eigen dynamic matrix to a MATLAB .mat file, generic version.*
- `template<>`  
`void saveMATLABmatrix` (const Eigen::MatrixBase< [dmat](#) > &A, const std::string &mat\_file, const std::string &var\_name, const std::string &mode)  
*Saves an Eigen dynamic matrix to a MATLAB .mat file, specialization for double matrices ([qpp::dmat](#))*
- `template<>`  
`void saveMATLABmatrix` (const Eigen::MatrixBase< [cmat](#) > &A, const std::string &mat\_file, const std::string &var\_name, const std::string &mode)  
*Saves an Eigen dynamic matrix to a MATLAB .mat file, specialization for complex matrices ([qpp::cmat](#))*
- `template<typename Derived1 , typename Derived2 >`  
`DynMat< typename Derived1::Scalar > applyCTRL` (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< std::size\_t > &ctrl, const std::vector< std::size\_t > &subsys, std::size\_t n, std::size\_t d=2)  
*Applies the controlled-gate A to the part subsys of a multipartite state vector or density matrix.*
- `template<typename Derived1 , typename Derived2 >`  
`DynMat< typename Derived1::Scalar > apply` (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< std::size\_t > &subsys, std::size\_t n, std::size\_t d=2)  
*Applies the gate A to the part subsys of a multipartite state vector or density matrix.*
- `template<typename Derived >`  
`cmat channel` (const Eigen::MatrixBase< Derived > &rho, const std::vector< [cmat](#) > &Ks)  
*Applies the channel specified by the set of Kraus operators Ks to the density matrix rho.*
- `template<typename Derived >`  
`cmat channel` (const Eigen::MatrixBase< Derived > &rho, const std::vector< [cmat](#) > &Ks, const std::vector< std::size\_t > &subsys, std::size\_t n, std::size\_t d=2)  
*Applies the channel specified by the set of Kraus operators Ks to the part of the density matrix rho specified by subsys.*
- `cmat super` (const std::vector< [cmat](#) > &Ks)  
*Superoperator matrix representation.*
- `cmat choi` (const std::vector< [cmat](#) > &Ks)  
*Choi matrix representation.*
- `std::vector< cmat > choi2kraus` (const [cmat](#) &A)  
*Extracts orthogonal Kraus operators from Choi matrix.*
- `template<typename Derived >`  
`DynMat< typename Derived::Scalar > ptrace1` (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size\_t > &dims)  
*Partial trace.*

- `template<typename Derived >`  
`DynMat< typename Derived::Scalar > ptrace2` (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size\_t > &dims)  
*Partial trace.*
- `template<typename Derived >`  
`DynMat< typename Derived::Scalar > ptrace` (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size\_t > &subsys, const std::vector< std::size\_t > &dims)  
*Partial trace.*
- `template<typename Derived >`  
`DynMat< typename Derived::Scalar > ptranspose` (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size\_t > &subsys, const std::vector< std::size\_t > &dims)  
*Partial transpose.*
- `template<typename Derived >`  
`DynMat< typename Derived::Scalar > syspermute` (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size\_t > &perm, const std::vector< std::size\_t > &dims)  
*System permutation.*
- `template<typename Derived >`  
`Derived rand` (std::size\_t rows, std::size\_t cols, double a=0, double b=1)  
*Generates a random matrix with entries uniformly distributed in the interval [a, b]*
- `template<>`  
`dmat rand` (std::size\_t rows, std::size\_t cols, double a, double b)  
*Generates a random real matrix with entries uniformly distributed in the interval [a, b], specialization for double matrices ([qpp::dmat](#))*
- `template<>`  
`cmat rand` (std::size\_t rows, std::size\_t cols, double a, double b)  
*Generates a random complex matrix with entries (both real and imaginary) uniformly distributed in the interval [a, b], specialization for complex matrices ([qpp::cmat](#))*
- `double rand` (double a=0, double b=1)  
*Generates a random real number uniformly distributed in the interval [a, b]*
- `int randint` (int a=std::numeric\_limits< int >::min(), int b=std::numeric\_limits< int >::max())  
*Generates a random integer (int) uniformly distributed in the interval [a, b].*
- `template<typename Derived >`  
`Derived randn` (std::size\_t rows, std::size\_t cols, double mean=0, double sigma=1)  
*Generates a random matrix with entries normally distributed in  $N(\text{mean}, \text{sigma})$*
- `template<>`  
`dmat randn` (std::size\_t rows, std::size\_t cols, double mean, double sigma)  
*Generates a random real matrix with entries normally distributed in  $N(\text{mean}, \text{sigma})$ , specialization for double matrices ([qpp::dmat](#))*
- `template<>`  
`cmat randn` (std::size\_t rows, std::size\_t cols, double mean, double sigma)  
*Generates a random complex matrix with entries (both real and imaginary) normally distributed in  $N(\text{mean}, \text{sigma})$ , specialization for complex matrices ([qpp::cmat](#))*
- `double randn` (double mean=0, double sigma=1)  
*Generates a random real number (double) normally distributed in  $N(\text{mean}, \text{sigma})$*
- `cmat randU` (std::size\_t D)  
*Generates a random unitary matrix.*
- `cmat randV` (std::size\_t Din, std::size\_t Dout)  
*Generates a random isometry matrix.*
- `std::vector< cmat > randkraus` (std::size\_t N, std::size\_t D)  
*Generates a set of random Kraus operators.*
- `cmat randH` (std::size\_t D)  
*Generates a random Hermitian matrix.*
- `ket randket` (std::size\_t D)



- Generates a random normalized ket (pure state vector)
- `cmat randrho` (`std::size_t D`)  
Generates a random density matrix.
- `std::vector< std::size_t > randperm` (`std::size_t n`)  
Generates a random uniformly distributed permutation.

## Variables

- `constexpr double chop` = 1e-10  
Used in `qpp::disp()` and `qpp::displn()` for setting to zero numbers that have their absolute value smaller than `qpp::ct->::chop`.
- `constexpr double eps` = 1e-12  
Used to decide whether a number or expression in double precision is zero or not.
- `constexpr std::size_t maxn` = 64  
Maximum number of qubits.
- `constexpr double pi` = 3.141592653589793238462643383279502884  
 $\pi$
- `constexpr double ee` = 2.718281828459045235360287471352662497  
Base of natural logarithm,  $e$ .
- `constexpr std::size_t infy` = -1  
Used to denote infinity.
- `const Init & init` = `Init::get_instance()`  
`qpp::Init` const Singleton
- `const Codes & codes` = `Codes::get_instance()`  
`qpp::Codes` const Singleton
- `const Gates & gt` = `Gates::get_instance()`  
`qpp::Gates` const Singleton
- `const States & st` = `States::get_instance()`  
`qpp::States` const Singleton
- `RandomDevices & rdevs` = `RandomDevices::get_instance()`  
`qpp::RandomDevices` Singleton

### 6.1.1 Typedef Documentation

#### 6.1.1.1 `using qpp::bra = typedef DynRowVect<cplx>`

Complex (double precision) dynamic Eigen row vector.

#### 6.1.1.2 `using qpp::cmat = typedef DynMat<cplx>`

Complex (double precision) dynamic Eigen matrix.

#### 6.1.1.3 `using qpp::cplx = typedef std::complex<double>`

Complex number in double precision.

#### 6.1.1.4 `using qpp::dmat = typedef DynMat<double>`

Real (double precision) dynamic Eigen matrix.

**6.1.1.5** `template<typename Scalar > using qpp::DynColVect = typedef Eigen::Matrix<Scalar, Eigen::Dynamic, 1>`

Dynamic Eigen column vector over the field specified by *Scalar*.

Example:

```
auto colvect = DynColVect<float>(2); // type of colvect is Eigen::Matrix<float, Eigen::Dynamic, 1>
```

**6.1.1.6** `template<typename Scalar > using qpp::DynMat = typedef Eigen::Matrix<Scalar, Eigen::Dynamic, Eigen::Dynamic>`

Dynamic Eigen matrix over the field specified by *Scalar*.

Example:

```
auto mat = DynMat<float>(2,3); // type of mat is Eigen::Matrix<float, Eigen::Dynamic, Eigen::Dynamic>
```

**6.1.1.7** `template<typename Scalar > using qpp::DynRowVect = typedef Eigen::Matrix<Scalar, 1, Eigen::Dynamic>`

Dynamic Eigen row vector over the field specified by *Scalar*.

Example:

```
auto rowvect = DynRowVect<float>(3); // type of rowvect is Eigen::Matrix<float, 1, Eigen::Dynamic>
```

**6.1.1.8** `using qpp::ket = typedef DynColVect<cplx>`

Complex (double precision) dynamic Eigen column vector.

## 6.1.2 Function Documentation

**6.1.2.1** `template<typename Derived > cmat qpp::absm ( const Eigen::MatrixBase< Derived > & A )`

Matrix absolut value.

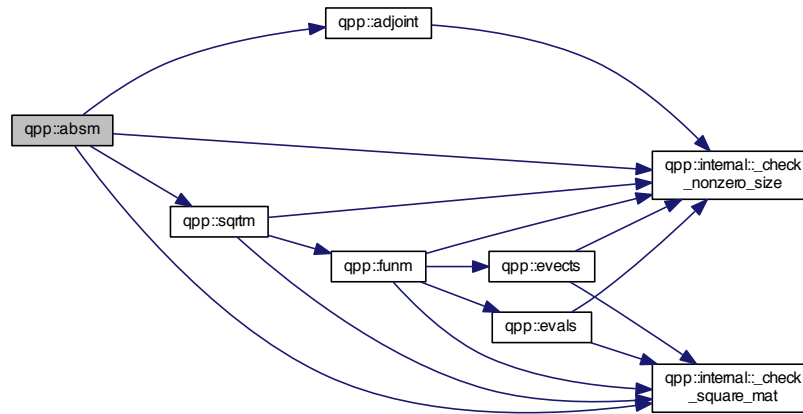
Parameters

A	Eigen expression
---	------------------

## Returns

Matrix absolut value of  $A$

Here is the call graph for this function:



### 6.1.2.2 `template<typename InputIterator> std::vector<double> qpp::abssq ( InputIterator first, InputIterator last )`

Computes the absolut values squared of a range of complex numbers.

## Parameters

<i>first</i>	Iterator to the first element of the range
<i>last</i>	Iterator to the last element of the range

## Returns

Real vector consisting of the range's absolut values squared

### 6.1.2.3 `template<typename Derived> std::vector<double> qpp::abssq ( const Eigen::MatrixBase< Derived> & V )`

Computes the absolut values squared of a column vector.

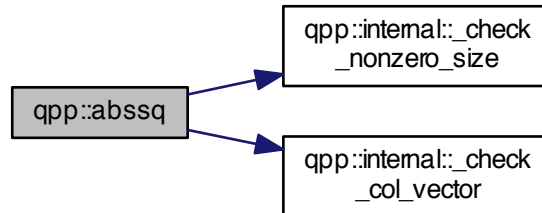
## Parameters

<i>V</i>	Eigen expression
----------	------------------

**Returns**

Real vector consisting of the absolut values squared

Here is the call graph for this function:



**6.1.2.4** `template<typename Derived > DynMat<typename Derived::Scalar> qpp::adjoint ( const Eigen::MatrixBase< Derived > & A )`

Adjoint.

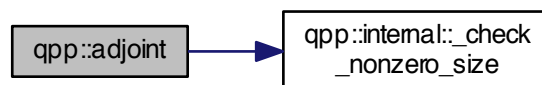
**Parameters**

<code>A</code>	Eigen expression
----------------	------------------

**Returns**

Adjoint (Hermitian conjugate) of  $A$ , as a dynamic matrix over the same scalar field as  $A$

Here is the call graph for this function:



**6.1.2.5** `template<typename Derived1 , typename Derived2 > DynMat<typename Derived1::Scalar> qpp::anticomm ( const Eigen::MatrixBase< Derived1 > & A, const Eigen::MatrixBase< Derived2 > & B )`

Anti-commutator.

Anti-commutator  $\{A, B\} = AB + BA$

Both  $A$  and  $B$  must be Eigen expressions over the same scalar field

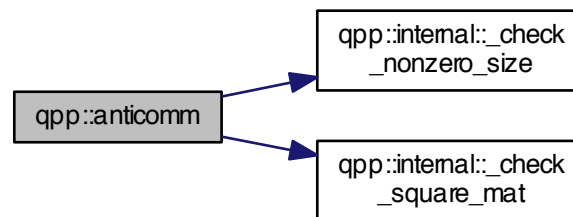
## Parameters

<i>A</i>	Eigen expression
<i>B</i>	Eigen expression

## Returns

Anti-commutator  $AB + BA$ , as a dynamic matrix over the same scalar field as *A*

Here is the call graph for this function:



6.1.2.6 `template<typename Derived1 , typename Derived2 > DynMat<typename Derived1::Scalar> qpp::apply ( const Eigen::MatrixBase< Derived1 > & state, const Eigen::MatrixBase< Derived2 > & A, const std::vector< std::size_t > & subsys, std::size_t n, std::size_t d = 2 )`

Applies the gate *A* to the part *subsys* of a multipartite state vector or density matrix.

## Note

The dimension of the gate *A* must match the dimension of *subsys*

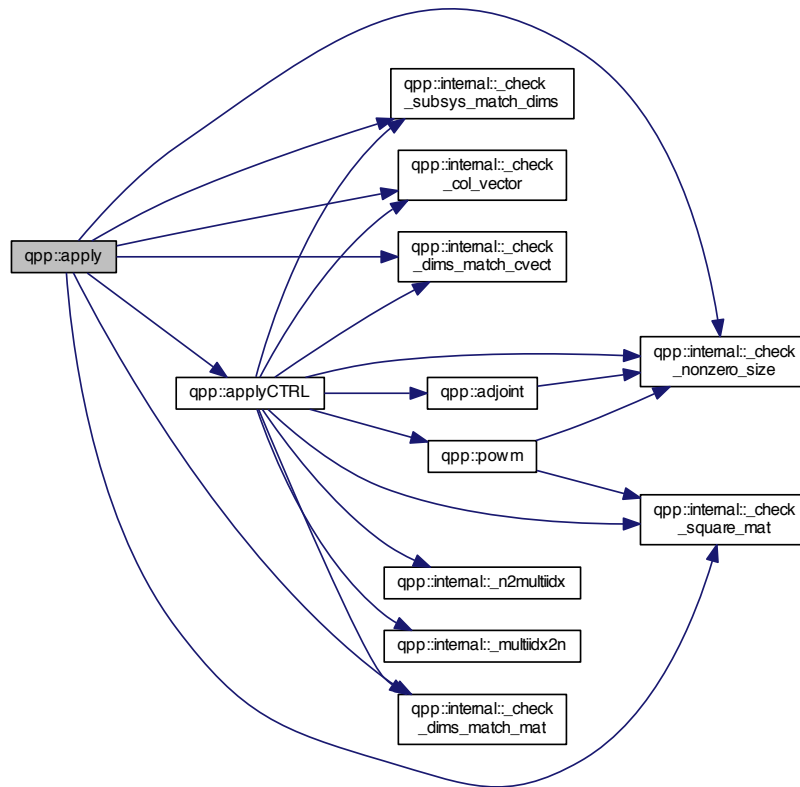
## Parameters

<i>state</i>	Eigen expression
<i>A</i>	Eigen expression
<i>subsys</i>	Subsystem indexes where the gate <i>A</i> is applied
<i>n</i>	Total number of subsystems
<i>d</i>	Local dimensions of all local Hilbert spaces (must all be equal)

## Returns

Gate  $A$  applied to the part *subsys* of *state*

Here is the call graph for this function:



```

6.1.2.7 template<typename Derived1 , typename Derived2 > DynMat<typename Derived1::Scalar> qpp::applyCTRL ( const
Eigen::MatrixBase< Derived1 > & state, const Eigen::MatrixBase< Derived2 > & A, const std::vector< std::size_t >
& ctrl, const std::vector< std::size_t > & subsys, std::size_t n, std::size_t d = 2 )

```

Applies the controlled-gate  $A$  to the part *subsys* of a multipartite state vector or density matrix.

## Note

The dimension of the gate  $A$  must match the dimension of *subsys*

## Parameters

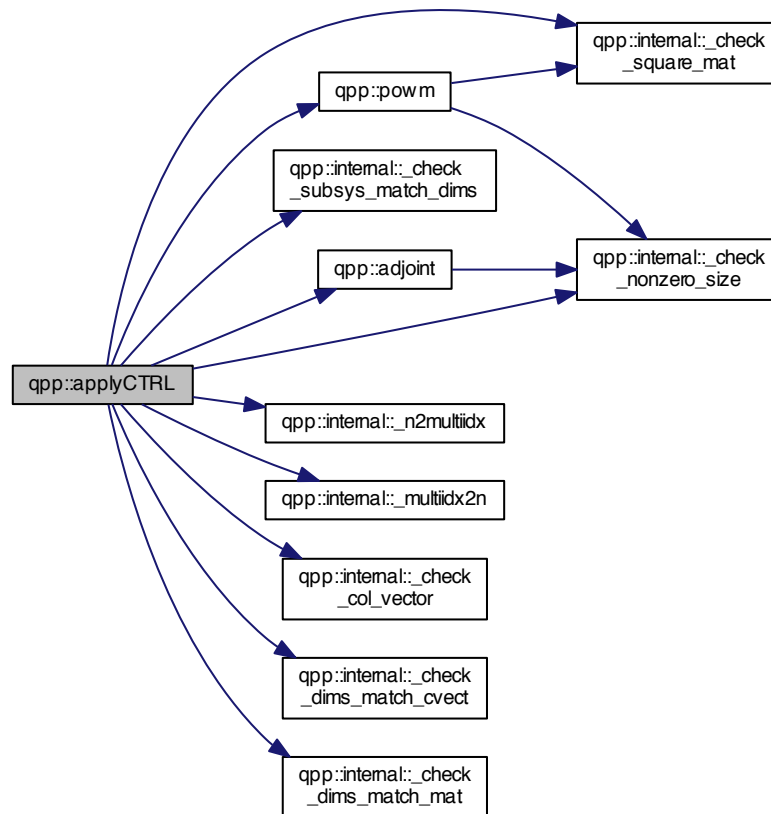
<i>state</i>	Eigen expression
<i>A</i>	Eigen expression
<i>ctrl</i>	Control subsystem indexes
<i>subsys</i>	Subsystem indexes where the gate $A$ is applied

$n$	Total number of subsystems
$d$	Local dimensions of all local Hilbert spaces (must all be equal)

**Returns**

CTRL-A gate applied to the part *subsys* of *state*

Here is the call graph for this function:



**6.1.2.8** `template<typename Derived > cmat qpp::channel ( const Eigen::MatrixBase< Derived > & rho, const std::vector< cmat > & Ks )`

Applies the channel specified by the set of Kraus operators *Ks* to the density matrix *rho*.

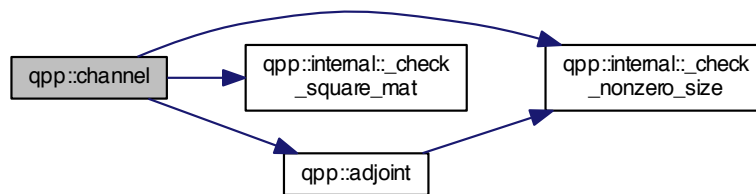
**Parameters**

<i>rho</i>	Eigen expression
<i>Ks</i>	Set of Kraus operators

**Returns**

Output density matrix after the action of the channel

Here is the call graph for this function:



**6.1.2.9** `template<typename Derived> cmat qpp::channel ( const Eigen::MatrixBase< Derived> & rho, const std::vector< cmat> & Ks, const std::vector< std::size_t> & subsys, std::size_t n, std::size_t d = 2 )`

Applies the channel specified by the set of Kraus operators *Ks* to the part of the density matrix *rho* specified by *subsys*.

**Parameters**

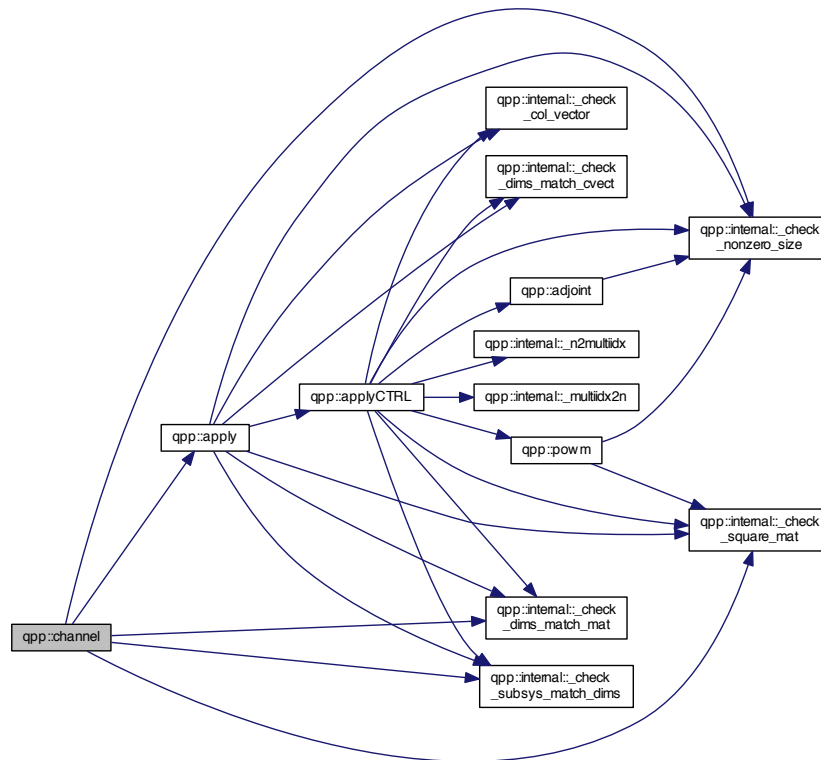
<i>rho</i>	Eigen expression
<i>Ks</i>	Set of Kraus operators
<i>subsys</i>	Subsystems' indexes
<i>n</i>	Total number of subsystems
<i>d</i>	Local dimensions of all local Hilbert spaces (must all be equal)

**Returns**

Output density matrix after the action of the channel



Here is the call graph for this function:



#### 6.1.2.10 cmat qpp::choi ( const std::vector< cmat > & Ks )

Choi matrix representation.

Constructs the Choi matrix of the channel specified by the set of Kraus operators  $K_s$  in the standard operator basis  $\{|i\rangle\langle j|\}$  ordered in lexicographical order, i.e.  $|0\rangle\langle 0|$ ,  $|0\rangle\langle 1|$  etc.

#### Note

The superoperator matrix  $S$  and the Choi matrix  $C$  are related by  $S_{ab,mn} = C_{ma,nb}$

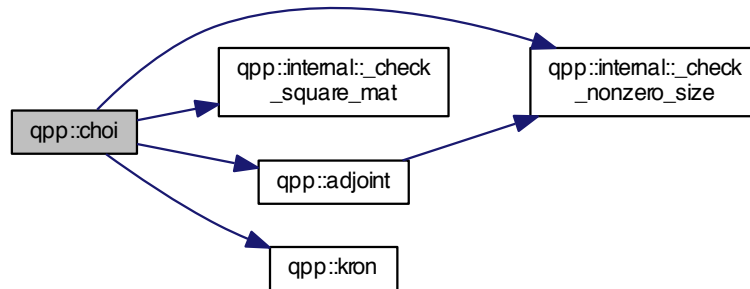
#### Parameters

$K_s$	Set of Kraus operators
-------	------------------------

**Returns**

Choi matrix representation

Here is the call graph for this function:



### 6.1.2.11 `std::vector<cmat> qpp::choi2kraus ( const cmat & A )`

Extracts orthogonal Kraus operators from Choi matrix.

Extracts a set of orthogonal (under Hilbert-Schmidt operator norm) Kraus operators from the Choi representation  $A$  of the channel

**Note**

The Kraus operators satisfy  $Tr(K_i^\dagger K_j) = \delta_{ij}$  for all  $i \neq j$

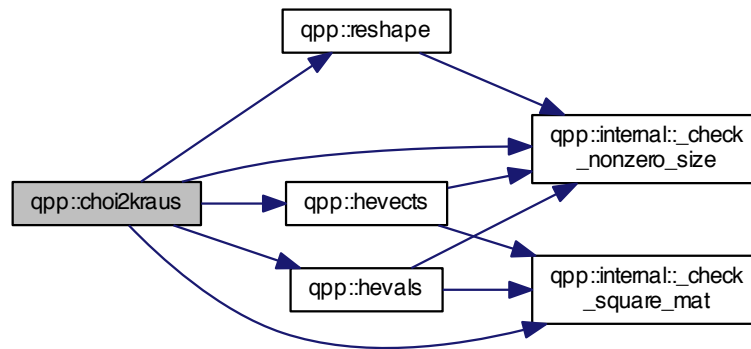
**Parameters**

$A$	Choi matrix
-----	-------------

## Returns

Set of Kraus operators

Here is the call graph for this function:



6.1.2.12 `template<typename Derived1 , typename Derived2 > DynMat<typename Derived1::Scalar> qpp::comm ( const Eigen::MatrixBase< Derived1 > & A, const Eigen::MatrixBase< Derived2 > & B )`

Commutator.

Commutator  $[A, B] = AB - BA$

Both  $A$  and  $B$  must be Eigen expressions over the same scalar field

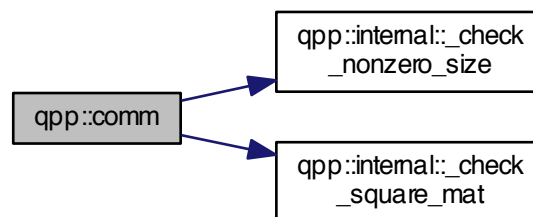
## Parameters

$A$	Eigen expression
$B$	Eigen expression

## Returns

Commutator  $AB - BA$ , as a dynamic matrix over the same scalar field as  $A$

Here is the call graph for this function:



6.1.2.13 `std::vector<std::size_t> qpp::compperm ( const std::vector< std::size_t > & perm, const std::vector< std::size_t > & sigma )`

Compose permutations.

## Parameters

<i>perm</i>	Permutation
<i>sigma</i>	Permutation

## Returns

Composition of the permutations  $perm \circ sigma = perm(sigma)$

Here is the call graph for this function:



#### 6.1.2.14 `template<typename Derived> double qpp::concurrence ( const Eigen::MatrixBase< Derived> & A )`

Wootters concurrence of the bi-partite qubit mixed state *A*.

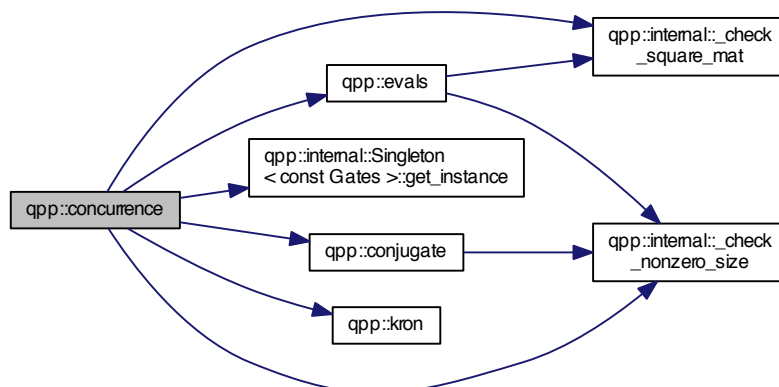
## Parameters

<i>A</i>	Eigen expression
----------	------------------

## Returns

Wootters concurrence

Here is the call graph for this function:



6.1.2.15 `template<typename Derived > DynMat<typename Derived::Scalar> qpp::conjugate ( const Eigen::MatrixBase<Derived > & A )`

Complex conjugate.

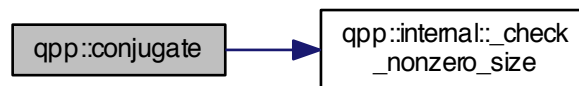
## Parameters

$A$	Eigen expression
-----	------------------

## Returns

Complex conjugate of  $A$ , as a dynamic matrix over the same scalar field as  $A$

Here is the call graph for this function:



#### 6.1.2.16 `template<typename Derived> cmat qpp::cosm ( const Eigen::MatrixBase< Derived> & A )`

Matrix cos.

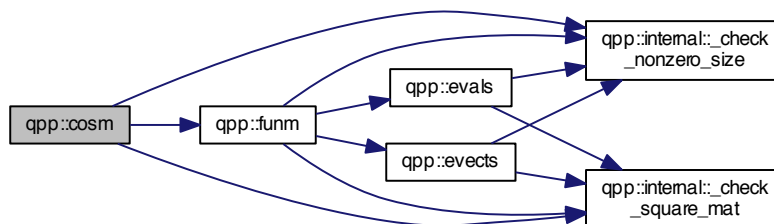
## Parameters

$A$	Eigen expression
-----	------------------

## Returns

Matrix cosine of  $A$

Here is the call graph for this function:



#### 6.1.2.17 `template<typename OutputScalar, typename Derived> DynMat<OutputScalar> qpp::cwise ( const Eigen::MatrixBase< Derived> & A, OutputScalar (*)(const typename Derived::Scalar &) f )`

Functor.

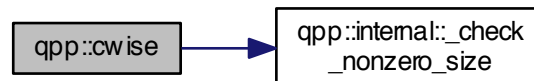
## Parameters

$A$	Eigen expression
$f$	Pointer-to-function from scalars of $A$ to <i>OutputScalar</i>

## Returns

Component-wise  $f(A)$ , as a dynamic matrix over the *OutputScalar* scalar field

Here is the call graph for this function:



6.1.2.18 `template<typename Derived> Derived::Scalar qpp::det ( const Eigen::MatrixBase< Derived> & A )`

Determinant.

## Parameters

$A$	Eigen expression
-----	------------------

## Returns

Determinant of  $A$ , as a scalar in the same scalar field as  $A$  Returns  $\pm\infty$  when the determinant overflows/underflows

Here is the call graph for this function:



6.1.2.19 `template<typename InputIterator> void qpp::disp ( const InputIterator & first, const InputIterator & last, const std::string & separator, const std::string & start = " [", const std::string & end = " ] ", std::ostream & os = std::cout )`

Displays a range. Does not add a newline.

See also

[\*qpp::displn\(\)\*](#)



## Parameters

<i>first</i>	Iterator to the first element of the range
<i>last</i>	Iterator to the last element of the range
<i>separator</i>	Separator
<i>start</i>	Left marking
<i>end</i>	Right marking
<i>os</i>	Output stream

## Returns

Output stream

6.1.2.20 `template<typename T> std::ostream& qpp::disp ( const T & x, const std::string & separator, const std::string & start = " [ ", const std::string & end = " ] ", std::ostream & os = std::cout )`

Displays a standard container that supports `std::begin`, `std::end` and forward iteration. Does not add a newline.

## See also

[\*qpp::displn\(\)\*](#)

## Parameters

<i>x</i>	Container
<i>separator</i>	Separator
<i>start</i>	Left marking
<i>end</i>	Right marking
<i>os</i>	Output stream

## Returns

Output stream

Here is the call graph for this function:



6.1.2.21 `template<typename T> std::ostream& qpp::disp ( const T * x, const std::size_t n, const std::string & separator, const std::string & start = " [ ", const std::string & end = " ] ", std::ostream & os = std::cout )`

Displays a C-style array. Does not add a newline.

## See also

[\*qpp::displn\(\)\*](#)

## Parameters

<i>x</i>	Pointer to the first element
<i>n</i>	Number of elements to be displayed
<i>separator</i>	Separator
<i>start</i>	Left marking
<i>end</i>	Right marking
<i>os</i>	Output stream

## Returns

Output stream

**6.1.2.22** `template<typename Derived> std::ostream& qpp::disp ( const Eigen::MatrixBase< Derived> & A, double chop = qpp::chop, std::ostream & os = std::cout )`

Displays an Eigen expression in matrix friendly form. Does not add a new line.

## See also

[\*qpp::displn\(\)\*](#)

## Parameters

<i>A</i>	Eigen expression
<i>chop</i>	Set to zero the elements smaller in absolute value than <i>chop</i>
<i>os</i>	Output stream

## Returns

Output stream

**6.1.2.23** `std::ostream& qpp::disp ( const cplx z, double chop = qpp::chop, std::ostream & os = std::cout )`

Displays a number (implicitly converted to `std::complex<double>`) in friendly form. Does not add a new line.

## See also

[\*qpp::displn\(\)\*](#)

## Parameters

<i>z</i>	Real/complex number
<i>chop</i>	Set to zero the elements smaller in absolute value than <i>chop</i>
<i>os</i>	Output stream

## Returns

Output stream

Here is the call graph for this function:



**6.1.2.24** `template<typename InputIterator > std::ostream& qpp::displn ( const InputIterator & first, const InputIterator & last, const std::string & separator, const std::string & start = " [ ", const std::string & end = " ] ", std::ostream & os = std::cout )`

Displays a range. Adds a newline.

## See also

[\*qpp::disp\(\)\*](#)

## Parameters

<i>first</i>	Iterator to the first element of the range
<i>last</i>	Iterator to the last element of the range
<i>separator</i>	Separator
<i>start</i>	Left marking
<i>end</i>	Right marking
<i>os</i>	Output stream

## Returns

Output stream

Here is the call graph for this function:



**6.1.2.25** `template<typename T > std::ostream& qpp::displn ( const T & x, const std::string & separator, const std::string & start = " [ ", const std::string & end = " ] ", std::ostream & os = std::cout )`

Displays a standard container that supports std::begin, std::end and forward iteration. Adds a newline.

See also

[\*qpp::disp\(\)\*](#)

Parameters

<i>x</i>	Container
<i>separator</i>	Separator
<i>start</i>	Left marking
<i>end</i>	Right marking
<i>os</i>	Output stream

Returns

Output stream

Here is the call graph for this function:



**6.1.2.26** `template<typename T> std::ostream& qpp::displn ( const T * x, const std::size_t n, const std::string & separator,  
const std::string & start = " [ ", const std::string & end = " ] ", std::ostream & os = std::cout )`

Displays a C-style array. Adds a newline.

See also

[\*qpp::disp\(\)\*](#)

Parameters

<i>x</i>	Pointer to the first element
<i>n</i>	Number of elements to be displayed
<i>separator</i>	Separator
<i>start</i>	Left marking
<i>end</i>	Right marking
<i>os</i>	Output stream

## Returns

Output stream

Here is the call graph for this function:



**6.1.2.27** `template<typename Derived> std::ostream& qpp::displn ( const Eigen::MatrixBase< Derived> & A, double chop = qpp::chop, std::ostream & os = std::cout )`

Displays an Eigen expression in matrix friendly form. Adds a newline.

## See also

[\*qpp::disp\(\)\*](#)

## Parameters

<i>A</i>	Eigen expression
<i>chop</i>	Set to zero the elements smaller in absolute value than <i>chop</i>
<i>os</i>	Output stream

## Returns

Output stream

Here is the call graph for this function:



**6.1.2.28** `std::ostream& qpp::displn ( const cplx z, double chop = qpp::chop, std::ostream & os = std::cout )`

Displays a number (implicitly converted to `std::complex<double>`) in friendly form. Adds a new line.

## See also

[\*qpp::disp\(\)\*](#)

## Parameters

<i>z</i>	Real/complex number
<i>chop</i>	Set to zero the elements smaller in absolute value than <i>chop</i>
<i>os</i>	Output stream

## Returns

Output stream

Here is the call graph for this function:



6.1.2.29 `template<typename Derived> double qpp::entanglement ( const Eigen::MatrixBase< Derived> & A, const std::vector< std::size_t> & dims )`

Entanglement of the bi-partite pure state *A*.

Defined as the von-Neumann entropy of the reduced density matrix of one of the subsystems

## See also

[qpp::shannon\(\)](#)

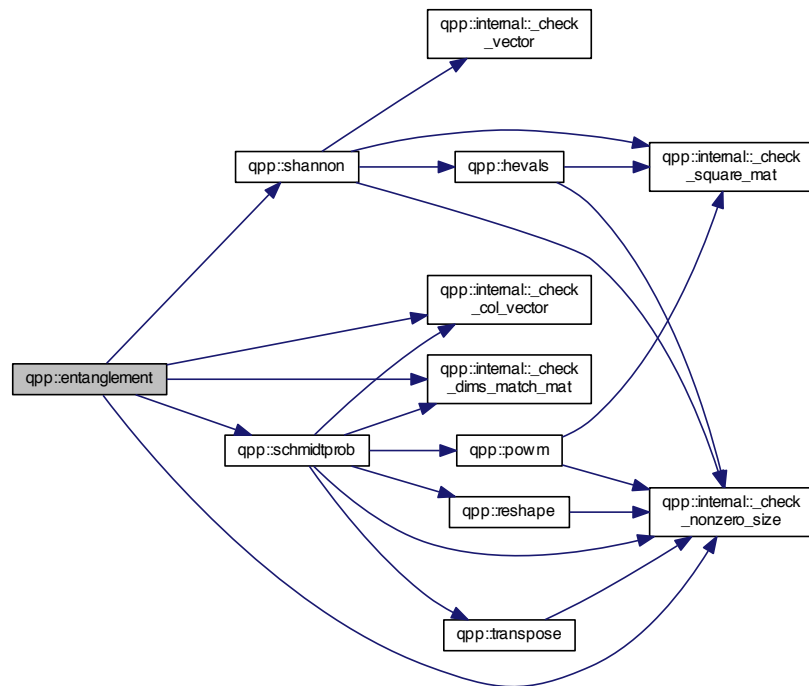
## Parameters

<i>A</i>	Eigen expression
<i>dims</i>	Subsystems' dimensions

## Returns

Entanglement, with the logarithm in base 2

Here is the call graph for this function:



6.1.2.30 `template<typename Derived> DynColVect<cplx> qpp::evals ( const Eigen::MatrixBase< Derived> & A )`

Eigenvalues.

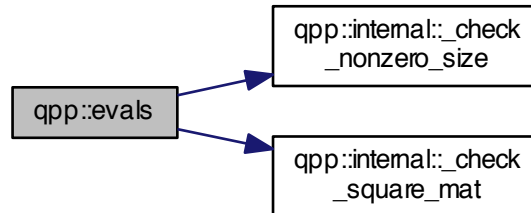
## Parameters

A	Eigen expression
---	------------------

**Returns**

Eigenvalues of  $A$ , as a complex dynamic column vector

Here is the call graph for this function:



**6.1.2.31** `template<typename Derived> cmat qpp::evecs ( const Eigen::MatrixBase< Derived> & A )`

Eigenvectors.

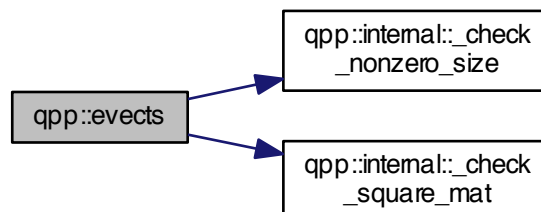
**Parameters**

$A$	Eigen expression
-----	------------------

**Returns**

Eigenvectors of  $A$ , as columns of a complex matrix

Here is the call graph for this function:



**6.1.2.32** `template<typename Derived> cmat qpp::expm ( const Eigen::MatrixBase< Derived> & A )`

Matrix exponential.



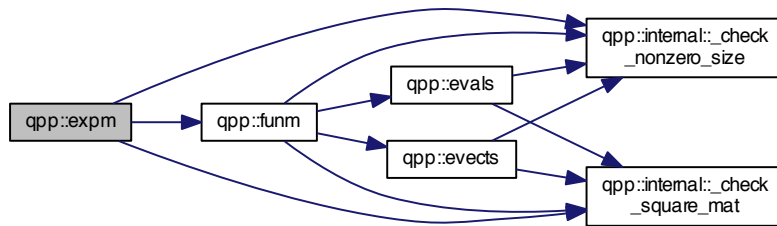
## Parameters

$A$	Eigen expression
-----	------------------

## Returns

Matrix exponential of  $A$

Here is the call graph for this function:



6.1.2.33 `template<typename Derived> cmat qpp::funm ( const Eigen::MatrixBase< Derived> & A, cplx(*) (const cplx &) f )`

Functional calculus  $f(A)$

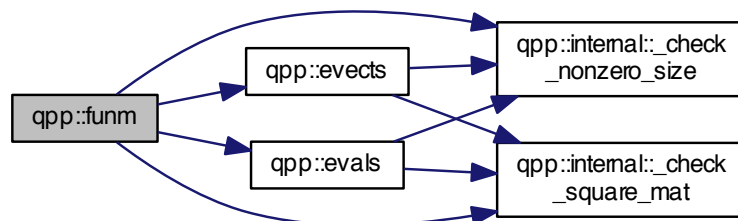
## Parameters

$A$	Eigen expression
$f$	Pointer-to-function from complex to complex

## Returns

$f(A)$

Here is the call graph for this function:



6.1.2.34 `template<typename Derived> double qpp::gconcurrence ( const Eigen::MatrixBase< Derived> & A )`

G-concurrence of the bi-partite pure state  $A$ .

**Note**

Both local dimensions must be equal

Uses [qpp::logdet\(\)](#) to avoid overflows

**See also**

[qpp::logdet\(\)](#)

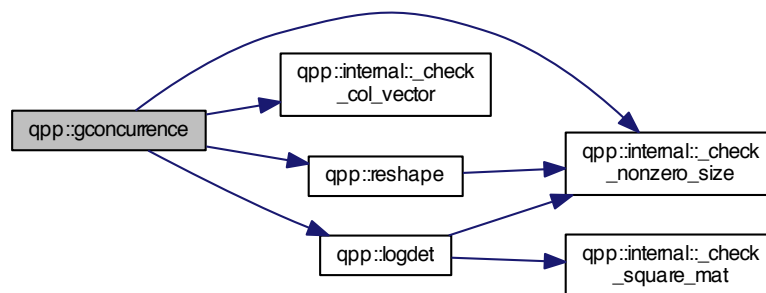
**Parameters**

A	Eigen expression
---	------------------

**Returns**

G-concurrence

Here is the call graph for this function:



**6.1.2.35** `template<typename Derived> DynMat<typename Derived::Scalar> qpp::grams ( const std::vector< Derived> & Vs )`

Gram-Schmidt orthogonalization (std::vector overload)

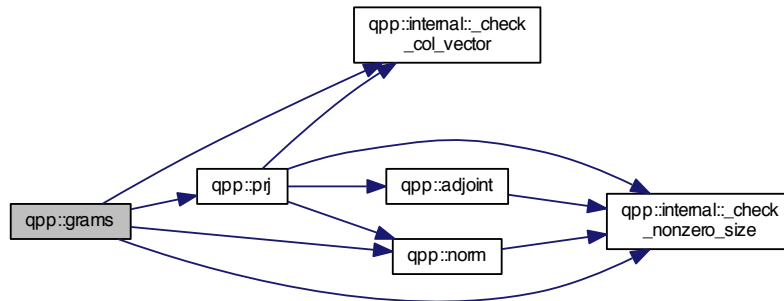
**Parameters**

Vs	std::vector of Eigen expressions as column vectors
----	--

## Returns

Gram-Schmidt vectors of  $V$ s as columns of a dynamic matrix over the same scalar field as its arguments

Here is the call graph for this function:



**6.1.2.36** `template<typename Derived> DynMat<typename Derived::Scalar> qpp::grams ( const std::initializer_list<Derived> & Vs )`

Gram-Schmidt orthogonalization (std::initializer\_list overload)

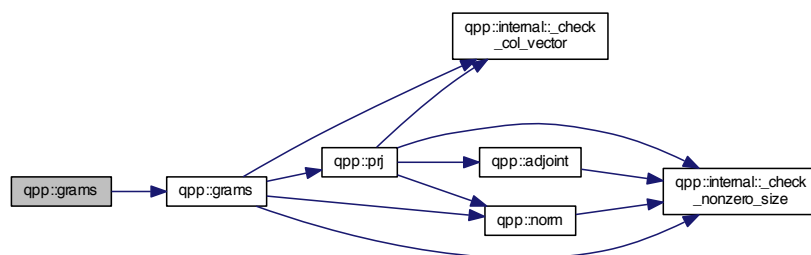
## Parameters

$V$ s	std::initializer_list of Eigen expressions as column vectors
-------	--

## Returns

Gram-Schmidt vectors of  $V$ s as columns of a dynamic matrix over the same scalar field as its arguments

Here is the call graph for this function:



**6.1.2.37** `template<typename Derived> DynMat<typename Derived::Scalar> qpp::grams ( const Eigen::MatrixBase<Derived> & A )`

Gram-Schmidt orthogonalization (Eigen expression (matrix) overload)

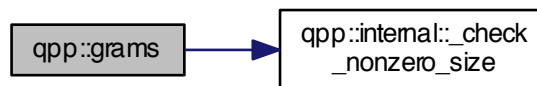
## Parameters

$A$	Eigen expression, the input vectors are the columns of $A$
-----	--

## Returns

Gram-Schmidt vectors of the columns of  $A$ , as columns of a dynamic matrix over the same scalar field as  $A$

Here is the call graph for this function:



**6.1.2.38** `template<typename Derived> DynColVect<double> qpp::hevals ( const Eigen::MatrixBase< Derived> & A )`

Hermitian eigenvalues.

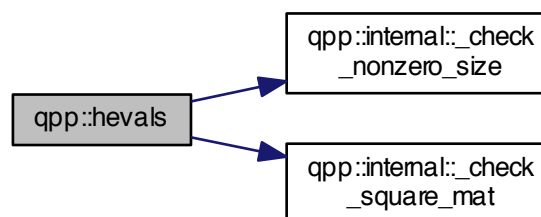
## Parameters

$A$	Eigen expression
-----	------------------

## Returns

Eigenvalues of Hermitian  $A$ , as a real dynamic column vector

Here is the call graph for this function:



**6.1.2.39** `template<typename Derived> cmat qpp::hevects ( const Eigen::MatrixBase< Derived> & A )`

Hermitian eigenvectors.

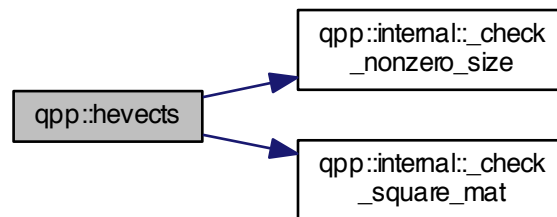
## Parameters

$A$	Eigen expression
-----	------------------

## Returns

Eigenvectors of Hermitian  $A$ , as columns of a complex matrix

Here is the call graph for this function:



6.1.2.40 `template<typename Derived> DynMat<typename Derived::Scalar> qpp::inverse ( const Eigen::MatrixBase<Derived> & A )`

Inverse.

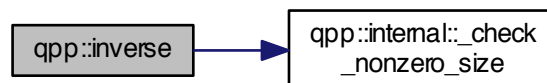
## Parameters

$A$	Eigen expression
-----	------------------

## Returns

Inverse of  $A$ , as a dynamic matrix over the same scalar field as  $A$

Here is the call graph for this function:



6.1.2.41 `std::vector<std::size_t> qpp::invperm ( const std::vector< std::size_t > & perm )`

Inverse permutation.

## Parameters

<i>perm</i>	Permutation
-------------	-------------

## Returns

Inverse of the permutation *perm*

Here is the call graph for this function:



#### 6.1.2.42 `template<typename T > DynMat<typename T::Scalar> qpp::kron ( const T & head )`

Kronecker product (variadic overload)

Used to stop the recursion for the variadic template version of [qpp::kron\(\)](#)

## Parameters

<i>head</i>	Eigen expression
-------------	------------------

## Returns

Its argument *head*

#### 6.1.2.43 `template<typename T , typename... Args> DynMat<typename T::Scalar> qpp::kron ( const T & head, const Args &... tail )`

Kronecker product (variadic overload)

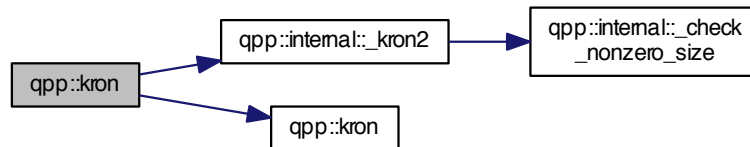
## Parameters

<i>head</i>	Eigen expression
<i>tail</i>	Variadic Eigen expression (zero or more parameters)

**Returns**

Kronecker product of all input parameters, evaluated from left to right, as a dynamic matrix over the same scalar field as its arguments

Here is the call graph for this function:



**6.1.2.44** `template<typename Derived> DynMat<typename Derived::Scalar> qpp::kron ( const std::vector< Derived> & As )`

Kronecker product (std::vector overload)

**Parameters**

<b>As</b>	std::vector of Eigen expressions
-----------	----------------------------------

**Returns**

Kronecker product of all elements in As, evaluated from left to right, as a dynamic matrix over the same scalar field as its arguments

Here is the call graph for this function:



**6.1.2.45** `template<typename Derived> DynMat<typename Derived::Scalar> qpp::kron ( const std::initializer_list< Derived> & As )`

Kronecker product (std::initializer\_list overload)

**Parameters**

<i>As</i>	std::initializer_list of Eigen expressions, such as {A1, A2, ... ,Ak}
-----------	---

**Returns**

Kronecker product of all elements in *As*, evaluated from left to right, as a dynamic matrix over the same scalar field as its arguments

Here is the call graph for this function:



**6.1.2.46** `template<typename Derived> DynMat<typename Derived::Scalar> qpp::kronpow ( const Eigen::MatrixBase<Derived> & A, std::size_t n )`

Kronecker power.

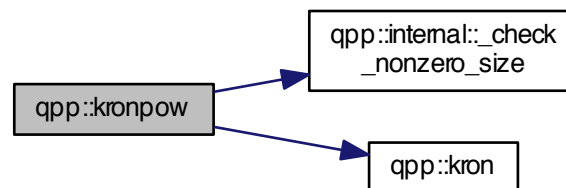
**Parameters**

<i>A</i>	Eigen expression
<i>n</i>	Non-negative integer

**Returns**

Kronecker product of *A* with itself *n* times  $A^{\otimes n}$ , as a dynamic matrix over the same scalar field as *A*

Here is the call graph for this function:



**6.1.2.47** `template<typename Derived> DynMat<typename Derived::Scalar> qpp::load ( const std::string & fname )`

Loads Eigen matrix from a binary file (internal format) in double precision.

The template parameter cannot be automatically deduced and must be explicitly provided, depending on the scalar field of the matrix that is being loaded.

Example:



```
// loads a previously saved Eigen dynamic complex matrix from "input.bin"
auto mat = load<cmat>("input.bin");
```

See also

[qpp::loadMATLABmatrix\(\)](#)

Parameters

<i>A</i>	Eigen expression
<i>fname</i>	Output file name

**6.1.2.48** `template<typename Derived > Derived qpp::loadMATLABmatrix ( const std::string & mat_file, const std::string & var_name )`

Loads an Eigen dynamic matrix from a MATLAB .mat file, generic version.

This is the generic version that always throws [qpp::Exception::Type::UNDEFINED\\_TYPE](#). It is specialized only for [qpp::dmat](#) and [qpp::cmat](#) (the only matrix types that can be loaded)

**6.1.2.49** `template<> dmat qpp::loadMATLABmatrix ( const std::string & mat_file, const std::string & var_name )`

Loads an Eigen dynamic matrix from a MATLAB .mat file, specialization for double matrices ([qpp::dmat](#))

The template parameter cannot be automatically deduced and must be explicitly provided

Example:

```
// loads a previously saved Eigen dynamic double matrix from the
MATLAB file "input.mat"
auto mat = loadMATLABmatrix<dmat>("input.mat");
```

Note

If *var\_name* is a complex matrix, only the real part is loaded

Parameters

<i>mat_file</i>	MATALB .mat file
<i>var_name</i>	Variable name in the .mat file representing the matrix to be loaded

Returns

Eigen double dynamic matrix ([qpp::dmat](#))

**6.1.2.50** `template<> cmat qpp::loadMATLABmatrix ( const std::string & mat_file, const std::string & var_name )`

Loads an Eigen dynamic matrix from a MATLAB .mat file, specialization for complex matrices ([qpp::cmat](#))

The template parameter cannot be automatically deduced and must be explicitly provided

Example:

```
// loads a previously saved Eigen dynamic complex matrix from the
MATLAB file "input.mat"
auto mat = loadMATLABmatrix<cmat>("input.mat");
```

## Parameters

<i>mat_file</i>	MATALB .mat file
<i>var_name</i>	Variable name in the .mat file representing the matrix to be loaded

## Returns

Eigen complex dynamic matrix ([qpp::cmat](#))

6.1.2.51 `template<typename Derived> Derived::Scalar qpp::logdet ( const Eigen::MatrixBase< Derived> & A )`

Logarithm of the determinant.

Especially useful when the determinant overflows/underflows

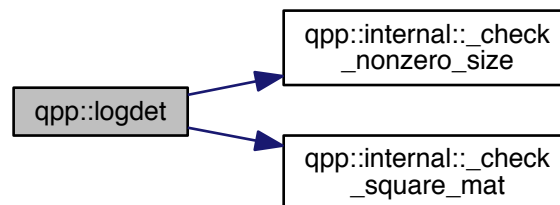
## Parameters

<i>A</i>	Eigen expression
----------	------------------

## Returns

Logarithm of the determinant of *A*, as a scalar in the same scalar field as *A*

Here is the call graph for this function:



6.1.2.52 `template<typename Derived> cmat qpp::logm ( const Eigen::MatrixBase< Derived> & A )`

Matrix logarithm.

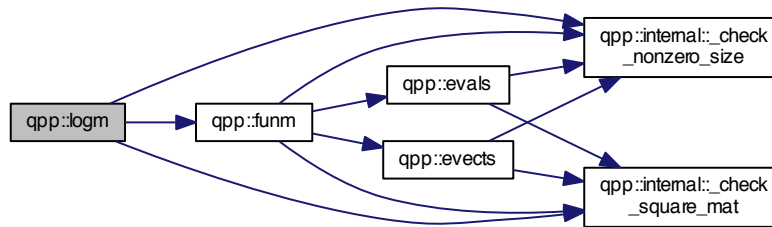
## Parameters

<i>A</i>	Eigen expression
----------	------------------

## Returns

Matrix logarithm of  $A$

Here is the call graph for this function:



**6.1.2.53** `template<typename Derived> double qpp::lognegativity ( const Eigen::MatrixBase< Derived> & A, const std::vector< std::size_t> & dims )`

Logarithmic negativity of the bi-partite mixed state  $A$ .

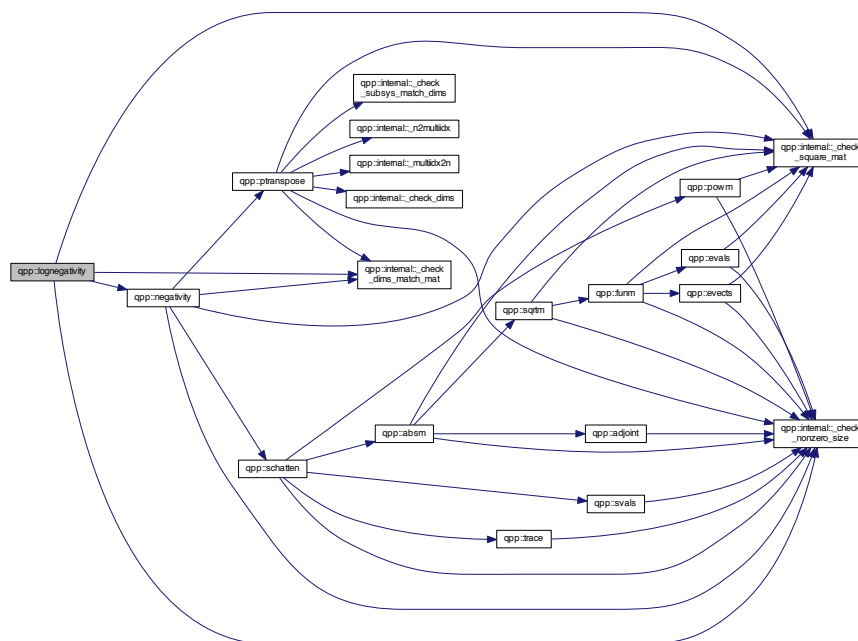
## Parameters

$A$	Eigen expression
$dims$	Subsystems' dimensions

## Returns

Logarithmic negativity, with the logarithm in base 2

Here is the call graph for this function:



6.1.2.54 `template<typename Derived> std::pair<std::vector<double>, std::vector<cmat>> qpp::measure ( const Eigen::MatrixBase< Derived> & A, const std::vector< cmat> & Ks )`

Measures the state  $A$  using the set of Kraus operators  $Ks$ .

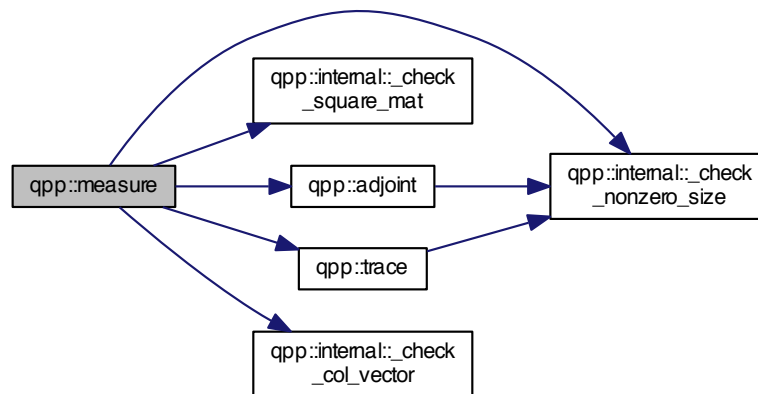
Parameters

$A$	Eigen expression
$Ks$	Set of Kraus operators

Returns

Pair of vector of probabilities and vector of post-measurement normalized states

Here is the call graph for this function:



6.1.2.55 `template<typename Derived> std::pair<std::vector<double>, std::vector<cmat>> qpp::measure ( const Eigen::MatrixBase< Derived> & A, const std::initializer_list< cmat> & Ks )`

Measures the state  $A$  using the set of Kraus operators  $Ks$  (`std::initializer_list` overload)

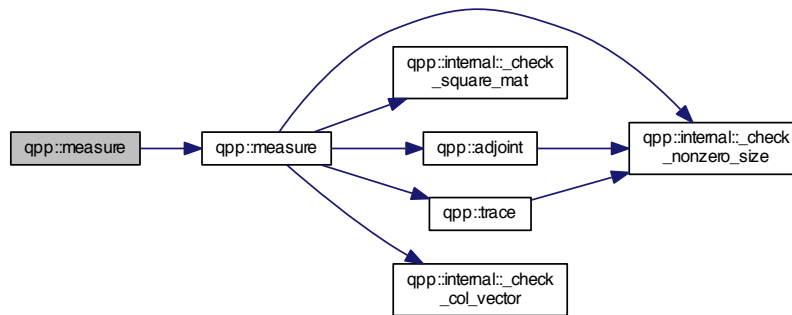
Parameters

$A$	Eigen expression
$Ks$	Set of Kraus operators

## Returns

Pair of vector of probabilities and vector of post-measurement normalized states

Here is the call graph for this function:



6.1.2.56 `template<typename Derived> std::pair<std::vector<double>, std::vector<cmat>> qpp::measure ( const Eigen::MatrixBase< Derived> & A, const cmat & M )`

Measures the state  $A$  in the orthonormal basis specified by the eigenvectors of  $M$ .

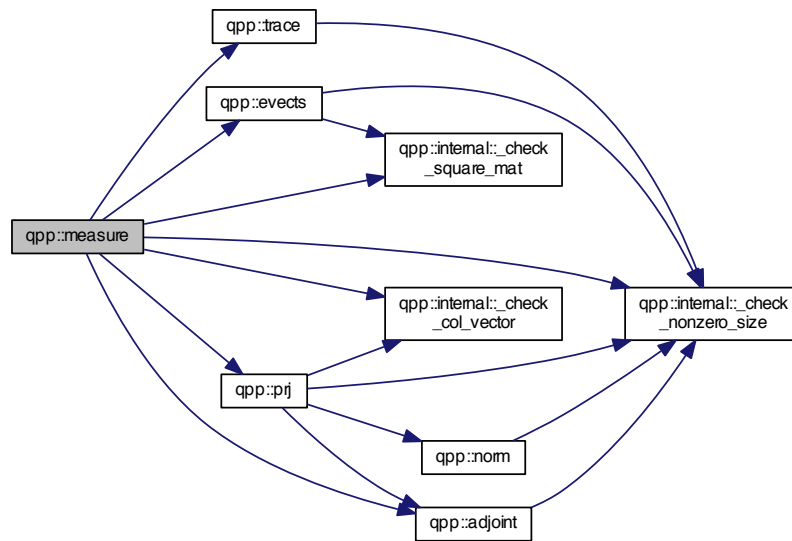
## Parameters

$A$	Eigen expression
$M$	Normal matrix whose eigenvectors define the measurement basis

## Returns

Pair of vector of probabilities and vector of post-measurement normalized states

Here is the call graph for this function:



#### 6.1.2.57 ket qpp::mket ( const std::vector< std::size\_t > & mask, const std::vector< std::size\_t > & dims )

Multi-partite qudit ket (different dimensions overload)

Constructs the multi-partite qudit ket  $|\text{mask}\rangle$ , where *mask* is a std::vector of non-negative integers  
Each element in *mask* has to be smaller than the corresponding element in *dims*

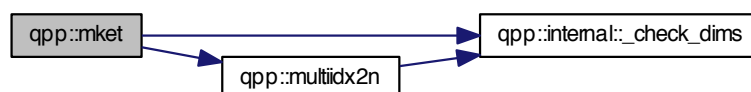
##### Parameters

<i>mask</i>	std::vector of non-negative integers
<i>dims</i>	Dimensions of the multi-partite system

##### Returns

Multi-partite qudit state vector, as a complex dynamic column vector

Here is the call graph for this function:



#### 6.1.2.58 ket qpp::mket ( const std::vector< std::size\_t > & mask, std::size\_t d = 2 )

Multi-partite qudit ket (same dimensions overload)

Constructs the multi-partite qudit ket  $|\text{mask}\rangle$ , all subsystem having equal dimension  $d$   
 $\text{mask}$  is a `std::vector` of non-negative integers, and each element in  $\text{mask}$  has to be strictly smaller than  $d$

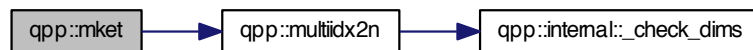
**Parameters**

<i>mask</i>	<code>std::vector</code> of non-negative integers
<i>d</i>	Subsystems' dimension

**Returns**

Multi-partite qudit state vector, as a complex dynamic column vector

Here is the call graph for this function:



#### 6.1.2.59 `cmat qpp::mprj ( const std::vector< std::size_t > & mask, const std::vector< std::size_t > & dims )`

Projector onto multi-partite qudit ket (different dimensions overload)

Constructs the projector onto the multi-partite qudit ket  $|\text{mask}\rangle$ , where  $\text{mask}$  is a `std::vector` of non-negative integers  
 Each element in  $\text{mask}$  has to be smaller than the corresponding element in  $\text{dims}$

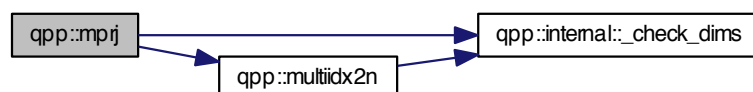
**Parameters**

<i>mask</i>	<code>std::vector</code> of non-negative integers
<i>dims</i>	Dimensions of the multi-partite system

**Returns**

Projector onto multi-partite qudit state vector, as a complex dynamic matrix

Here is the call graph for this function:



#### 6.1.2.60 `cmat qpp::mprj ( const std::vector< std::size_t > & mask, std::size_t d = 2 )`

Projector onto multi-partite qudit ket (same dimensions overload)

Constructs the projector onto the multi-partite qudit ket  $|\text{mask}\rangle$ , all subsystem having equal dimension  $d$   
 $\text{mask}$  is a `std::vector` of non-negative integers, and each element in  $\text{mask}$  has to be strictly smaller than  $d$

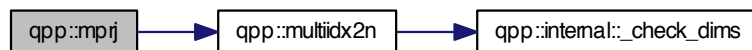
## Parameters

<i>mask</i>	std::vector of non-negative integers
<i>d</i>	Subsystems' dimension

## Returns

Projector onto multi-partite qudit state vector, as a complex dynamic matrix

Here is the call graph for this function:



6.1.2.61 `std::size_t qpp::multiidx2n ( const std::vector< std::size_t > & midx, const std::vector< std::size_t > & dims )`

Multi-index to non-negative integer index.

Uses standard lexicographical order, i.e. 00...0, 00...1 etc.

## Parameters

<i>midx</i>	Multi-index
<i>dims</i>	Dimensions of the multi-partite system

## Returns

Non-negative integer index

Here is the call graph for this function:



6.1.2.62 `std::vector<std::size_t> qpp::n2multiidx ( std::size_t n, const std::vector< std::size_t > & dims )`

Non-negative integer index to multi-index.

Uses standard lexicographical order, i.e. 00...0, 00...1 etc.



## Parameters

<i>n</i>	Non-negative integer index
<i>dims</i>	Dimensions of the multi-partite system

## Returns

Multi-index of the same size as *dims*

Here is the call graph for this function:



6.1.2.63 `template<typename Derived > double qpp::negativity ( const Eigen::MatrixBase< Derived > & A, const std::vector< std::size_t > & dims )`

Negativity of the bi-partite mixed state *A*.

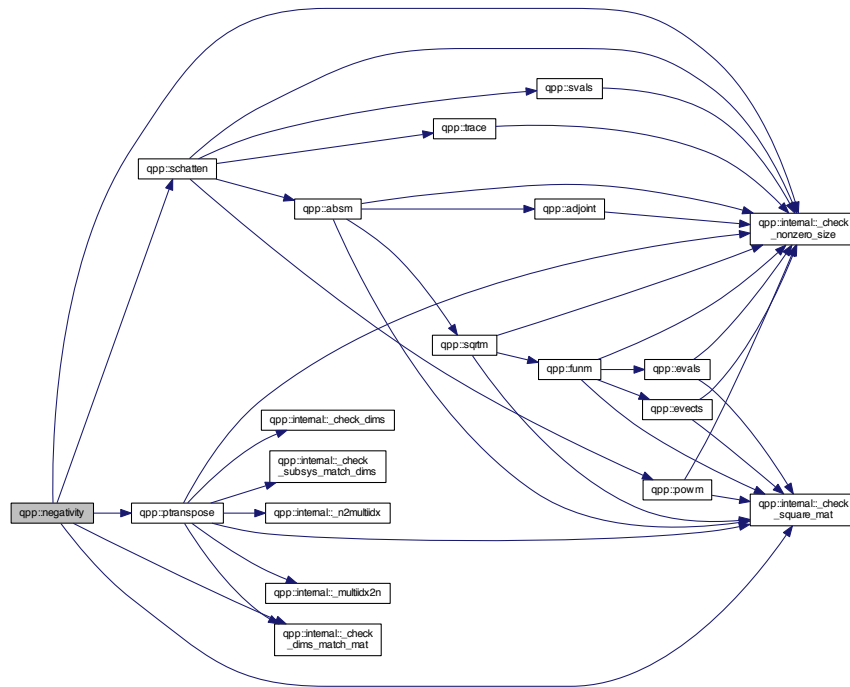
## Parameters

<i>A</i>	Eigen expression
<i>dims</i>	Subsystems' dimensions

## Returns

Negativity

Here is the call graph for this function:



#### 6.1.2.64 `template<typename Derived> double qpp::norm ( const Eigen::MatrixBase< Derived> & A )`

Frobenius norm.

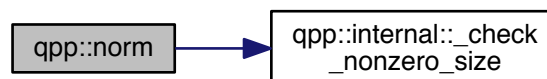
Parameters

<code>A</code>	Eigen expression
----------------	------------------

Returns

Frobenius norm of `A`, as a real number

Here is the call graph for this function:



#### 6.1.2.65 `std::complex<double> qpp::omega ( std::size_t D )`

D-th root of unity.

## Parameters

$D$	Non-negative integer
-----	----------------------

## Returns

D-th root of unity  $\exp(2\pi i/D)$

**6.1.266** `constexpr std::complex<double> qpp::operator""_i ( unsigned long long int x )`

User-defined literal for complex  $i = \sqrt{-1}$  (integer overload)

Example:

```
auto z = 4_i; // type of z is std::complex<double>
```

**6.1.267** `constexpr std::complex<double> qpp::operator""_i ( long double x )`

User-defined literal for complex  $i = \sqrt{-1}$  (real overload)

Example:

```
auto z = 4.5_i; // type of z is std::complex<double>
```

**6.1.268** `template<typename Derived > DynMat<typename Derived::Scalar> qpp::powm ( const Eigen::MatrixBase<Derived > &A, std::size_t n )`

Matrix power.

Explicitly multiplies the matrix  $A$  with itself  $n$  times

By convention  $A^0 = I$

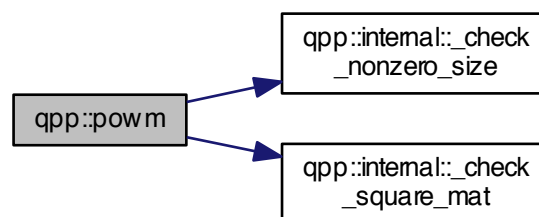
## Parameters

$A$	Eigen expression
$n$	Non-negative integer

## Returns

Matrix power  $A^n$ , as a dynamic matrix over the same scalar field as  $A$

Here is the call graph for this function:



6.1.2.69 `template<typename Derived > DynMat<typename Derived::Scalar> qpp::prj ( const Eigen::MatrixBase< Derived > & V )`

Projector.

Normalized projector onto state vector

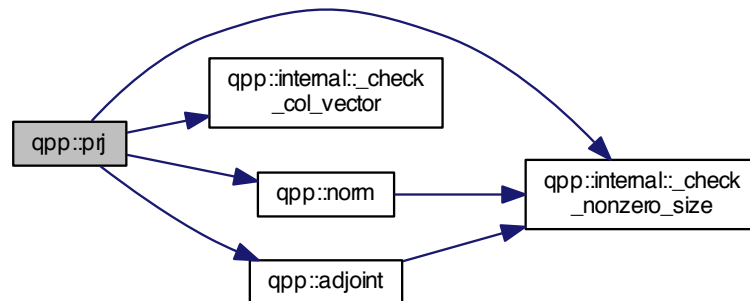
Parameters

<code>V</code>	Eigen expression
----------------	------------------

Returns

Projector onto the state vector  $V$ , or the matrix  $Zero$  if  $V$  has norm zero (i.e. smaller than `qpp::eps`), as a dynamic matrix over the same scalar field as  $A$

Here is the call graph for this function:



6.1.2.70 `template<typename Derived > Derived::Scalar qpp::prod ( const Eigen::MatrixBase< Derived > & A )`

Element-wise product of  $A$ .

Parameters

<code>A</code>	Eigen expression
----------------	------------------

Returns

Element-wise product of  $A$ , as a scalar in the same scalar field as  $A$

Here is the call graph for this function:



6.1.2.71 `template<typename InputIterator > auto qpp::prod ( InputIterator first, InputIterator last ) -> typename InputIterator::value_type`

Element-wise product of a range.

Parameters

<i>first</i>	Iterator to the first element of the range
<i>last</i>	Iterator to the last element of the range

Returns

Element-wise product of the range, as a scalar in the same scalar field as the range

6.1.2.72 `template<typename Derived > DynMat<typename Derived::Scalar> qpp::ptrace ( const Eigen::MatrixBase<Derived > & A, const std::vector< std::size_t > & subsys, const std::vector< std::size_t > & dims )`

Partial trace.

Partial trace of the multi-partite density matrix over a list of subsystems

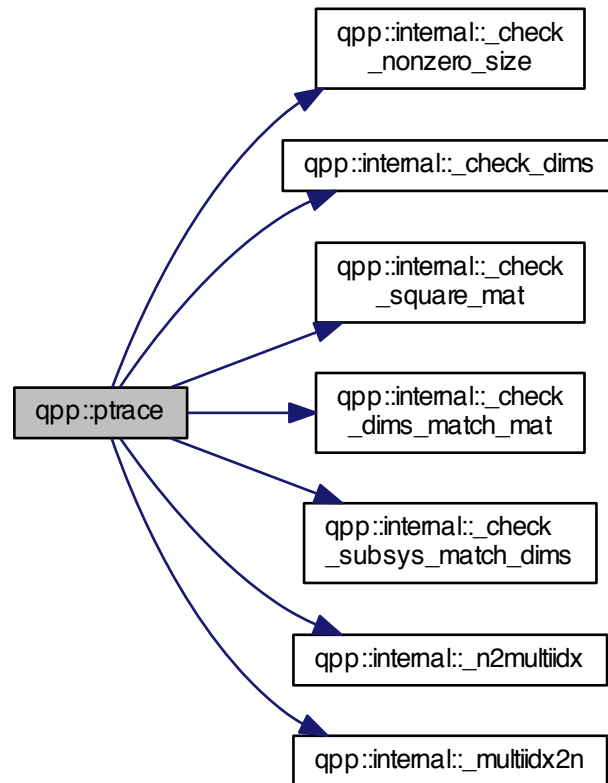
Parameters

<i>A</i>	Eigen expression
<i>subsys</i>	Subsystem indexes
<i>dims</i>	Dimensions of the multi-partite system

Returns

Partial trace  $Tr_{subsys}(\cdot)$  over the subsystems *subsys* in a multi-partite system, as a dynamic matrix over the same scalar field as *A*

Here is the call graph for this function:



**6.1.2.73** `template<typename Derived> DynMat<typename Derived::Scalar> qpp::ptrace1 ( const Eigen::MatrixBase<Derived> & A, const std::vector< std::size_t > & dims )`

Partial trace.

Partial trace of density matrix over the first subsystem in a bi-partite system

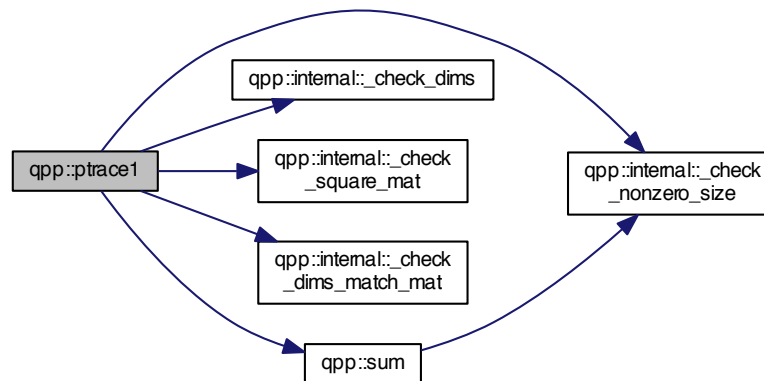
Parameters

<i>A</i>	Eigen expression
<i>dims</i>	Dimensions of bi-partite system (must be a <code>std::vector</code> with 2 elements)

## Returns

Partial trace  $Tr_A(\cdot)$  over the first subsystem  $A$  in a bi-partite system  $A \otimes B$ , as a dynamic matrix over the same scalar field as  $A$

Here is the call graph for this function:



6.1.2.74 `template<typename Derived> DynMat<typename Derived::Scalar> qpp::ptrace2 ( const Eigen::MatrixBase<Derived> & A, const std::vector< std::size_t > & dims )`

Partial trace.

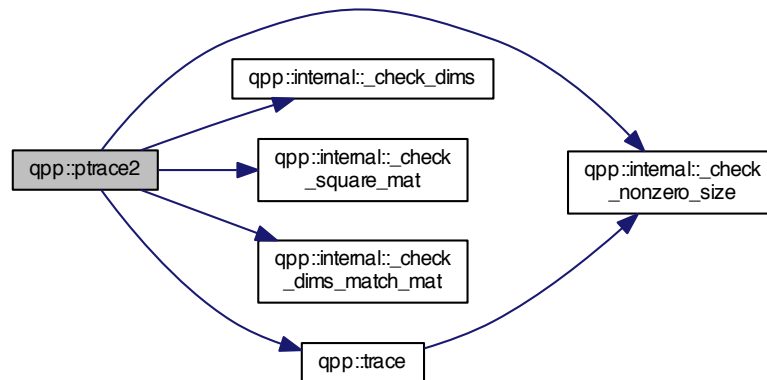
## Parameters

$A$	Eigen expression
$dims$	Dimensions of bi-partite system (must be a <code>std::vector</code> with 2 elements)

## Returns

Partial trace  $Tr_B(\cdot)$  over the second subsystem  $B$  in a bi-partite system  $A \otimes B$ , as a dynamic matrix over the same scalar field as  $A$

Here is the call graph for this function:



**6.1.2.75** `template<typename Derived> DynMat<typename Derived::Scalar> qpp::ptranspose ( const Eigen::MatrixBase<Derived> & A, const std::vector< std::size_t > & subsys, const std::vector< std::size_t > & dims )`

Partial transpose.

Partial transpose of the multi-partite density matrix over a list of subsystems

**Parameters**

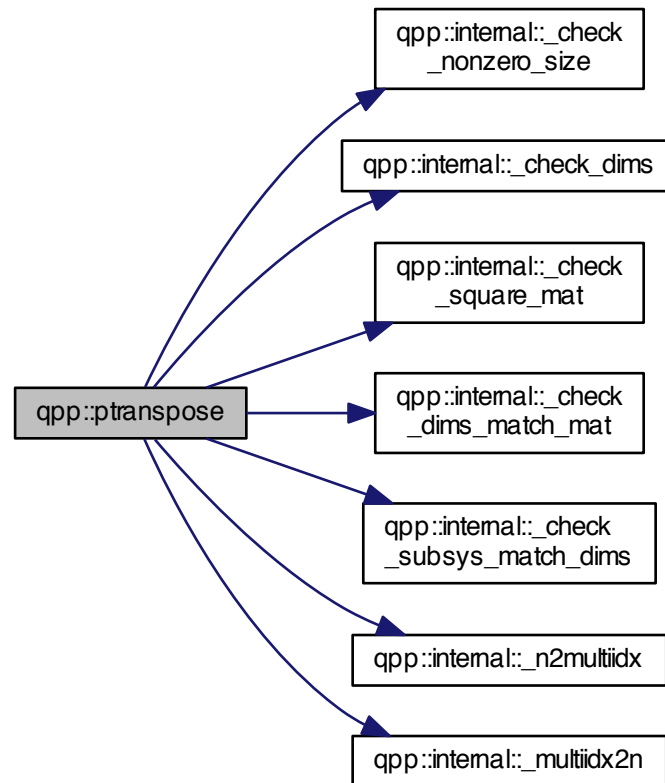
<i>A</i>	Eigen expression
<i>subsys</i>	Subsystem indexes
<i>dims</i>	Dimensions of the multi-partite system

**Returns**

Partial transpose  $(\cdot)^{T_{subsys}}$  over the subsystems *subsys* in a multi-partite system, as a dynamic matrix over the same scalar field as *A*



Here is the call graph for this function:



6.1.2.76 `template<typename Derived> double qpp::qmutualinfo ( const Eigen::MatrixBase< Derived> & A, const std::vector< std::size_t> & subsysA, const std::vector< std::size_t> & subsysB, const std::vector< std::size_t> & dims )`

Quantum mutual information between 2 subsystems of a composite system.

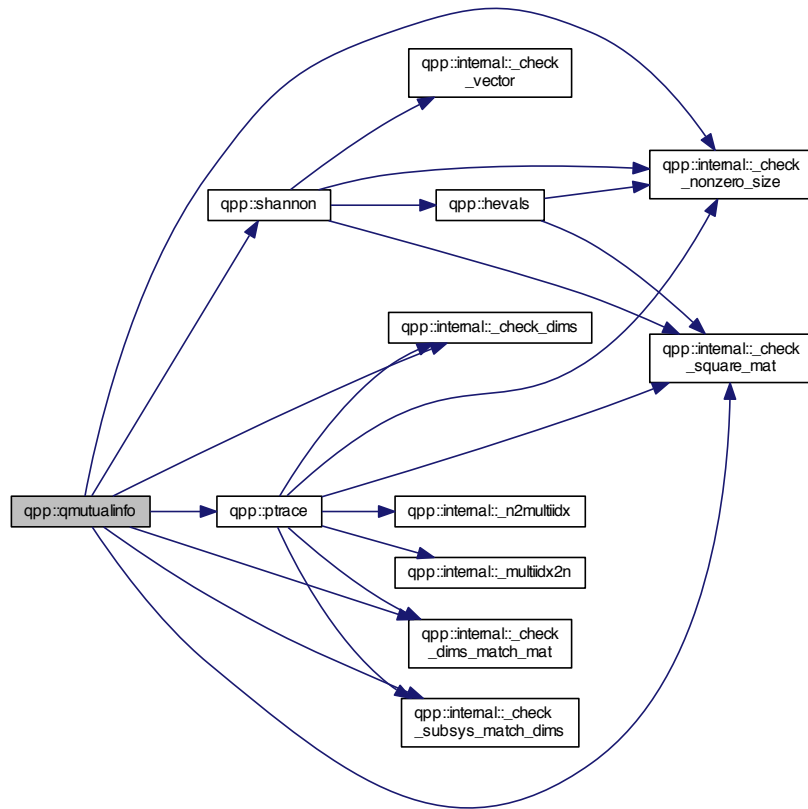
Parameters

<i>A</i>	Eigen expression
<i>subsysA</i>	Indexes of the first subsystem
<i>subsysB</i>	Indexes of the second subsystem
<i>dims</i>	Subsystems' dimensions

## Returns

Mutual information between the 2 subsystems

Here is the call graph for this function:



**6.1.2.77** `template<typename Derived> Derived qpp::rand ( std::size_t rows, std::size_t cols, double a = 0, double b = 1 )`

Generates a random matrix with entries uniformly distributed in the interval [a, b)

If complex, then both real and imaginary parts are uniformly distributed in [a, b)

This is the generic version that always throws `qpp::Exception::Type::UNDEFINED_TYPE`. It is specialized only for `qpp::dmat` and `qpp::cmat`

**6.1.2.78** `template<> dmat qpp::rand ( std::size_t rows, std::size_t cols, double a, double b )`

Generates a random real matrix with entries uniformly distributed in the interval [a, b), specialization for double matrices (`qpp::dmat`)

The template parameter cannot be automatically deduced and must be explicitly provided

Example:

```
// generates a 3 x 3 random Eigen::MatrixXd, with entries uniformly distributed in [-1,1)
auto mat = rand<dmat>(3, 3, -1, 1);
```

## Parameters

<i>rows</i>	Number of rows of the random generated matrix
<i>cols</i>	Number of columns of the random generated matrix
<i>a</i>	Beginning of the interval, belongs to it
<i>b</i>	End of the interval, does not belong to it

## Returns

Random real matrix

6.1.2.79 `template<> cmat qpp::rand ( std::size_t rows, std::size_t cols, double a, double b )`

Generates a random complex matrix with entries (both real and imaginary) uniformly distributed in the interval [a, b), specialization for complex matrices ([qpp::cmat](#))

The template parameter cannot be automatically deduced and must be explicitly provided

Example:

```
// generates a 3 x 3 random Eigen::MatrixXcd, with entries (both real and imaginary) uniformly distributed
// in [-1,1)
auto mat = rand<cmat>(3, 3, -1, 1);
```

## Parameters

<i>rows</i>	Number of rows of the random generated matrix
<i>cols</i>	Number of columns of the random generated matrix
<i>a</i>	Beginning of the interval, belongs to it
<i>b</i>	End of the interval, does not belong to it

## Returns

Random complex matrix

Here is the call graph for this function:



6.1.2.80 `double qpp::rand ( double a = 0, double b = 1 )`

Generates a random real number uniformly distributed in the interval [a, b)

## Parameters

<i>a</i>	Beginning of the interval, belongs to it
<i>b</i>	End of the interval, does not belong to it

**Returns**

Random real number (double) uniformly distributed in the interval [a, b)

Here is the call graph for this function:

**6.1.2.81 cmat qpp::randH ( std::size\_t *D* )**

Generates a random Hermitian matrix.

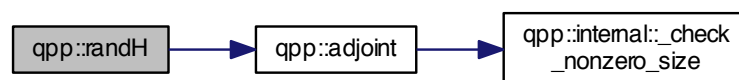
**Parameters**

<i>D</i>	Dimension of the Hilbert space
----------	--------------------------------

**Returns**

Random Hermitian matrix

Here is the call graph for this function:

**6.1.2.82 int qpp::randint ( int *a* = std::numeric\_limits<int>::min(), int *b* = std::numeric\_limits<int>::max() )**

Generates a random integer (int) uniformly distributed in the interval [a, b].

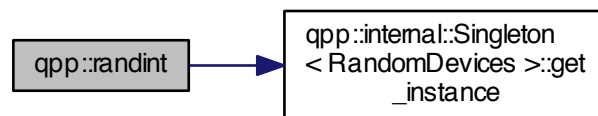
## Parameters

<i>a</i>	Beginning of the interval, belongs to it
<i>b</i>	End of the interval, does not belong to it

## Returns

Random integer (int) uniformly distributed in the interval [a, b]

Here is the call graph for this function:

6.1.2.83 `ket qpp::randket ( std::size_t D )`

Generates a random normalized ket (pure state vector)

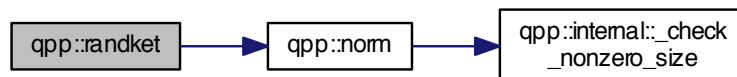
## Parameters

<i>D</i>	Dimension of the Hilbert space
----------	--------------------------------

## Returns

Random normalized ket

Here is the call graph for this function:

6.1.2.84 `std::vector<cmat> qpp::randkraus ( std::size_t N, std::size_t D )`

Generates a set of random Kraus operators.

## Note

The set of Kraus operators satisfy the closure condition  $\sum_i K_i^\dagger K_i = I$

## Parameters

$N$	Number of Kraus operators
$D$	Dimension of the Hilbert space

## Returns

Set of  $N$  Kraus operators satisfying the closure condition

Here is the call graph for this function:



**6.1.2.85** `template<typename Derived > Derived qpp::randn ( std::size_t rows, std::size_t cols, double mean = 0, double sigma = 1 )`

Generates a random matrix with entries normally distributed in  $N(\text{mean}, \text{sigma})$

If complex, then both real and imaginary parts are normally distributed in  $N(\text{mean}, \text{sigma})$

This is the generic version that always throws [`qpp::Exception::Type::UNDEFINED\_TYPE`](#). It is specialized only for [`qpp::dmat`](#) and [`qpp::cmat`](#)

**6.1.2.86** `template<> dmat qpp::randn ( std::size_t rows, std::size_t cols, double mean, double sigma )`

Generates a random real matrix with entries normally distributed in  $N(\text{mean}, \text{sigma})$ , specialization for double matrices ([`qpp::dmat`](#))

The template parameter cannot be automatically deduced and must be explicitly provided

Example:

```
// generates a 3 x 3 random Eigen::MatrixXd, with entries normally distributed in N(0,2)
auto mat = randn<dmat>(3, 3, 0, 2);
```

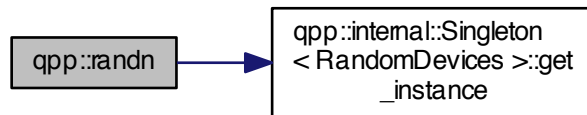
## Parameters

<i>rows</i>	Number of rows of the random generated matrix
<i>cols</i>	Number of columns of the random generated matrix
<i>mean</i>	Mean
<i>sigma</i>	Standard deviation

**Returns**

Random real matrix

Here is the call graph for this function:



### 6.1.2.87 `template<> cmat qpp::randn ( std::size_t rows, std::size_t cols, double mean, double sigma )`

Generates a random complex matrix with entries (both real and imaginary) normally distributed in  $N(\text{mean}, \text{sigma})$ , specialization for complex matrices ([qpp::cmat](#))

The template parameter cannot be automatically deduced and must be explicitly provided

Example:

```
// generates a 3 x 3 random Eigen::MatrixXcd, with entries (both real and imaginary) normally distributed
// in N(0,2)
auto mat = randn<cmat>(3, 3, 0, 2);
```

**Parameters**

<i>rows</i>	Number of rows of the random generated matrix
<i>cols</i>	Number of columns of the random generated matrix
<i>mean</i>	Mean
<i>sigma</i>	Standard deviation

**Returns**

Random complex matrix

Here is the call graph for this function:



### 6.1.2.88 `double qpp::randn ( double mean = 0, double sigma = 1 )`

Generates a random real number (double) normally distributed in  $N(\text{mean}, \text{sigma})$

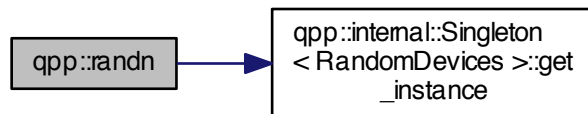
## Parameters

<i>mean</i>	Mean
<i>sigma</i>	Standard deviation

## Returns

Random real number normally distributed in  $N(\text{mean}, \text{sigma})$

Here is the call graph for this function:



#### 6.1.2.89 `std::vector<std::size_t> qpp::randperm ( std::size_t n )`

Generates a random uniformly distributed permutation.

Uses Knuth's shuffle method (as implemented by `std::shuffle`), so that all permutations are equally probable

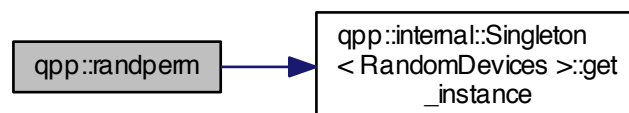
## Parameters

<i>n</i>	Size of the permutation
----------	-------------------------

## Returns

Random permutation of size *n*

Here is the call graph for this function:



#### 6.1.2.90 `cmat qpp::randrho ( std::size_t D )`

Generates a random density matrix.



## Parameters

<i>D</i>	Dimension of the Hilbert space
----------	--------------------------------

## Returns

Random density matrix

6.1.2.91 `cmat qpp::randU ( std::size_t D )`

Generates a random unitary matrix.

## Parameters

<i>D</i>	Dimension of the Hilbert space
----------	--------------------------------

## Returns

Random unitary

6.1.2.92 `cmat qpp::randV ( std::size_t Din, std::size_t Dout )`

Generates a random isometry matrix.

## Parameters

<i>Din</i>	Size of the input Hilbert space
<i>Dout</i>	Size of the output Hilbert space

## Returns

Random isometry matrix

Here is the call graph for this function:

6.1.2.93 `template<typename Derived> double qpp::renyi ( const Eigen::MatrixBase< Derived> & A, double alpha )`

Renyi-  $\alpha$  entropy of the probability distribution/density matrix  $A$ , for  $\alpha \geq 0$ .

## Parameters

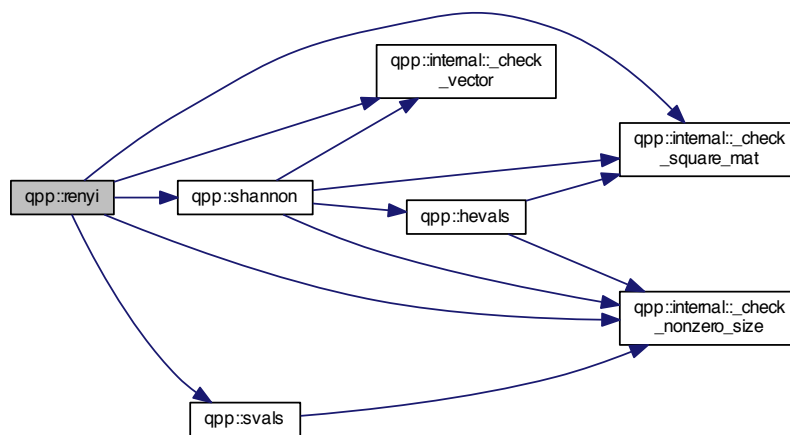
<i>A</i>	Eigen expression, representing a probability distribution (real dynamic column vector) or a density matrix (complex dynamic matrix)
----------	---

<i>alpha</i>	Non-negative real number, use <a href="#">qpp::infy</a> for $\alpha = \infty$
--------------	---

**Returns**

Renyi-  $\alpha$  entropy, with the logarithm in base 2

Here is the call graph for this function:



**6.1.2.94** `template<typename Derived > DynMat<typename Derived::Scalar> qpp::reshape ( const Eigen::MatrixBase<Derived > & A, std::size_t rows, std::size_t cols )`

Reshape.

Uses column-major order when reshaping (same as MATLAB)

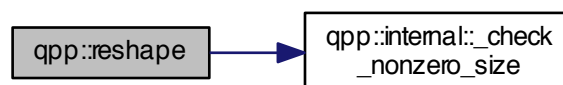
**Parameters**

<i>A</i>	Eigen expression
<i>rows</i>	Number of rows of the reshaped matrix
<i>cols</i>	Number of columns of the reshaped matrix

**Returns**

Reshaped matrix with *rows* rows and *cols* columns, as a dynamic matrix over the same scalar field as *A*

Here is the call graph for this function:



6.1.2.95 `template<typename Derived > void qpp::save ( const Eigen::MatrixBase< Derived > & A, const std::string & fname )`

Saves Eigen expression to a binary file (internal format) in double precision.

See also

[qpp::saveMATLABmatrix\(\)](#)

Parameters

<i>A</i>	Eigen expression
<i>fname</i>	Output file name

6.1.2.96 `template<typename Derived > void qpp::saveMATLABmatrix ( const Eigen::MatrixBase< Derived > & A, const std::string & mat_file, const std::string & var_name, const std::string & mode )`

Saves an Eigen dynamic matrix to a MATLAB .mat file, generic version.

This is the generic version that always throws [qpp::Exception::Type::UNDEFINED\\_TYPE](#). It is specialized only for [qpp::dmat](#) and [qpp::cmat](#) (the only matrix types that can be saved)

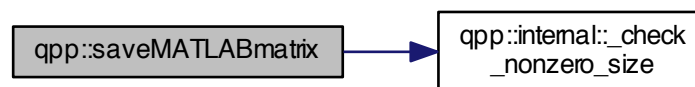
6.1.2.97 `template<> void qpp::saveMATLABmatrix ( const Eigen::MatrixBase< dmat > & A, const std::string & mat_file, const std::string & var_name, const std::string & mode )`

Saves an Eigen dynamic matrix to a MATLAB .mat file, specialization for double matrices ([qpp::dmat](#))

Parameters

<i>A</i>	Eigen expression over the complex field
<i>mat_file</i>	MATALB .mat file
<i>var_name</i>	Variable name in the .mat file representing the matrix to be saved
<i>mode</i>	Saving mode (append, overwrite etc.), see MATLAB's <i>matOpen()</i> documentation for details

Here is the call graph for this function:



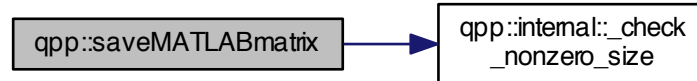
6.1.2.98 `template<> void qpp::saveMATLABmatrix ( const Eigen::MatrixBase< cmat > & A, const std::string & mat_file, const std::string & var_name, const std::string & mode )`

Saves an Eigen dynamic matrix to a MATLAB .mat file, specialization for complex matrices ([qpp::cmat](#))

## Parameters

<i>A</i>	Eigen expression over the complex field
<i>mat_file</i>	MATALB .mat file
<i>var_name</i>	Variable name in the .mat file representing the matrix to be saved
<i>mode</i>	Saving mode (append, overwrite etc.), see MATLAB's <i>matOpen()</i> documentation for details

Here is the call graph for this function:



6.1.2.99 `template<typename Derived > double qpp::schatten ( const Eigen::MatrixBase< Derived > & A, std::size_t p )`

Schatten norm.

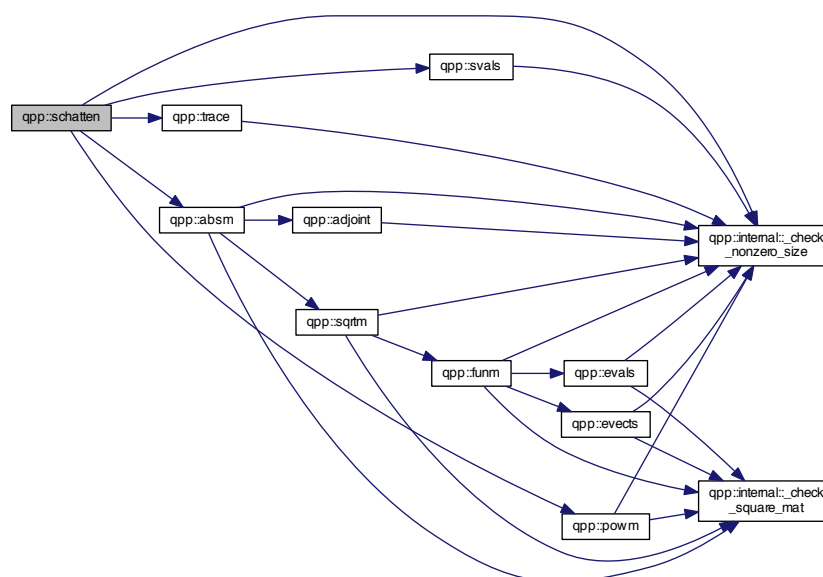
## Parameters

<i>A</i>	Eigen expression
<i>p</i>	Integer, greater or equal to 1

## Returns

Schatten-*p* norm of *A*, as a real number

Here is the call graph for this function:



6.1.2.100 `template<typename Derived > DynColVect<cplx> qpp::schmidtcoeff ( const Eigen::MatrixBase< Derived > & A, const std::vector< std::size_t > & dims )`

Schmidt coefficients of the bi-partite pure state  $A$ .

#### Note

The sum of the squares of the Schmidt coefficients equals 1

#### See also

[`qpp::schmidtprob\(\)`](#)

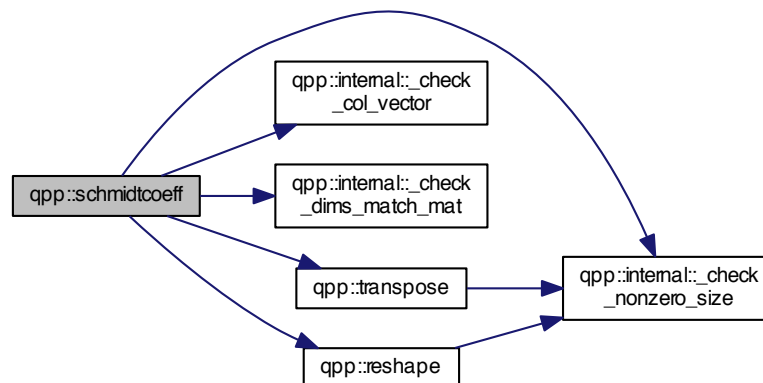
#### Parameters

$A$	Eigen expression
$dims$	Subsystems' dimensions

#### Returns

Schmidt coefficients of  $A$ , as a complex dynamic column vector

Here is the call graph for this function:



6.1.2.101 `template<typename Derived > DynColVect<double> qpp::schmidtprob ( const Eigen::MatrixBase< Derived > & A, const std::vector< std::size_t > & dims )`

Schmidt probabilities of the bi-partite pure state  $A$ .

Defined as the squares of the Schmidt coefficients

The sum of the Schmidt probabilities equals 1

#### See also

[`qpp::schmidtcoeff\(\)`](#)

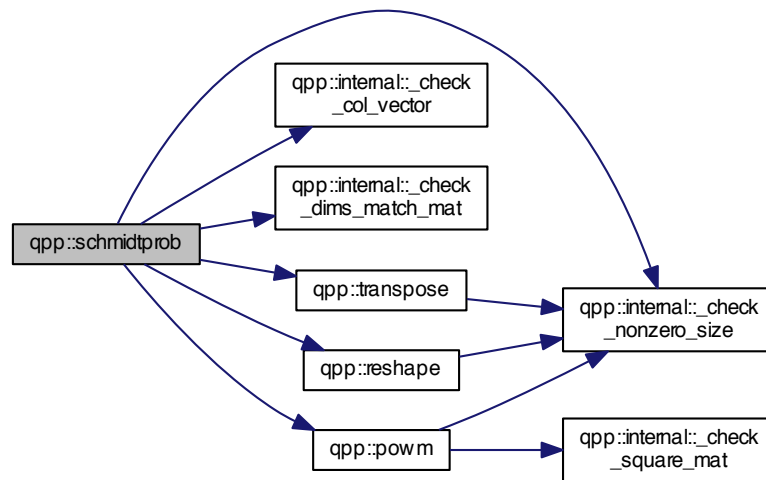
## Parameters

<i>A</i>	Eigen expression
<i>dims</i>	Subsystems' dimensions

## Returns

Schmidt probabilities of  $A$ , as a real dynamic column vector

Here is the call graph for this function:



**6.1.2.102** `template<typename Derived> cmat qpp::schmidtU ( const Eigen::MatrixBase< Derived> & A, const std::vector< std::size_t> & dims )`

Schmidt basis on Alice's side.

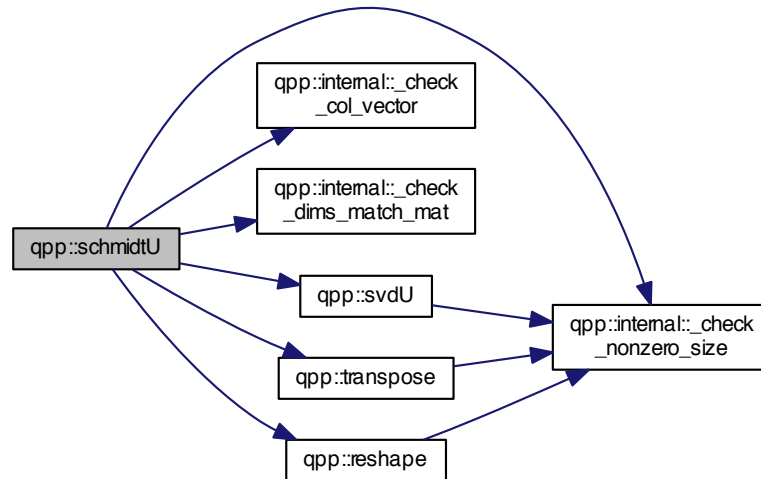
## Parameters

<i>A</i>	Eigen expression
<i>dims</i>	Subsystems' dimensions

## Returns

Unitary matrix  $U$  whose columns represent the Schmidt basis vectors on Alice's side.

Here is the call graph for this function:



6.1.2.103 `template<typename Derived > cmat qpp::schmidtV ( const Eigen::MatrixBase< Derived > & A, const std::vector< std::size_t > & dims )`

Schmidt basis on Bob's side.

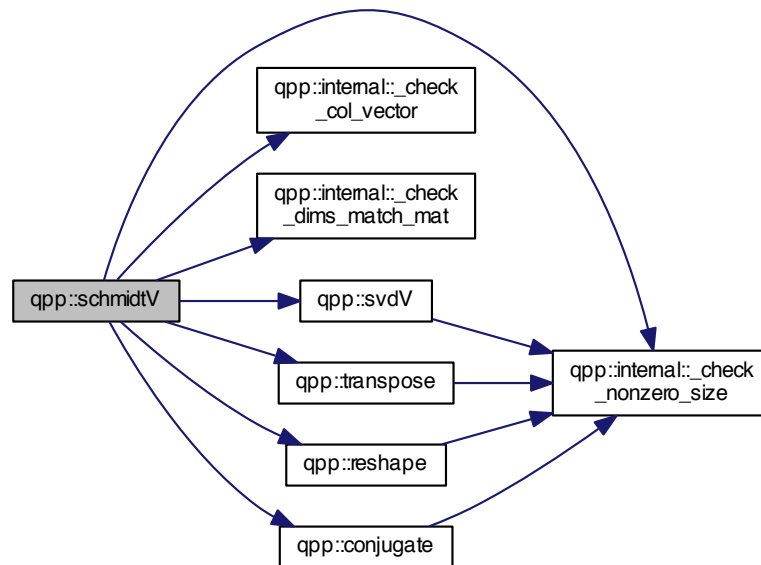
## Parameters

<i>A</i>	Eigen expression
<i>dims</i>	Subsystems' dimensions

**Returns**

Unitary matrix  $V$  whose columns represent the Schmidt basis vectors on Bob's side.

Here is the call graph for this function:



**6.1.2.104** `template<typename Derived> double qpp::shannon ( const Eigen::MatrixBase< Derived> & A )`

Shannon/von-Neumann entropy of the probability distribution/density matrix  $A$ .

**Parameters**

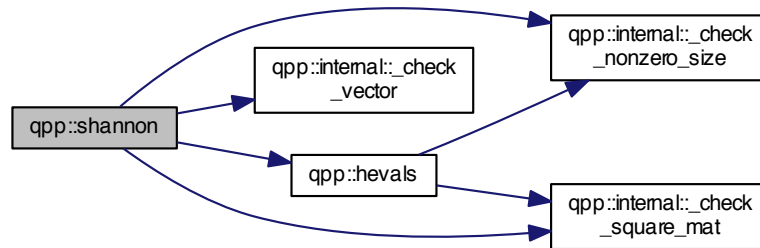
$A$	Eigen expression, representing a probability distribution (real dynamic column vector) or a density matrix (complex dynamic matrix)
-----	---



## Returns

Shannon/von-Neumann entropy, with the logarithm in base 2

Here is the call graph for this function:



6.1.2.105 `template<typename Derived> cmat qpp::sinm ( const Eigen::MatrixBase< Derived> & A )`

Matrix sin.

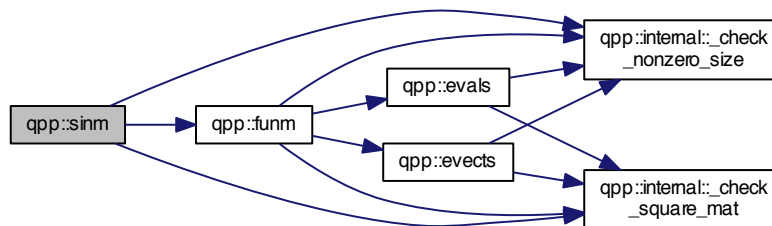
## Parameters

<i>A</i>	Eigen expression
----------	------------------

## Returns

Matrix sine of *A*

Here is the call graph for this function:



6.1.2.106 `template<typename Derived> cmat qpp::spectralpowm ( const Eigen::MatrixBase< Derived> & A, const cplx z )`

Matrix power.

Uses the spectral decomposition of *A* to compute the matrix power  
By convention  $A^0 = I$

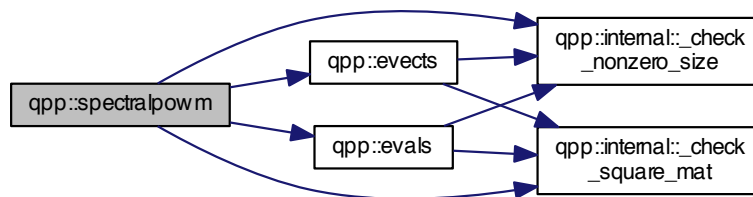
## Parameters

$A$	Eigen expression
$z$	Complex number

## Returns

Matrix power  $A^z$

Here is the call graph for this function:



6.1.2.107 `template<typename Derived> cmat qpp::sqrtm ( const Eigen::MatrixBase< Derived> & A )`

Matrix square root.

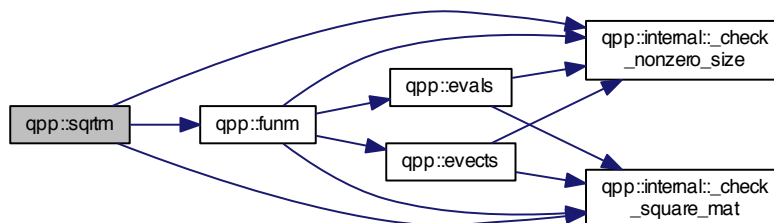
## Parameters

$A$	Eigen expression
-----	------------------

## Returns

Matrix square root of  $A$

Here is the call graph for this function:



6.1.2.108 `template<typename Derived> Derived::Scalar qpp::sum ( const Eigen::MatrixBase< Derived> & A )`

Element-wise sum of  $A$ .

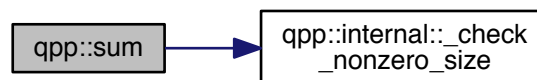
## Parameters

<i>A</i>	Eigen expression
----------	------------------

## Returns

Element-wise sum of *A*, as a scalar in the same scalar field as *A*

Here is the call graph for this function:



**6.1.2.109** `template<typename InputIterator > auto qpp::sum ( InputIterator first, InputIterator last ) -> typename InputIterator::value_type`

Element-wise sum of a range.

## Parameters

<i>first</i>	Iterator to the first element of the range
<i>last</i>	Iterator to the last element of the range

## Returns

Element-wise sum of the range, as a scalar in the same scalar field as the range

**6.1.2.110** `cmat qpp::super ( const std::vector< cmat > & Ks )`

Superoperator matrix representation.

Constructs the superoperator matrix of the channel specified by the set of Kraus operators *Ks* in the standard operator basis  $\{|i\rangle\langle j|\}$  ordered in lexicographical order, i.e.  $|0\rangle\langle 0|$ ,  $|0\rangle\langle 1|$  etc.

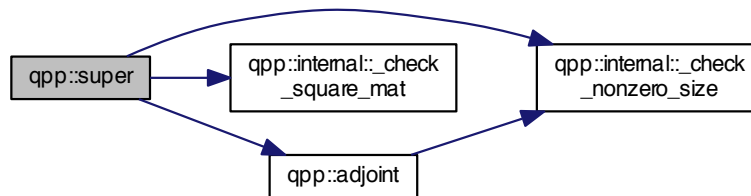
## Parameters

<i>Ks</i>	Set of Kraus operators
-----------	------------------------

**Returns**

Superoperator matrix representation

Here is the call graph for this function:



6.1.2.111 `template<typename Derived > DynColVect<double> qpp::svals ( const Eigen::MatrixBase< Derived > & A )`

Singular values.

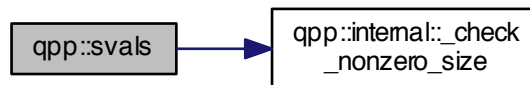
**Parameters**

<code>A</code>	Eigen expression
----------------	------------------

**Returns**

Singular values of  $A$ , as a real dynamic column vector

Here is the call graph for this function:



6.1.2.112 `template<typename Derived > cmat qpp::svdU ( const Eigen::MatrixBase< Derived > & A )`

Left singular vectors.

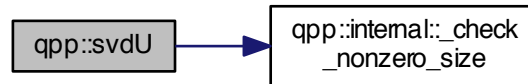
**Parameters**

<code>A</code>	Eigen expression
----------------	------------------

**Returns**

Complex dynamic matrix, whose columns are the left singular vectors of  $A$

Here is the call graph for this function:



**6.1.2.113** `template<typename Derived> cmat qpp::svdV ( const Eigen::MatrixBase< Derived> & A )`

Right singular vectors.

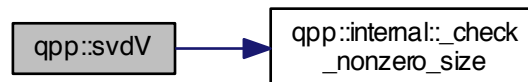
**Parameters**

$A$	Eigen expression
-----	------------------

**Returns**

Complex dynamic matrix, whose columns are the right singular vectors of  $A$

Here is the call graph for this function:



**6.1.2.114** `template<typename Derived> DynMat<typename Derived::Scalar> qpp::syspermute ( const Eigen::MatrixBase< Derived> & A, const std::vector< std::size_t> & perm, const std::vector< std::size_t> & dims )`

System permutation.

Permutes the subsystems in a state vector or density matrix

The qubit  $perm[i]$  is permuted to the location  $i$

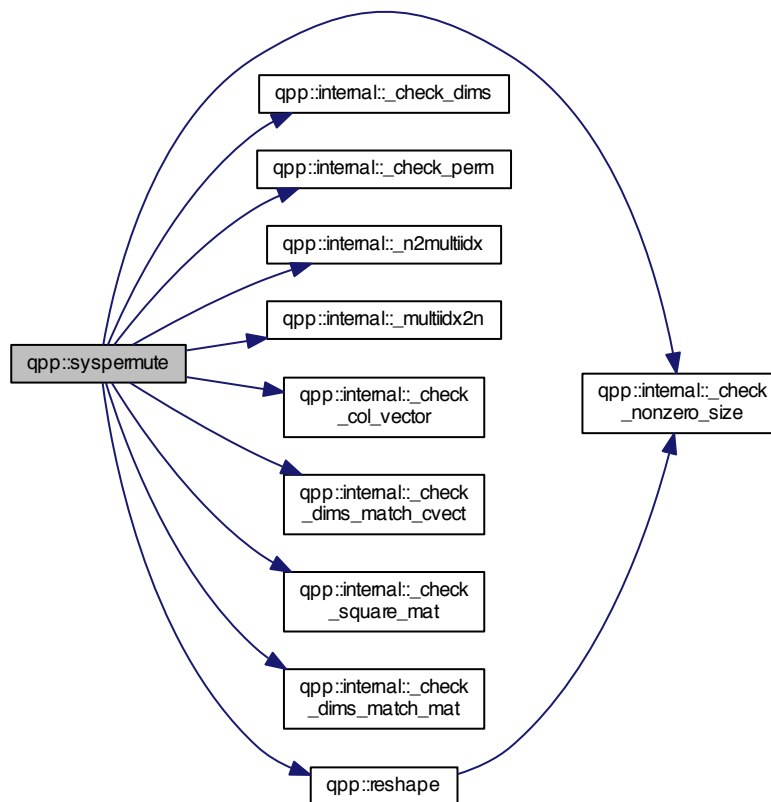
**Parameters**

<i>A</i>	Eigen expression
<i>perm</i>	Permutation
<i>dims</i>	Subsystems' dimensions

### Returns

Permuted system, as a dynamic matrix over the same scalar field as *A*

Here is the call graph for this function:



6.1.2.115 `template<typename Derived > Derived::Scalar qpp::trace ( const Eigen::MatrixBase< Derived > & A )`

Trace.

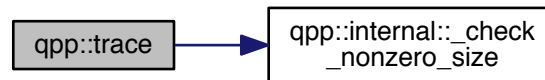
### Parameters

<i>A</i>	Eigen expression
----------	------------------

## Returns

Trace of  $A$ , as a scalar in the same scalar field as  $A$

Here is the call graph for this function:



6.1.2.116 `template<typename Derived > DynMat<typename Derived::Scalar> qpp::transpose ( const Eigen::MatrixBase<Derived > & A )`

Transpose.

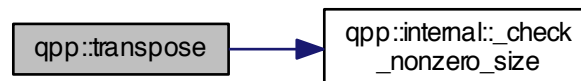
## Parameters

$A$	Eigen expression
-----	------------------

## Returns

Transpose of  $A$ , as a dynamic matrix over the same scalar field as  $A$

Here is the call graph for this function:



6.1.2.117 `template<typename Derived > double qpp::tsallis ( const Eigen::MatrixBase<Derived > & A, double alpha )`

Tsallis-  $\alpha$  entropy of the probability distribution/density matrix  $A$ , for  $\alpha \geq 0$

.

When  $\alpha \rightarrow 1$  the Tsallis entropy converges to the Shannon/von-Neumann entropy, with the logarithm in base  $e$

## Parameters

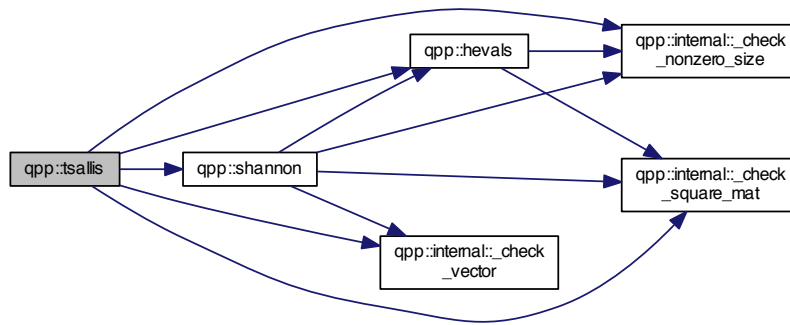
$A$	Eigen expression, representing a probability distribution (real dynamic column vector) or a density matrix (complex dynamic matrix)
-----	---

<i>alpha</i>	Non-negative real number
--------------	--------------------------

### Returns

Renyi-  $\alpha$  entropy, with the logarithm in base 2

Here is the call graph for this function:



## 6.1.3 Variable Documentation

### 6.1.3.1 constexpr double qpp::chop = 1e-10

Used in [qpp::disp\(\)](#) and [qpp::displn\(\)](#) for setting to zero numbers that have their absolute value smaller than [qpp::ct::chop](#).

### 6.1.3.2 const Codes& qpp::codes = Codes::get\_instance()

[qpp::Codes](#) const Singleton

Initializes the codes, see the class [qpp::Codes](#)

### 6.1.3.3 constexpr double qpp::ee = 2.718281828459045235360287471352662497

Base of natural logarithm,  $e$ .

### 6.1.3.4 constexpr double qpp::eps = 1e-12

Used to decide whether a number or expression in double precision is zero or not.

Example:

```
if(std::abs(x) < qpp::eps) // x is zero
```

### 6.1.3.5 const Gates& qpp::gt = Gates::get\_instance()

[qpp::Gates](#) const Singleton

Initializes the gates, see the class [qpp::Gates](#)



## 6.1.3.6 constexpr std::size\_t qpp::infy = -1

Used to denote infinity.

## 6.1.3.7 const Init&amp; qpp::init = Init::get\_instance()

[qpp::Init](#) const Singleton

Additional initializations/cleanups

## 6.1.3.8 constexpr std::size\_t qpp::maxn = 64

Maximum number of qubits.

Used internally to allocate arrays on the stack (for speed reasons)

## 6.1.3.9 constexpr double qpp::pi = 3.141592653589793238462643383279502884

$\pi$

## 6.1.3.10 RandomDevices&amp; qpp::rdevs = RandomDevices::get\_instance()

[qpp::RandomDevices](#) Singleton

Initializes the random devices, see the class [qpp::RandomDevices](#)

## 6.1.3.11 const States&amp; qpp::st = States::get\_instance()

[qpp::States](#) const Singleton

Initializes the states, see the class [qpp::States](#)

## 6.2 qpp::experimental Namespace Reference

### Classes

- class [Qudit](#)

### Functions

- template<typename Derived1 , typename Derived2 >  
[DynMat](#)< typename Derived1::Scalar > [apply](#) (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< std::size\_t > &subsys, const std::vector< std::size\_t > &dims)  
*Applies the gate A to the part subsys of a multipartite state vector or density matrix.*
- template<typename Derived >  
[cmat channel](#) (const Eigen::MatrixBase< Derived > &rho, const std::vector< [cmat](#) > &Ks, const std::vector< std::size\_t > &subsys, const std::vector< std::size\_t > &dims)  
*Applies the channel specified by the set of Kraus operators Ks to the part of the density matrix rho specified by subsys.*
- [cmat super](#) (const std::vector< [cmat](#) > &Ks)  
*Superoperator matrix representation.*

- `template<typename Derived >`  
`DynMat< typename Derived::Scalar > CTRL (const Eigen::MatrixBase< Derived > &A, const std::vector<`  
`std::size_t > &ctrl, const std::vector< std::size_t > &subsys, std::size_t n, std::size_t d=2)`

*Generates the multipartite multiple-controlled-A gate in matrix form.*

- `cmat choi (const std::vector< cmat > &Ks)`

*Choi matrix representation.*

- `std::vector< cmat > randkraus (std::size_t n, std::size_t D)`

*Generates a set of random Kraus operators.*

- `template<typename Derived >`  
`double renyi_inf (const Eigen::MatrixBase< Derived > &A)`

*Renyi- $\infty$  entropy (min entropy) of the probability distribution/density matrix A.*

## 6.2.1 Detailed Description

Experimental/test functions, do not use/modify these functions/classes

## 6.2.2 Function Documentation

- 6.2.2.1 `template<typename Derived1 , typename Derived2 > DynMat<typename Derived1::Scalar> qpp::experimental::apply`  
`( const Eigen::MatrixBase< Derived1 > & state, const Eigen::MatrixBase< Derived2 > & A, const std::vector<`  
`std::size_t > & subsys, const std::vector< std::size_t > & dims )`

Applies the gate *A* to the part *subsys* of a multipartite state vector or density matrix.

### Note

The dimension of the gate *A* must match the dimension of *subsys*

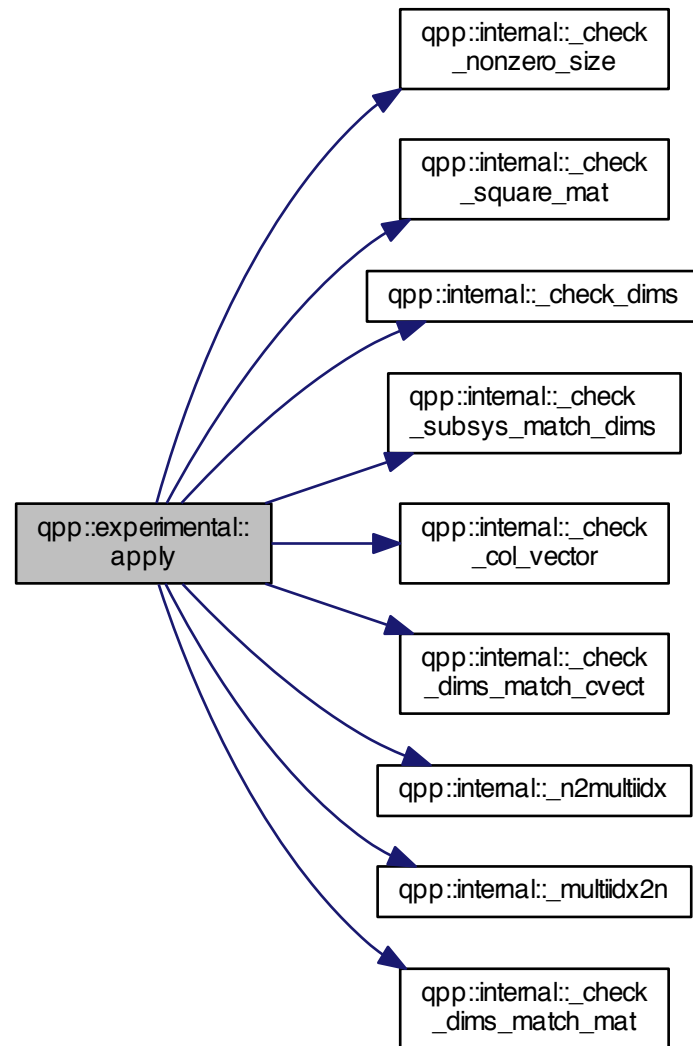
### Parameters

<i>state</i>	Eigen expression
<i>A</i>	Eigen expression
<i>subsys</i>	Subsystem indexes where the gate <i>A</i> is applied
<i>dims</i>	Local dimensions of all local Hilbert spaces (can be different)

## Returns

Gate  $A$  applied to the part *subsys* of *state*

Here is the call graph for this function:



**6.2.2.2** `template<typename Derived> cmat qpp::experimental::channel ( const Eigen::MatrixBase< Derived> & rho, const std::vector< cmat> & Ks, const std::vector< std::size_t> & subsys, const std::vector< std::size_t> & dims )`

Applies the channel specified by the set of Kraus operators *Ks* to the part of the density matrix *rho* specified by *subsys*.

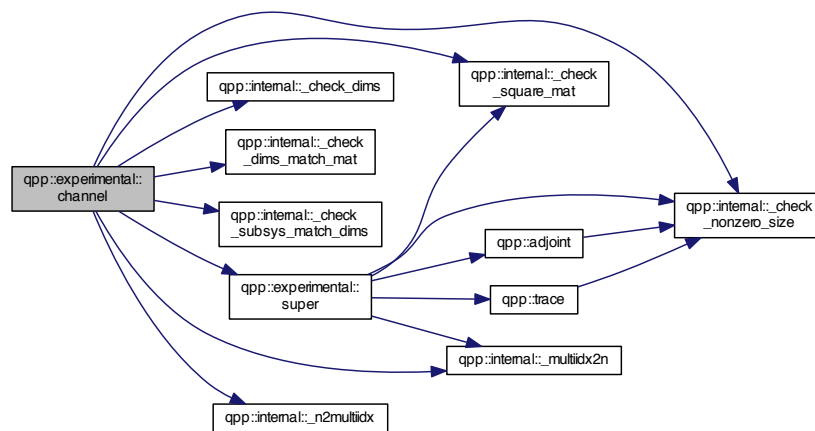
## Parameters

<i>rho</i>	Eigen expression
<i>Ks</i>	Set of Kraus operators
<i>subsys</i>	Subsystems' indexes
<i>dims</i>	Local dimensions of all local Hilbert spaces (can be different)

## Returns

Output density matrix after the action of the channel

Here is the call graph for this function:



### 6.2.2.3 cmat qpp::experimental::choi ( const std::vector< cmat > & Ks )

Choi matrix representation.

Constructs the Choi matrix of the channel specified by the set of Kraus operators  $Ks$  in the standard operator basis  $\{|i\rangle\langle j|\}$  ordered in lexicographical order, i.e.  $|0\rangle\langle 0|$ ,  $|0\rangle\langle 1|$  etc.

## Note

The superoperator matrix  $S$  and the Choi matrix  $C$  are related by  $S_{ab,mn} = C_{ma,nb}$

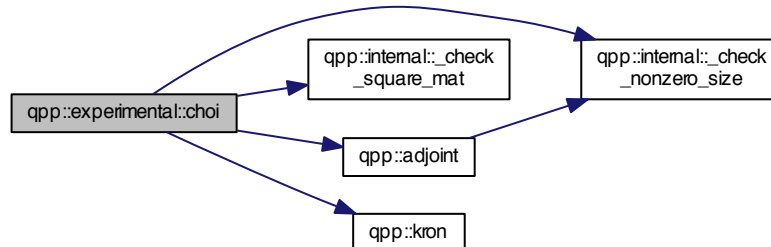
## Parameters

<i>Ks</i>	Set of Kraus operators
-----------	------------------------

## Returns

Choi matrix representation

Here is the call graph for this function:



6.2.2.4 `template<typename Derived > DynMat<typename Derived::Scalar> qpp::experimental::CTRL ( const Eigen::MatrixBase< Derived > & A, const std::vector< std::size_t > & ctrl, const std::vector< std::size_t > & subsys, std::size_t n, std::size_t d = 2 )`

Generates the multipartite multiple-controlled-A gate in matrix form.

## Note

The dimension of the gate *A* must match the dimension of *subsys*

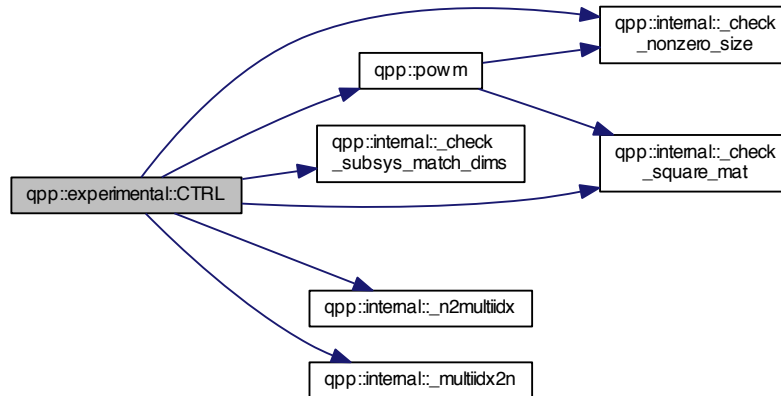
## Parameters

<i>A</i>	Eigen expression
<i>ctrl</i>	Control subsystem indexes
<i>subsys</i>	Subsystem indexes where the gate <i>A</i> is applied
<i>n</i>	Total number of subsystems
<i>d</i>	Local dimensions of all local Hilbert spaces (must all be equal)

**Returns**

CTRL-A gate, as a matrix over the same scalar field as  $A$

Here is the call graph for this function:



#### 6.2.2.5 `std::vector<cmat> qpp::experimental::randkraus ( std::size_t n, std::size_t D )`

Generates a set of random Kraus operators.

**Note**

The set of Kraus operators satisfy the closure condition  $\sum_i K_i^\dagger K_i = I$

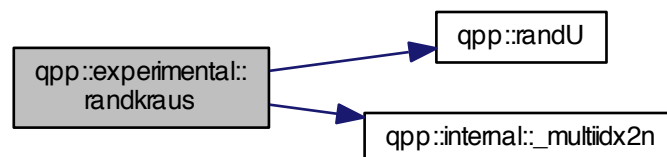
**Parameters**

$n$	Number of Kraus operators
$D$	Dimension of the Hilbert space

**Returns**

Set of  $n$  Kraus operators satisfying the closure condition

Here is the call graph for this function:



6.2.2.6 `template<typename Derived > double qpp::experimental::renyi_inf ( const Eigen::MatrixBase< Derived > & A )`

Renyi-  $\infty$  entropy (min entropy) of the probability distribution/density matrix  $A$ .

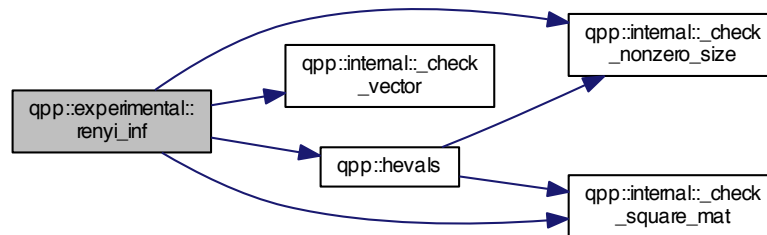
## Parameters

A	Eigen expression, representing a probability distribution (real dynamic column vector) or a density matrix (complex dynamic matrix)
---	---

## Returns

Renyi-  $\infty$  entropy (min entropy), with the logarithm in base 2

Here is the call graph for this function:



### 6.2.2.7 cmat qpp::experimental::super ( const std::vector< cmat > & Ks )

Superoperator matrix representation.

Constructs the superoperator matrix of the channel specified by the set of Kraus operators  $K_s$  in the standard operator basis  $\{|i\rangle\langle j|\}$  ordered in lexicographical order, i.e.  $|0\rangle\langle 0|$ ,  $|0\rangle\langle 1|$  etc.

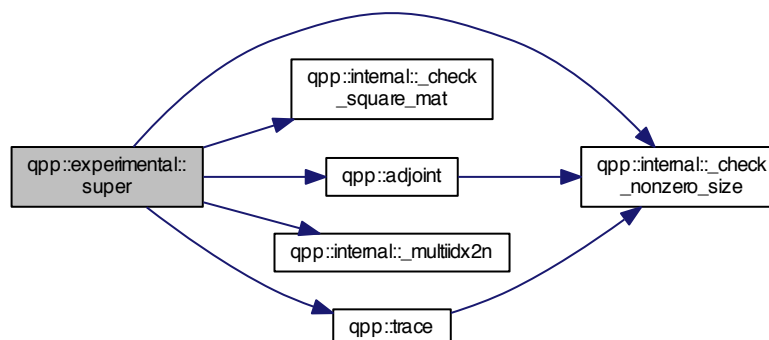
## Parameters

$K_s$	Set of Kraus operators
-------	------------------------

## Returns

Superoperator matrix representation

Here is the call graph for this function:





## 6.3 qpp::internal Namespace Reference

### Classes

- class [Singleton](#)  
*Singleton policy class, used internally to implement the singleton pattern via CRTP (Curiously recurring template pattern)*

### Functions

- void [\\_n2multiidx](#) (std::size\_t n, std::size\_t numdims, const std::size\_t \*dims, std::size\_t \*result)
- std::size\_t [\\_multiidx2n](#) (const std::size\_t \*midx, std::size\_t numdims, const std::size\_t \*dims)
- template<typename Derived >  
 bool [\\_check\\_square\\_mat](#) (const Eigen::MatrixBase< Derived > &A)
- template<typename Derived >  
 bool [\\_check\\_vector](#) (const Eigen::MatrixBase< Derived > &A)
- template<typename Derived >  
 bool [\\_check\\_row\\_vector](#) (const Eigen::MatrixBase< Derived > &A)
- template<typename Derived >  
 bool [\\_check\\_col\\_vector](#) (const Eigen::MatrixBase< Derived > &A)
- template<typename T >  
 bool [\\_check\\_nonzero\\_size](#) (const T &x)
- bool [\\_check\\_dims](#) (const std::vector< std::size\_t > &dims)
- template<typename Derived >  
 bool [\\_check\\_dims\\_match\\_mat](#) (const std::vector< std::size\_t > &dims, const Eigen::MatrixBase< Derived > &A)
- template<typename Derived >  
 bool [\\_check\\_dims\\_match\\_cvect](#) (const std::vector< std::size\_t > &dims, const Eigen::MatrixBase< Derived > &V)
- template<typename Derived >  
 bool [\\_check\\_dims\\_match\\_rvect](#) (const std::vector< std::size\_t > &dims, const Eigen::MatrixBase< Derived > &V)
- bool [\\_check\\_eq\\_dims](#) (const std::vector< std::size\_t > &dims, std::size\_t dim)
- bool [\\_check\\_subsys\\_match\\_dims](#) (const std::vector< std::size\_t > &subsys, const std::vector< std::size\_t > &dims)
- bool [\\_check\\_perm](#) (const std::vector< std::size\_t > &perm)
- template<typename Derived1 , typename Derived2 >  
[DynMat](#)< typename Derived1::Scalar > [\\_kron2](#) (const Eigen::MatrixBase< Derived1 > &A, const Eigen::MatrixBase< Derived2 > &B)
- template<typename T >  
 void [variadic\\_vector\\_emplace](#) (std::vector< T > &)
- template<typename T , typename First , typename... Args>  
 void [variadic\\_vector\\_emplace](#) (std::vector< T > &v, First &&first, Args &&...args)

#### 6.3.1 Detailed Description

Internal implementation details, do not use/modify these functions/classes

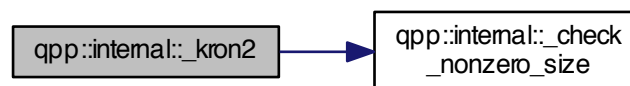
#### 6.3.2 Function Documentation

6.3.2.1 template<typename Derived > bool qpp::internal::\_check\_col\_vector ( const Eigen::MatrixBase< Derived > & A )

6.3.2.2 bool qpp::internal::\_check\_dims ( const std::vector< std::size\_t > & dims )

- 6.3.2.3 `template<typename Derived > bool qpp::internal::_check_dims_match_cvect ( const std::vector< std::size_t > & dims, const Eigen::MatrixBase< Derived > & V )`
- 6.3.2.4 `template<typename Derived > bool qpp::internal::_check_dims_match_mat ( const std::vector< std::size_t > & dims, const Eigen::MatrixBase< Derived > & A )`
- 6.3.2.5 `template<typename Derived > bool qpp::internal::_check_dims_match_rvect ( const std::vector< std::size_t > & dims, const Eigen::MatrixBase< Derived > & V )`
- 6.3.2.6 `bool qpp::internal::_check_eq_dims ( const std::vector< std::size_t > & dims, std::size_t dim )`
- 6.3.2.7 `template<typename T > bool qpp::internal::_check_nonzero_size ( const T & x )`
- 6.3.2.8 `bool qpp::internal::_check_perm ( const std::vector< std::size_t > & perm )`
- 6.3.2.9 `template<typename Derived > bool qpp::internal::_check_row_vector ( const Eigen::MatrixBase< Derived > & A )`
- 6.3.2.10 `template<typename Derived > bool qpp::internal::_check_square_mat ( const Eigen::MatrixBase< Derived > & A )`
- 6.3.2.11 `bool qpp::internal::_check_subsys_match_dims ( const std::vector< std::size_t > & subsys, const std::vector< std::size_t > & dims )`
- 6.3.2.12 `template<typename Derived > bool qpp::internal::_check_vector ( const Eigen::MatrixBase< Derived > & A )`
- 6.3.2.13 `template<typename Derived1, typename Derived2 > DynMat<typename Derived1::Scalar> qpp::internal::_kron2 ( const Eigen::MatrixBase< Derived1 > & A, const Eigen::MatrixBase< Derived2 > & B )`

Here is the call graph for this function:



- 6.3.2.14 `std::size_t qpp::internal::_multiidx2n ( const std::size_t * midx, std::size_t numdims, const std::size_t * dims )`  
[inline]
- 6.3.2.15 `void qpp::internal::_n2multiidx ( std::size_t n, std::size_t numdims, const std::size_t * dims, std::size_t * result )`  
[inline]
- 6.3.2.16 `template<typename T > void qpp::internal::variadic_vector_emplace ( std::vector< T > & )`

6.3.2.17 `template<typename T , typename First , typename... Args> void qpp::internal::variadic_vector_emplace (`  
`std::vector< T > & v, First && first, Args &&... args )`

Here is the call graph for this function:





## Chapter 7

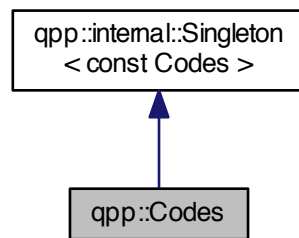
# Class Documentation

### 7.1 qpp::Codes Class Reference

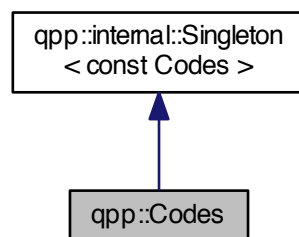
const Singleton class that defines quantum error correcting codes

```
#include <codes.h>
```

Inheritance diagram for qpp::Codes:



Collaboration diagram for qpp::Codes:



## Public Types

- enum `Type` { `Type::FIVE_QUBIT` = 1, `Type::SEVEN_QUBIT_STEANE`, `Type::NINE_QUBIT_SHOR` }  
Code types, add more codes here if needed.

## Public Member Functions

- `ket codeword` (`Type` type, `std::size_t` i) const  
Returns the codeword of the specified code.

## Private Member Functions

- `Codes` ()=default  
Default constructor.

## Friends

- class `internal::Singleton< const Codes >`

## Additional Inherited Members

### 7.1.1 Detailed Description

const Singleton class that defines quantum error correcting codes

### 7.1.2 Member Enumeration Documentation

#### 7.1.2.1 enum `qpp::Codes::Type` [strong]

Code types, add more codes here if needed.

See also

[`qpp::Codes::codeword\(\)`](#)

Enumerator

**`FIVE_QUBIT`** [[5,1,3]] qubit code  
**`SEVEN_QUBIT_STEANE`** [[7,1,3]] Steane qubit code  
**`NINE_QUBIT_SHOR`** [[9,1,3]] Shor qubit code

### 7.1.3 Constructor & Destructor Documentation

#### 7.1.3.1 `qpp::Codes::Codes ( )` [private],[default]

Default constructor.

### 7.1.4 Member Function Documentation

#### 7.1.4.1 `ket qpp::Codes::codeword ( Type type, std::size_t i ) const` [inline]

Returns the codeword of the specified code.

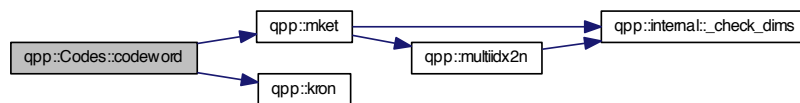
## Parameters

<i>type</i>	Code type, defined in the enum <code>qpp::Codes::Types</code>
<i>i</i>	Codeword index

## Returns

*i*-th codeword of the code *type*

Here is the call graph for this function:



## 7.1.5 Friends And Related Function Documentation

7.1.5.1 friend class `internal::Singleton< const Codes >` [friend]

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

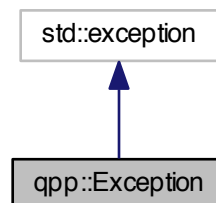
- `include/classes/codes.h`

## 7.2 qpp::Exception Class Reference

Generates custom exceptions, used when validating function parameters.

```
#include <exception.h>
```

Inheritance diagram for `qpp::Exception`:



Collaboration diagram for `qpp::Exception`:



## Public Types

- enum `Type` {  
`Type::UNKNOWN_EXCEPTION = 1`, `Type::ZERO_SIZE`, `Type::MATRIX_NOT_SQUARE`, `Type::MATRIX_NOT_CVECTOR`,  
`Type::MATRIX_NOT_RVECTOR`, `Type::MATRIX_NOT_VECTOR`, `Type::MATRIX_NOT_SQUARE_OR_CVECTOR`, `Type::MATRIX_NOT_SQUARE_OR_RVECTOR`,  
`Type::MATRIX_NOT_SQUARE_OR_VECTOR`, `Type::MATRIX_MISMATCH_SUBSYS`, `Type::DIMS_INVALID`, `Type::DIMS_NOT_EQUAL`,  
`Type::DIMS_MISMATCH_MATRIX`, `Type::DIMS_MISMATCH_CVECTOR`, `Type::DIMS_MISMATCH_RVECTOR`, `Type::DIMS_MISMATCH_VECTOR`,  
`Type::SUBSYS_MISMATCH_DIMS`, `Type::PERM_INVALID`, `Type::NOT_QUBIT_GATE`, `Type::NOT_QUBIT_SUBSYS`,  
`Type::NOT_BIPARTITE`, `Type::OUT_OF_RANGE`, `Type::TYPE_MISMATCH`, `Type::UNDEFINED_TYPE`,  
`Type::NO_CODEWORD`, `Type::CUSTOM_EXCEPTION` }  
*Exception types, add more exceptions here if needed.*

## Public Member Functions

- `Exception` (const `std::string` &where, const `Type` &type)  
*Constructs an exception.*
- `Exception` (const `std::string` &where, const `std::string` &custom)  
*Constructs an exception.*
- virtual const char \* `what` () const noexcept override  
*Overrides `std::exception::what()`*

## Private Member Functions

- `std::string` `_construct_exception_msg` ()  
*Constructs the exception's description from its type.*



## Private Attributes

- `std::string _where`
- `std::string _msg`
- `Type _type`
- `std::string _custom`

### 7.2.1 Detailed Description

Generates custom exceptions, used when validating function parameters.

Customize this class if more exceptions are needed

### 7.2.2 Member Enumeration Documentation

#### 7.2.2.1 `enum qpp::Exception::Type` [strong]

[Exception](#) types, add more exceptions here if needed.

See also

`qpp::Exception::_construct_exception_msg()`

#### Enumerator

**UNKNOWN\_EXCEPTION** UNKNOWN\_EXCEPTION. Unknown exception

**ZERO\_SIZE** ZERO\_SIZE. Zero sized object, e.g. empty `Eigen::Matrix` or `std::vector` with no elements

**MATRIX\_NOT\_SQUARE** MATRIX\_NOT\_SQUARE. `Eigen::Matrix` is not square

**MATRIX\_NOT\_CVECTOR** MATRIX\_NOT\_CVECTOR. `Eigen::Matrix` is not a column vector

**MATRIX\_NOT\_RVECTOR** MATRIX\_NOT\_RVECTOR. `Eigen::Matrix` is not a row vector

**MATRIX\_NOT\_VECTOR** MATRIX\_NOT\_VECTOR. `Eigen::Matrix` is not a row/column vector

**MATRIX\_NOT\_SQUARE\_OR\_CVECTOR** MATRIX\_NOT\_SQUARE\_OR\_CVECTOR. `Eigen::Matrix` is not square nor a column vector

**MATRIX\_NOT\_SQUARE\_OR\_RVECTOR** MATRIX\_NOT\_SQUARE\_OR\_RVECTOR. `Eigen::Matrix` is not square nor a row vector

**MATRIX\_NOT\_SQUARE\_OR\_VECTOR** MATRIX\_NOT\_SQUARE\_OR\_VECTOR. `Eigen::Matrix` is not square nor a row/column vector

**MATRIX\_MISMATCH\_SUBSYS** SUBSYS\_MISMATCH\_MATRIX.

**DIMS\_INVALID** DIMS\_INVALID. Matrix size mismatch subsystems' size (e.g. in [apply\(\)](#), or [channel\(\)](#) `std::vector<std::size_t>` representing the dimensions has zero size or contains zeros

**DIMS\_NOT\_EQUAL** DIMS\_NOT\_EQUAL. `std::vector<std::size_t>` representing the dimensions contains non-equal elements

**DIMS\_MISMATCH\_MATRIX** DIMS\_MISMATCH\_MATRIX. Product of the dimensions' `std::vector<std::size_t>` is not equal to the number of rows of `Eigen::Matrix` (assumed to be square)

**DIMS\_MISMATCH\_CVECTOR** DIMS\_MISMATCH\_CVECTOR. Product of the dimensions' `std::vector<std::size_t>` is not equal to the number of cols of `Eigen::Matrix` (assumed to be a column vector)

**DIMS\_MISMATCH\_RVECTOR** DIMS\_MISMATCH\_RVECTOR. Product of the dimensions' `std::vector<std::size_t>` is not equal to the number of cols of `Eigen::Matrix` (assumed to be a row vector)

**DIMS\_MISMATCH\_VECTOR** DIMS\_MISMATCH\_VECTOR. Product of the dimensions' `std::vector<std::size_t>` is not equal to the number of cols of `Eigen::Matrix` (assumed to be a row/column vector)

**SUBSYS\_MISMATCH\_DIMS** SUBSYS\_MISMATCH\_DIMS. `std::vector<std::size_t>` representing the subsystems' labels has duplicates, or has entries that are larger than the size of the `std::vector<std::size_t>` representing the dimensions

**PERM\_INVALID** PERM\_INVALID. Invalid `std::vector<std::size_t>` permutation

**NOT\_QUBIT\_GATE** NOT\_QUBIT\_GATE. `Eigen::Matrix` is not 2 x 2

**NOT\_QUBIT\_SUBSYS** NOT\_QUBIT\_SUBSYS. Subsystems are not 2-dimensional

**NOT\_BIPARTITE** NOT\_BIPARTITE. `std::vector<std::size_t>` representing the dimensions has size different from 2

**OUT\_OF\_RANGE** OUT\_OF\_RANGE. Parameter out of range

**TYPE\_MISMATCH** TYPE\_MISMATCH. Types do not match (i.e. `Matrix<double>` vs `Matrix<cplx>`)

**UNDEFINED\_TYPE** UNDEFINED\_TYPE. Templated function not defined for this type

**NO\_CODEWORD**

**CUSTOM\_EXCEPTION** < NO\_CODEWORD Codeword does not exist, thrown when calling [qpp::Codes::codeword\(\)](#) with invalid *i* CUSTOM\_EXCEPTION Custom exception, user must provide a custom message

## 7.2.3 Constructor & Destructor Documentation

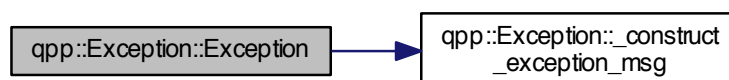
### 7.2.3.1 `qpp::Exception::Exception ( const std::string & where, const Type & type )` `[inline]`

Constructs an exception.

Parameters

<i>where</i>	Text representing where the exception occurred
<i>type</i>	<a href="#">Exception</a> 's type, see the strong enumeration <a href="#">qpp::Exception::Type</a>

Here is the call graph for this function:



### 7.2.3.2 `qpp::Exception::Exception ( const std::string & where, const std::string & custom )` `[inline]`

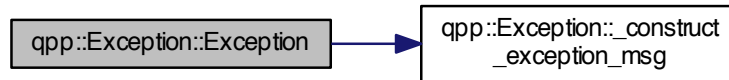
Constructs an exception.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

<i>where</i>	Text representing where the exception occurred
<i>custom</i>	<a href="#">Exception</a> 's description

Here is the call graph for this function:



## 7.2.4 Member Function Documentation

**7.2.4.1** `std::string qpp::Exception::_construct_exception_msg ( )` `[inline]`, `[private]`

Constructs the exception's description from its type.

Must modify the code of this function if more exceptions are added

Returns

[Exception](#)'s description

**7.2.4.2** `virtual const char* qpp::Exception::what ( ) const` `[inline]`, `[override]`, `[virtual]`, `[noexcept]`

Overrides `std::exception::what()`

Returns

[Exception](#)'s description

## 7.2.5 Member Data Documentation

**7.2.5.1** `std::string qpp::Exception::_custom` `[private]`

**7.2.5.2** `std::string qpp::Exception::_msg` `[private]`

**7.2.5.3** `Type qpp::Exception::_type` `[private]`

**7.2.5.4** `std::string qpp::Exception::_where` `[private]`

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

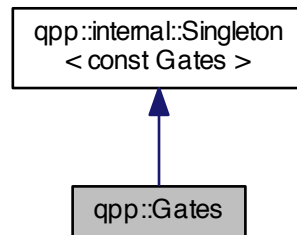
- `include/classes/exception.h`

## 7.3 qpp::Gates Class Reference

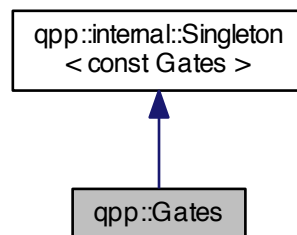
const Singleton class that implements most commonly used gates

```
#include <gates.h>
```

Inheritance diagram for qpp::Gates:



Collaboration diagram for qpp::Gates:



## Public Member Functions

- **cmat Rn** (double theta, std::vector< double > n) const  
*Rotation of theta about the 3-dimensional real unit vector n.*
- **cmat Zd** (std::size\_t D) const  
*Generalized Z gate for qudits.*
- **cmat Fd** (std::size\_t D) const  
*Fourier transform gate for qudits.*
- **cmat Xd** (std::size\_t D) const  
*Generalized X gate for qudits.*
- template<typename Derived = Eigen::MatrixXcd>  
Derived **Id** (std::size\_t D) const  
*Identity gate.*
- template<typename Derived >  
**DynMat**< typename Derived::Scalar > **CTRL** (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size\_t > &ctrl, const std::vector< std::size\_t > &subsys, std::size\_t n, std::size\_t d=2) const  
*Generates the multipartite multiple-controlled-A gate in matrix form.*
- template<typename Derived >  
**DynMat**< typename Derived::Scalar > **expandout** (const Eigen::MatrixBase< Derived > &A, std::size\_t pos, const std::vector< std::size\_t > &dims) const

*Expands out.*

## Public Attributes

- [cmat Id2](#) { cmat::Identity(2, 2) }  
*Identity gate.*
- [cmat H](#) { cmat::Zero(2, 2) }  
*Hadamard gate.*
- [cmat X](#) { cmat::Zero(2, 2) }  
*Pauli Sigma-X gate.*
- [cmat Y](#) { cmat::Zero(2, 2) }  
*Pauli Sigma-Y gate.*
- [cmat Z](#) { cmat::Zero(2, 2) }  
*Pauli Sigma-Z gate.*
- [cmat S](#) { cmat::Zero(2, 2) }  
*S gate.*
- [cmat T](#) { cmat::Zero(2, 2) }  
*T gate.*
- [cmat CNOTab](#) { cmat::Identity(4, 4) }  
*Controlled-NOT control target gate.*
- [cmat CZ](#) { cmat::Identity(4, 4) }  
*Controlled-Phase gate.*
- [cmat CNOTba](#) { cmat::Zero(4, 4) }  
*Controlled-NOT target control gate.*
- [cmat SWAP](#) { cmat::Identity(4, 4) }  
*SWAP gate.*
- [cmat TOF](#) { cmat::Identity(8, 8) }  
*Toffoli gate.*
- [cmat FRED](#) { cmat::Identity(8, 8) }  
*Fredkin gate.*

## Private Member Functions

- [Gates](#) ()  
*Initializes the gates.*

## Friends

- class [internal::Singleton](#)< const [Gates](#) >

## Additional Inherited Members

### 7.3.1 Detailed Description

const Singleton class that implements most commonly used gates

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 qpp::Gates::Gates ( ) [inline], [private]

Initializes the gates.

### 7.3.3 Member Function Documentation

**7.3.3.1** `template<typename Derived > DynMat<typename Derived::Scalar> qpp::Gates::CTRL ( const Eigen::MatrixBase<Derived> & A, const std::vector< std::size_t > & ctrl, const std::vector< std::size_t > & subsys, std::size_t n, std::size_t d = 2 ) const [inline]`

Generates the multipartite multiple-controlled- $A$  gate in matrix form.

#### Note

The dimension of the gate  $A$  must match the dimension of  $subsys$

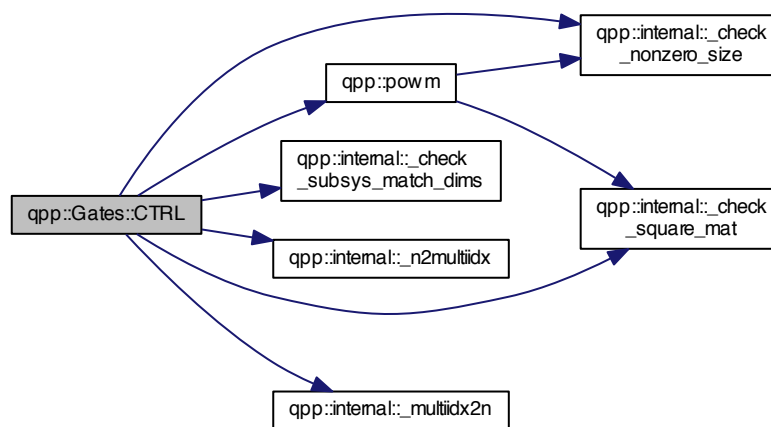
#### Parameters

$A$	Eigen expression
$ctrl$	Control subsystem indexes
$subsys$	Subsystem indexes where the gate $A$ is applied
$n$	Total number of subsystems
$d$	Local dimensions of all local Hilbert spaces (must all be equal)

#### Returns

CTRL- $A$  gate, as a matrix over the same scalar field as  $A$

Here is the call graph for this function:



**7.3.3.2** `template<typename Derived > DynMat<typename Derived::Scalar> qpp::Gates::expandout ( const Eigen::MatrixBase<Derived> & A, std::size_t pos, const std::vector< std::size_t > & dims ) const [inline]`

Expands out.

Expands out  $A$  as a matrix in a multi-partite system

Faster than using `qpp::kron(I, I, ..., I, A, I, ..., I)`

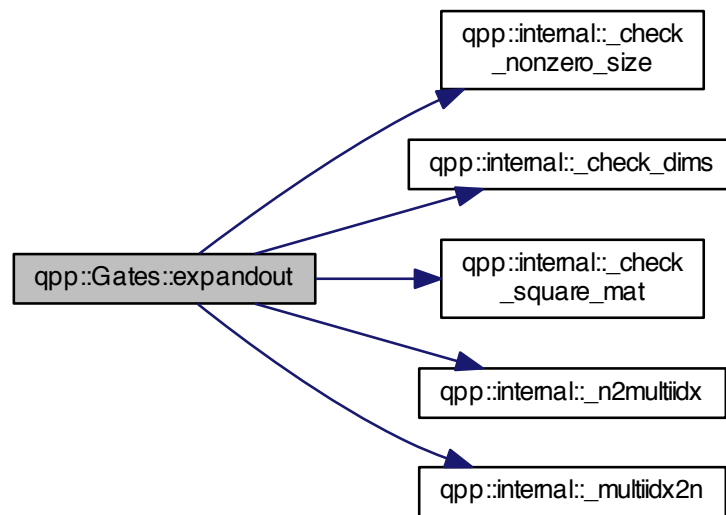
## Parameters

<i>A</i>	Eigen expression
<i>pos</i>	Position
<i>dims</i>	Dimensions of the multi-partite system

## Returns

Tensor product  $I \otimes \cdots \otimes I \otimes A \otimes I \otimes \cdots \otimes I$ , with  $A$  on position  $pos$ , as a dynamic matrix over the same scalar field as  $A$

Here is the call graph for this function:



### 7.3.3.3 `cmat qpp::Gates::Fd ( std::size_t D ) const [inline]`

Fourier transform gate for qudits.

## Note

Defined as  $F = \sum_{jk} \exp(2\pi i jk/D) |j\rangle\langle k|$

## Parameters

<i>D</i>	Dimension of the Hilbert space
----------	--------------------------------

## Returns

Fourier transform gate for qudits

Here is the call graph for this function:



**7.3.3.4** `template<typename Derived = Eigen::MatrixXcd> Derived qpp::Gates::Id ( std::size_t D ) const` `[inline]`

Identity gate.

## Note

Can change the return type from complex matrix (default) by explicitly specifying the template parameter

## Parameters

<i>D</i>	Dimension of the Hilbert space
----------	--------------------------------

## Returns

Identity gate

**7.3.3.5** `cmat qpp::Gates::Rn ( double theta, std::vector< double > n ) const` `[inline]`

Rotation of *theta* about the 3-dimensional real unit vector *n*.

## Parameters

<i>theta</i>	Rotation angle
<i>n</i>	3-dimensional real unit vector

## Returns

Rotation gate

**7.3.3.6** `cmat qpp::Gates::Xd ( std::size_t D ) const` `[inline]`

Generalized X gate for qudits.

## Note

Defined as  $X = \sum_j |j \oplus 1\rangle \langle j|$



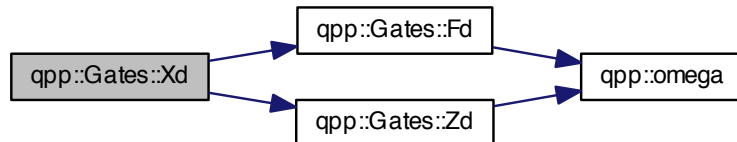
## Parameters

$D$	Dimension of the Hilbert space
-----	--------------------------------

## Returns

Generalized X gate for qudits

Here is the call graph for this function:



### 7.3.3.7 `cmat qpp::Gates::Zd ( std::size_t D ) const [inline]`

Generalized Z gate for qudits.

## Note

Defined as  $Z = \sum_j \exp(2\pi i j / D) |j\rangle\langle j|$

## Parameters

$D$	Dimension of the Hilbert space
-----	--------------------------------

## Returns

Generalized Z gate for qudits

Here is the call graph for this function:



## 7.3.4 Friends And Related Function Documentation

### 7.3.4.1 `friend class internal::Singleton< const Gates > [friend]`

## 7.3.5 Member Data Documentation

7.3.5.1 `cmat qpp::Gates::CNOTab { cmat::Identity(4, 4) }`

Controlled-NOT control target gate.

7.3.5.2 `cmat qpp::Gates::CNOTba { cmat::Zero(4, 4) }`

Controlled-NOT target control gate.

7.3.5.3 `cmat qpp::Gates::CZ { cmat::Identity(4, 4) }`

Controlled-Phase gate.

7.3.5.4 `cmat qpp::Gates::FRED { cmat::Identity(8, 8) }`

Fredkin gate.

7.3.5.5 `cmat qpp::Gates::H { cmat::Zero(2, 2) }`

Hadamard gate.

7.3.5.6 `cmat qpp::Gates::Id2 { cmat::Identity(2, 2) }`

Identity gate.

7.3.5.7 `cmat qpp::Gates::S { cmat::Zero(2, 2) }`

S gate.

7.3.5.8 `cmat qpp::Gates::SWAP { cmat::Identity(4, 4) }`

SWAP gate.

7.3.5.9 `cmat qpp::Gates::T { cmat::Zero(2, 2) }`

T gate.

7.3.5.10 `cmat qpp::Gates::TOF { cmat::Identity(8, 8) }`

Toffoli gate.

7.3.5.11 `cmat qpp::Gates::X { cmat::Zero(2, 2) }`

Pauli Sigma-X gate.

7.3.5.12 `cmat qpp::Gates::Y { cmat::Zero(2, 2) }`

Pauli Sigma-Y gate.

7.3.5.13 `cmat qpp::Gates::Z { cmat::Zero(2, 2) }`

Pauli Sigma-Z gate.

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

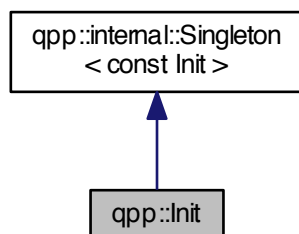
- [include/classes/gates.h](#)

## 7.4 qpp::Init Class Reference

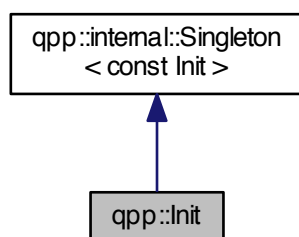
const Singleton class that performs additional initializations/cleanups

```
#include <init.h>
```

Inheritance diagram for qpp::Init:



Collaboration diagram for qpp::Init:



### Public Member Functions

- [Init \(\)](#)

*Additional initializations.*

## Private Member Functions

- [~Init\(\)](#)  
*Cleanups.*

## Friends

- class [internal::Singleton< const Init >](#)

## Additional Inherited Members

### 7.4.1 Detailed Description

const Singleton class that performs additional initializations/cleanups

### 7.4.2 Constructor & Destructor Documentation

#### 7.4.2.1 [qpp::Init::Init\(\)](#) [inline]

Additional initializations.

#### 7.4.2.2 [qpp::Init::~~Init\(\)](#) [inline], [private]

Cleanups.

### 7.4.3 Friends And Related Function Documentation

#### 7.4.3.1 [friend class internal::Singleton< const Init >](#) [friend]

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

- include/classes/[init.h](#)

## 7.5 [qpp::experimental::Qudit](#) Class Reference

```
#include <qudit.h>
```

## Public Member Functions

- [Qudit](#) (const [cmat](#) &rho=[States::get\\_instance\(\)](#).pz0)
- [std::size\\_t measure](#) (const [cmat](#) &U, bool destructive=false)
- [std::size\\_t measure](#) (bool destructive=false)
- [cmat getRho\(\)](#) const
- [std::size\\_t getD\(\)](#) const

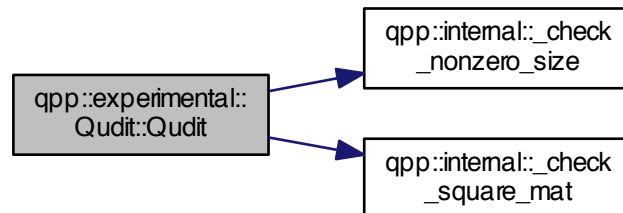
## Private Attributes

- [cmat\\_rho](#)
- [std::size\\_t \\_D](#)

### 7.5.1 Constructor & Destructor Documentation

7.5.1.1 `qpp::experimental::Qudit::Qudit ( const cmat & rho = States::get_instance() .pz0 ) [inline]`

Here is the call graph for this function:



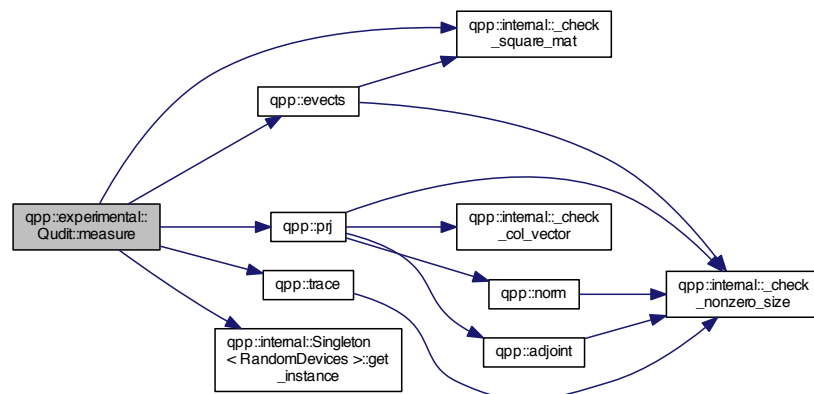
### 7.5.2 Member Function Documentation

7.5.2.1 `std::size_t qpp::experimental::Qudit::getD ( ) const [inline]`

7.5.2.2 `cmat qpp::experimental::Qudit::getRho ( ) const [inline]`

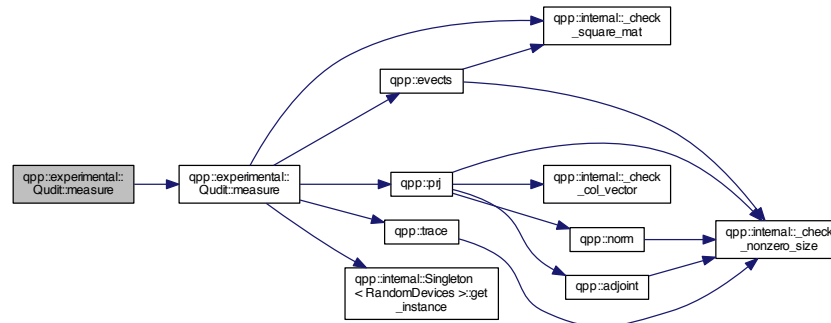
7.5.2.3 `std::size_t qpp::experimental::Qudit::measure ( const cmat & U, bool destructive = false ) [inline]`

Here is the call graph for this function:



#### 7.5.2.4 `std::size_t qpp::experimental::Qudit::measure ( bool destructive = false ) [inline]`

Here is the call graph for this function:



### 7.5.3 Member Data Documentation

#### 7.5.3.1 `std::size_t qpp::experimental::Qudit::_D [private]`

#### 7.5.3.2 `cmat qpp::experimental::Qudit::_rho [private]`

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

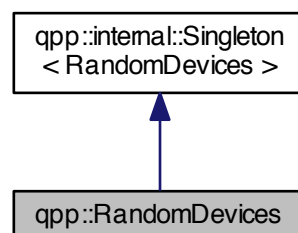
- [include/experimental/classes/qudit.h](#)

## 7.6 qpp::RandomDevices Class Reference

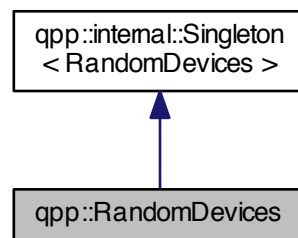
Singleton class that manages the source of randomness in the library.

```
#include <randevs.h>
```

Inheritance diagram for qpp::RandomDevices:



Collaboration diagram for qpp::RandomDevices:



### Public Attributes

- `std::mt19937 _rng`  
*Mersenne twister random number generator engine.*

### Private Member Functions

- `RandomDevices ()`  
*Initializes and seeds the random number generators.*

### Private Attributes

- `std::random_device _rd`  
*used to seed std::mt19937 \_rng*

### Friends

- class `internal::Singleton < RandomDevices >`

### Additional Inherited Members

#### 7.6.1 Detailed Description

Singleton class that manages the source of randomness in the library.

It consists of a wrapper around an `std::mt19937` Mersenne twister random number generator engine and an `std::random_device` engine. The latter is used to seed the Mersenne twister. The class also seeds the standard `std::srand` C number generator, as it is used by Eigen.

#### 7.6.2 Constructor & Destructor Documentation

##### 7.6.2.1 `qpp::RandomDevices::RandomDevices ( )` `[inline]`, `[private]`

Initializes and seeds the random number generators.

### 7.6.3 Friends And Related Function Documentation

7.6.3.1 friend class `internal::Singleton< RandomDevices >` [`friend`]

### 7.6.4 Member Data Documentation

7.6.4.1 `std::random_device qpp::RandomDevices::_rd` [`private`]

used to seed `std::mt19937 _rng`

7.6.4.2 `std::mt19937 qpp::RandomDevices::_rng`

Mersenne twister random number generator engine.

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

- `include/classes/randevs.h`

## 7.7 `qpp::internal::Singleton< T >` Class Template Reference

`Singleton` policy class, used internally to implement the singleton pattern via CRTP (Curiously recurring template pattern)

```
#include <singleton.h>
```

### Static Public Member Functions

- static `T & get_instance ()`

### Protected Member Functions

- `Singleton ()`=default
- virtual `~Singleton ()`
- `Singleton (const Singleton &)=delete`
- `Singleton & operator= (const Singleton &)=delete`

#### 7.7.1 Detailed Description

```
template<typename T>class qpp::internal::Singleton< T >
```

`Singleton` policy class, used internally to implement the singleton pattern via CRTP (Curiously recurring template pattern)

To implement a singleton, derive your class from `qpp::internal::Singleton`, make `qpp::internal::Singleton` a friend of your class, then declare the constructor of your class as private. To get an instance, use the static member function `qpp::internal::Singleton::get_instance()`, which returns a reference to your newly created singleton (thread-safe in C++11).

Example:

```
class MySingleton: public qpp::internal::Singleton<MySingleton>
{
    friend class qpp::internal::Singleton<MySingleton>;
public:
    // Declare all public members here
private:
    MySingleton()
```



```

    {
        // Implement the constructor here
    }
};

MySingleton& mySingleton = MySingleton::get_instance(); // Get an instance

```

See also

Code of [qpp::Gates](#), [qpp::RandomDevices](#), [qpp::States](#) or [qpp.h](#) for real world examples of usage.

## 7.7.2 Constructor & Destructor Documentation

**7.7.2.1** `template<typename T> qpp::internal::Singleton< T >::Singleton ( )` [protected],[default]

**7.7.2.2** `template<typename T> virtual qpp::internal::Singleton< T >::~~Singleton ( )` [inline],[protected],[virtual]

**7.7.2.3** `template<typename T> qpp::internal::Singleton< T >::Singleton ( const Singleton< T > & )` [protected],[delete]

## 7.7.3 Member Function Documentation

**7.7.3.1** `template<typename T> static T& qpp::internal::Singleton< T >::get_instance ( )` [inline],[static]

**7.7.3.2** `template<typename T> Singleton& qpp::internal::Singleton< T >::operator= ( const Singleton< T > & )` [protected],[delete]

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

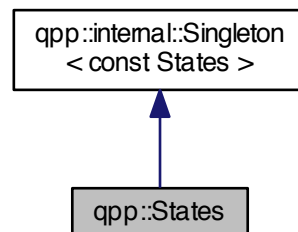
- include/classes/[singleton.h](#)

## 7.8 qpp::States Class Reference

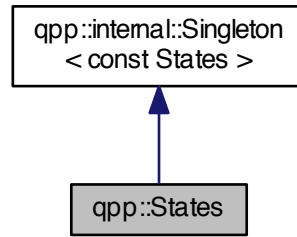
const Singleton class that implements most commonly used states

```
#include <states.h>
```

Inheritance diagram for qpp::States:



Collaboration diagram for `qpp::States`:



## Public Attributes

- `ket x0` { `ket::Zero(2)` }  
*Pauli Sigma-X 0-eigenstate  $|+\rangle$*
- `ket x1` { `ket::Zero(2)` }  
*Pauli Sigma-X 1-eigenstate  $|-\rangle$*
- `ket y0` { `ket::Zero(2)` }  
*Pauli Sigma-Y 0-eigenstate.*
- `ket y1` { `ket::Zero(2)` }  
*Pauli Sigma-Y 1-eigenstate.*
- `ket z0` { `ket::Zero(2)` }  
*Pauli Sigma-Z 0-eigenstate  $|0\rangle$*
- `ket z1` { `ket::Zero(2)` }  
*Pauli Sigma-Z 1-eigenstate  $|1\rangle$*
- `cmat px0` { `cmat::Zero(2, 2)` }  
*Projector onto the Pauli Sigma-X 0-eigenstate  $|+\rangle\langle+|$ .*
- `cmat px1` { `cmat::Zero(2, 2)` }  
*Projector onto the Pauli Sigma-X 1-eigenstate  $|-\rangle\langle-|$ .*
- `cmat py0` { `cmat::Zero(2, 2)` }  
*Projector onto the Pauli Sigma-Y 0-eigenstate.*
- `cmat py1` { `cmat::Zero(2, 2)` }  
*Projector onto the Pauli Sigma-Y 1-eigenstate.*
- `cmat pz0` { `cmat::Zero(2, 2)` }  
*Projector onto the Pauli Sigma-Z 0-eigenstate  $|0\rangle\langle 0|$ .*
- `cmat pz1` { `cmat::Zero(2, 2)` }  
*Projector onto the Pauli Sigma-Z 1-eigenstate  $|1\rangle\langle 1|$ .*
- `ket b00` { `ket::Zero(4)` }  
*Bell-00 state (following the convention in Nielsen and Chuang)*
- `ket b01` { `ket::Zero(4)` }  
*Bell-01 state (following the convention in Nielsen and Chuang)*
- `ket b10` { `ket::Zero(4)` }  
*Bell-10 state (following the convention in Nielsen and Chuang)*
- `ket b11` { `ket::Zero(4)` }  
*Bell-11 state (following the convention in Nielsen and Chuang)*
- `cmat pb00` { `cmat::Zero(4, 4)` }

- Projector onto the Bell-00 state.*
- `cmat pb01 { cmat::Zero(4, 4) }`
- Projector onto the Bell-01 state.*
- `cmat pb10 { cmat::Zero(4, 4) }`
- Projector onto the Bell-10 state.*
- `cmat pb11 { cmat::Zero(4, 4) }`
- Projector onto the Bell-11 state.*
- `ket GHZ { ket::Zero(8) }`
- GHZ state.*
- `ket W { ket::Zero(8) }`
- W state.*
- `cmat pGHZ { cmat::Zero(8, 8) }`
- Projector onto the GHZ state.*
- `cmat pW { cmat::Zero(8, 8) }`
- Projector onto the W state.*

### Private Member Functions

- `States ()`

### Friends

- class `internal::Singleton< const States >`

### Additional Inherited Members

#### 7.8.1 Detailed Description

const Singleton class that implements most commonly used states

#### 7.8.2 Constructor & Destructor Documentation

7.8.2.1 `qpp::States::States ( ) [inline], [private]`

Initialize the states

#### 7.8.3 Friends And Related Function Documentation

7.8.3.1 `friend class internal::Singleton< const States > [friend]`

#### 7.8.4 Member Data Documentation

7.8.4.1 `ket qpp::States::b00 { ket::Zero(4) }`

Bell-00 state (following the convention in Nielsen and Chuang)

7.8.4.2 `ket qpp::States::b01 { ket::Zero(4) }`

Bell-01 state (following the convention in Nielsen and Chuang)

#### 7.8.4.3 ket qpp::States::b10 { ket::Zero(4) }

Bell-10 state (following the convention in Nielsen and Chuang)

#### 7.8.4.4 ket qpp::States::b11 { ket::Zero(4) }

Bell-11 state (following the convention in Nielsen and Chuang)

#### 7.8.4.5 ket qpp::States::GHZ { ket::Zero(8) }

GHZ state.

#### 7.8.4.6 cmat qpp::States::pb00 { cmat::Zero(4, 4) }

Projector onto the Bell-00 state.

#### 7.8.4.7 cmat qpp::States::pb01 { cmat::Zero(4, 4) }

Projector onto the Bell-01 state.

#### 7.8.4.8 cmat qpp::States::pb10 { cmat::Zero(4, 4) }

Projector onto the Bell-10 state.

#### 7.8.4.9 cmat qpp::States::pb11 { cmat::Zero(4, 4) }

Projector onto the Bell-11 state.

#### 7.8.4.10 cmat qpp::States::pGHZ { cmat::Zero(8, 8) }

Projector onto the GHZ state.

#### 7.8.4.11 cmat qpp::States::pW { cmat::Zero(8, 8) }

Projector onto the W state.

#### 7.8.4.12 cmat qpp::States::px0 { cmat::Zero(2, 2) }

Projector onto the Pauli Sigma-X 0-eigenstate  $|+\rangle\langle+|$ .

#### 7.8.4.13 cmat qpp::States::px1 { cmat::Zero(2, 2) }

Projector onto the Pauli Sigma-X 1-eigenstate  $|-\rangle\langle-|$ .

#### 7.8.4.14 cmat qpp::States::py0 { cmat::Zero(2, 2) }

Projector onto the Pauli Sigma-Y 0-eigenstate.

7.8.4.15 `cmat qpp::States::py1 { cmat::Zero(2, 2) }`

Projector onto the Pauli Sigma-Y 1-eigenstate.

7.8.4.16 `cmat qpp::States::pz0 { cmat::Zero(2, 2) }`

Projector onto the Pauli Sigma-Z 0-eigenstate  $|0\rangle\langle 0|$ .

7.8.4.17 `cmat qpp::States::pz1 { cmat::Zero(2, 2) }`

Projector onto the Pauli Sigma-Z 1-eigenstate  $|1\rangle\langle 1|$ .

7.8.4.18 `ket qpp::States::W { ket::Zero(8) }`

W state.

7.8.4.19 `ket qpp::States::x0 { ket::Zero(2) }`

Pauli Sigma-X 0-eigenstate  $|+\rangle$

7.8.4.20 `ket qpp::States::x1 { ket::Zero(2) }`

Pauli Sigma-X 1-eigenstate  $|-\rangle$

7.8.4.21 `ket qpp::States::y0 { ket::Zero(2) }`

Pauli Sigma-Y 0-eigenstate.

7.8.4.22 `ket qpp::States::y1 { ket::Zero(2) }`

Pauli Sigma-Y 1-eigenstate.

7.8.4.23 `ket qpp::States::z0 { ket::Zero(2) }`

Pauli Sigma-Z 0-eigenstate  $|0\rangle$

7.8.4.24 `ket qpp::States::z1 { ket::Zero(2) }`

Pauli Sigma-Z 1-eigenstate  $|1\rangle$

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

- [include/classes/states.h](#)

## 7.9 qpp::Timer Class Reference

Measures time.

```
#include <timer.h>
```

## Public Member Functions

- [Timer](#) ()  
*Constructs an instance with the current time as the starting point.*
- void [tic](#) ()  
*Resets the chronometer.*
- const [Timer](#) & [toc](#) ()  
*Stops the chronometer.*
- double [seconds](#) () const  
*Time passed in seconds.*

## Protected Attributes

- std::chrono::steady\_clock::time\_point [\\_start](#)
- std::chrono::steady\_clock::time\_point [\\_end](#)

## Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [Timer](#) &rhs)  
*Overload for std::ostream operators.*

### 7.9.1 Detailed Description

Measures time.

Uses a std::chrono::steady\_clock. It is not affected by wall clock changes during runtime.

### 7.9.2 Constructor & Destructor Documentation

#### 7.9.2.1 `qpp::Timer::Timer ( )` `[inline]`

Constructs an instance with the current time as the starting point.

### 7.9.3 Member Function Documentation

#### 7.9.3.1 `double qpp::Timer::seconds ( )` const `[inline]`

Time passed in seconds.

#### Returns

Number of seconds that passed between the instantiation/reset and invocation of `qpp::Timer::toc()`

#### 7.9.3.2 `void qpp::Timer::tic ( )` `[inline]`

Resets the chronometer.

Resets the starting/ending point to the current time

### 7.9.3.3 const Timer& qpp::Timer::toc ( ) [inline]

Stops the chronometer.

Set the current time as the ending point

#### Returns

Current instance

## 7.9.4 Friends And Related Function Documentation

### 7.9.4.1 std::ostream& operator<< ( std::ostream & *os*, const Timer & *rhs* ) [friend]

Overload for std::ostream operators.

#### Parameters

<i>os</i>	Output stream
<i>rhs</i>	<a href="#">Timer</a> instance

#### Returns

Writes to the output stream the number of seconds that passed between the instantiation/reset and invocation of [qpp::Timer::toc\(\)](#).

## 7.9.5 Member Data Documentation

### 7.9.5.1 std::chrono::steady\_clock::time\_point qpp::Timer::\_end [protected]

### 7.9.5.2 std::chrono::steady\_clock::time\_point qpp::Timer::\_start [protected]

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

- include/classes/[timer.h](#)



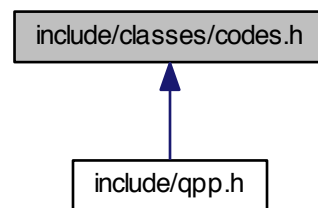


## Chapter 8

# File Documentation

### 8.1 include/classes/codes.h File Reference

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



#### Classes

- class [qpp::Codes](#)

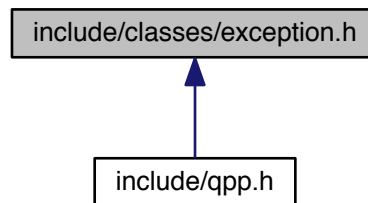
*const Singleton class that defines quantum error correcting codes*

#### Namespaces

- [qpp](#)

## 8.2 include/classes/exception.h File Reference

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



### Classes

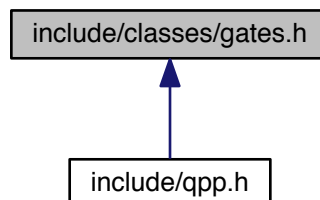
- class [qpp::Exception](#)  
*Generates custom exceptions, used when validating function parameters.*

### Namespaces

- [qpp](#)

## 8.3 include/classes/gates.h File Reference

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



### Classes

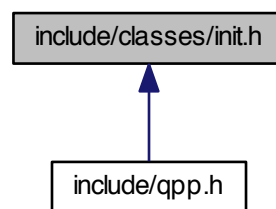
- class [qpp::Gates](#)  
*const Singleton class that implements most commonly used gates*

## Namespaces

- [qpp](#)

## 8.4 include/classes/init.h File Reference

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



## Classes

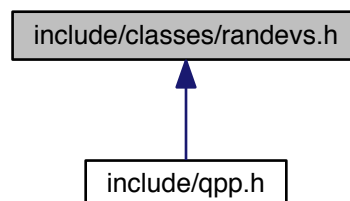
- class [qpp::Init](#)  
*const Singleton class that performs additional initializations/cleanups*

## Namespaces

- [qpp](#)

## 8.5 include/classes/randevs.h File Reference

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



## Classes

- class [qpp::RandomDevices](#)

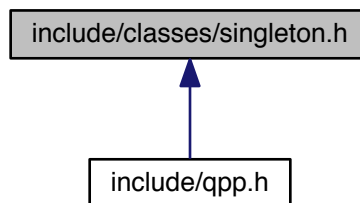
*Singleton class that manages the source of randomness in the library.*

## Namespaces

- [qpp](#)

## 8.6 include/classes/singleton.h File Reference

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



## Classes

- class [qpp::internal::Singleton< T >](#)

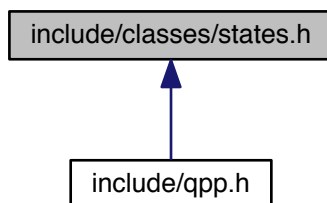
*[Singleton](#) policy class, used internally to implement the singleton pattern via CRTP (Curiously recurring template pattern)*

## Namespaces

- [qpp](#)
- [qpp::internal](#)

## 8.7 include/classes/states.h File Reference

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



### Classes

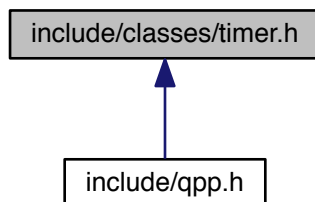
- class [qpp::States](#)  
*const Singleton class that implements most commonly used states*

### Namespaces

- [qpp](#)

## 8.8 include/classes/timer.h File Reference

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



### Classes

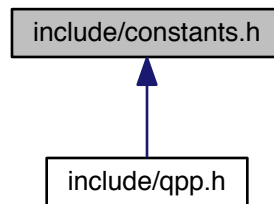
- class [qpp::Timer](#)  
*Measures time.*

## Namespaces

- [qpp](#)

## 8.9 include/constants.h File Reference

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



## Namespaces

- [qpp](#)

## Functions

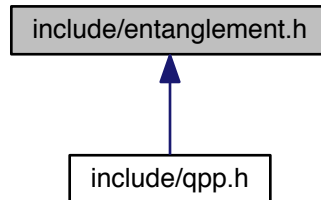
- constexpr std::complex< double > [qpp::operator""\\_i](#) (unsigned long long int x)  
*User-defined literal for complex  $i = \sqrt{-1}$  (integer overload)*
- constexpr std::complex< double > [qpp::operator""\\_i](#) (long double x)  
*User-defined literal for complex  $i = \sqrt{-1}$  (real overload)*
- std::complex< double > [qpp::omega](#) (std::size\_t D)  
*D-th root of unity.*

## Variables

- constexpr double [qpp::chop](#) = 1e-10  
*Used in [qpp::disp\(\)](#) and [qpp::displn\(\)](#) for setting to zero numbers that have their absolute value smaller than [qpp::ct](#)→  
[::chop](#).*
- constexpr double [qpp::eps](#) = 1e-12  
*Used to decide whether a number or expression in double precision is zero or not.*
- constexpr std::size\_t [qpp::maxn](#) = 64  
*Maximum number of qubits.*
- constexpr double [qpp::pi](#) = 3.141592653589793238462643383279502884  
 $\pi$
- constexpr double [qpp::ee](#) = 2.718281828459045235360287471352662497  
*Base of natural logarithm,  $e$ .*
- constexpr std::size\_t [qpp::infy](#) = -1  
*Used to denote infinity.*

## 8.10 include/entanglement.h File Reference

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



### Namespaces

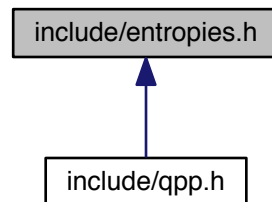
- [qpp](#)

### Functions

- `template<typename Derived >`  
`DynColVect< cplx > qpp::schmidtcoeff (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Schmidt coefficients of the bi-partite pure state A.*
- `template<typename Derived >`  
`cmat qpp::schmidtU (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Schmidt basis on Alice's side.*
- `template<typename Derived >`  
`cmat qpp::schmidtV (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Schmidt basis on Bob's side.*
- `template<typename Derived >`  
`DynColVect< double > qpp::schmidtprob (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Schmidt probabilities of the bi-partite pure state A.*
- `template<typename Derived >`  
`double qpp::entanglement (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Entanglement of the bi-partite pure state A.*
- `template<typename Derived >`  
`double qpp::gconcurrence (const Eigen::MatrixBase< Derived > &A)`  
*G-concurrence of the bi-partite pure state A.*
- `template<typename Derived >`  
`double qpp::negativity (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Negativity of the bi-partite mixed state A.*
- `template<typename Derived >`  
`double qpp::lognegativity (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`  
*Logarithmic negativity of the bi-partite mixed state A.*
- `template<typename Derived >`  
`double qpp::concurrence (const Eigen::MatrixBase< Derived > &A)`  
*Wootters concurrence of the bi-partite qubit mixed state A.*

## 8.11 include/entropies.h File Reference

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



### Namespaces

- [qpp](#)

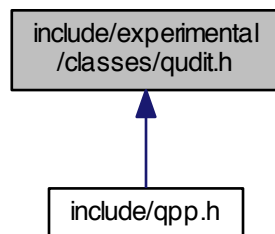
### Functions

- `template<typename Derived >`  
`double qpp::shannon (const Eigen::MatrixBase< Derived > &A)`  
*Shannon/von-Neumann entropy of the probability distribution/density matrix A.*
- `template<typename Derived >`  
`double qpp::renyi (const Eigen::MatrixBase< Derived > &A, double alpha)`  
*Renyi-  $\alpha$  entropy of the probability distribution/density matrix A, for  $\alpha \geq 0$ .*
- `template<typename Derived >`  
`double qpp::tsallis (const Eigen::MatrixBase< Derived > &A, double alpha)`  
*Tsallis-  $\alpha$  entropy of the probability distribution/density matrix A, for  $\alpha \geq 0$*   
*.*
- `template<typename Derived >`  
`double qpp::qmutualinfo (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &subsysA, const std::vector< std::size_t > &subsysB, const std::vector< std::size_t > &dims)`  
*Quantum mutual information between 2 subsystems of a composite system.*



## 8.12 include/experimental/classes/qudit.h File Reference

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



### Classes

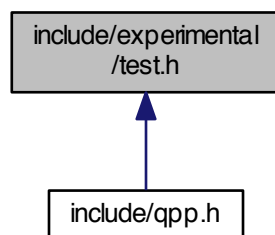
- class [qpp::experimental::Qudit](#)

### Namespaces

- [qpp](#)
- [qpp::experimental](#)

## 8.13 include/experimental/test.h File Reference

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



### Namespaces

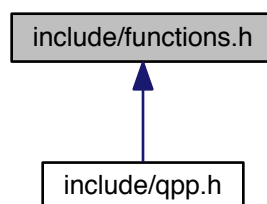
- [qpp::experimental](#)
- [qpp](#)

## Functions

- `template<typename Derived1 , typename Derived2 >`  
`DynMat< typename Derived1::Scalar > qpp::experimental::apply (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< std::size_t > &subsys, const std::vector< std::size_t > &dims)`  
*Applies the gate A to the part subsys of a multipartite state vector or density matrix.*
- `template<typename Derived >`  
`cmat qpp::experimental::channel (const Eigen::MatrixBase< Derived > &rho, const std::vector< cmat > &Ks, const std::vector< std::size_t > &subsys, const std::vector< std::size_t > &dims)`  
*Applies the channel specified by the set of Kraus operators Ks to the part of the density matrix rho specified by subsys.*
- `cmat qpp::experimental::super (const std::vector< cmat > &Ks)`  
*Superoperator matrix representation.*
- `template<typename Derived >`  
`DynMat< typename Derived::Scalar > qpp::experimental::CTRL (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &ctrl, const std::vector< std::size_t > &subsys, std::size_t n, std::size_t d=2)`  
*Generates the multipartite multiple-controlled-A gate in matrix form.*
- `cmat qpp::experimental::choi (const std::vector< cmat > &Ks)`  
*Choi matrix representation.*
- `std::vector< cmat > qpp::experimental::randkraus (std::size_t n, std::size_t D)`  
*Generates a set of random Kraus operators.*
- `template<typename Derived >`  
`double qpp::experimental::renyi\_inf (const Eigen::MatrixBase< Derived > &A)`  
*Renyi- $\infty$  entropy (min entropy) of the probability distribution/density matrix A.*

## 8.14 include/functions.h File Reference

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



## Namespaces

- [qpp](#)

## Functions

- `template<typename Derived >`  
`DynMat< typename Derived::Scalar > qpp::transpose (const Eigen::MatrixBase< Derived > &A)`

*Transpose.*

- `template<typename Derived >`  
`DynMat< typename Derived::Scalar > qpp::conjugate (const Eigen::MatrixBase< Derived > &A)`

*Complex conjugate.*

- `template<typename Derived >`  
`DynMat< typename Derived::Scalar > qpp::adjoint (const Eigen::MatrixBase< Derived > &A)`

*Adjoint.*

- `template<typename Derived >`  
`DynMat< typename Derived::Scalar > qpp::inverse (const Eigen::MatrixBase< Derived > &A)`

*Inverse.*

- `template<typename Derived >`  
`Derived::Scalar qpp::trace (const Eigen::MatrixBase< Derived > &A)`

*Trace.*

- `template<typename Derived >`  
`Derived::Scalar qpp::det (const Eigen::MatrixBase< Derived > &A)`

*Determinant.*

- `template<typename Derived >`  
`Derived::Scalar qpp::logdet (const Eigen::MatrixBase< Derived > &A)`

*Logarithm of the determinant.*

- `template<typename Derived >`  
`Derived::Scalar qpp::sum (const Eigen::MatrixBase< Derived > &A)`

*Element-wise sum of A.*

- `template<typename Derived >`  
`Derived::Scalar qpp::prod (const Eigen::MatrixBase< Derived > &A)`

*Element-wise product of A.*

- `template<typename Derived >`  
`double qpp::norm (const Eigen::MatrixBase< Derived > &A)`

*Frobenius norm.*

- `template<typename Derived >`  
`DynColVect< cplx > qpp::evals (const Eigen::MatrixBase< Derived > &A)`

*Eigenvalues.*

- `template<typename Derived >`  
`cmat qpp::evecs (const Eigen::MatrixBase< Derived > &A)`

*Eigenvectors.*

- `template<typename Derived >`  
`DynColVect< double > qpp::hevals (const Eigen::MatrixBase< Derived > &A)`

*Hermitian eigenvalues.*

- `template<typename Derived >`  
`cmat qpp::hevecs (const Eigen::MatrixBase< Derived > &A)`

*Hermitian eigenvectors.*

- `template<typename Derived >`  
`DynColVect< double > qpp::svals (const Eigen::MatrixBase< Derived > &A)`

*Singular values.*

- `template<typename Derived >`  
`cmat qpp::svdU (const Eigen::MatrixBase< Derived > &A)`

*Left singular vectors.*

- `template<typename Derived >`  
`cmat qpp::svdV (const Eigen::MatrixBase< Derived > &A)`

*Right singular vectors.*

- `template<typename Derived >`  
`cmat qpp::funm (const Eigen::MatrixBase< Derived > &A, cplx(*f)(const cplx &))`

*Functional calculus  $f(A)$*

- template<typename Derived >  
cmat [qpp::sqrtm](#) (const Eigen::MatrixBase< Derived > &A)  
*Matrix square root.*
- template<typename Derived >  
cmat [qpp::absm](#) (const Eigen::MatrixBase< Derived > &A)  
*Matrix absolut value.*
- template<typename Derived >  
cmat [qpp::expm](#) (const Eigen::MatrixBase< Derived > &A)  
*Matrix exponential.*
- template<typename Derived >  
cmat [qpp::logm](#) (const Eigen::MatrixBase< Derived > &A)  
*Matrix logarithm.*
- template<typename Derived >  
cmat [qpp::sinm](#) (const Eigen::MatrixBase< Derived > &A)  
*Matrix sin.*
- template<typename Derived >  
cmat [qpp::cosm](#) (const Eigen::MatrixBase< Derived > &A)  
*Matrix cos.*
- template<typename Derived >  
cmat [qpp::spectralpowm](#) (const Eigen::MatrixBase< Derived > &A, const cplx z)  
*Matrix power.*
- template<typename Derived >  
DynMat< typename Derived::Scalar > [qpp::powm](#) (const Eigen::MatrixBase< Derived > &A, std::size\_t n)  
*Matrix power.*
- template<typename Derived >  
double [qpp::schatten](#) (const Eigen::MatrixBase< Derived > &A, std::size\_t p)  
*Schatten norm.*
- template<typename OutputScalar , typename Derived >  
DynMat< OutputScalar > [qpp::cwise](#) (const Eigen::MatrixBase< Derived > &A, OutputScalar(\*f)(const type-  
name Derived::Scalar &))  
*Functor.*
- template<typename T >  
DynMat< typename T::Scalar > [qpp::kron](#) (const T &head)  
*Kronecker product (variadic overload)*
- template<typename T , typename... Args>  
DynMat< typename T::Scalar > [qpp::kron](#) (const T &head, const Args &...tail)  
*Kronecker product (variadic overload)*
- template<typename Derived >  
DynMat< typename Derived::Scalar > [qpp::kron](#) (const std::vector< Derived > &As)  
*Kronecker product (std::vector overload)*
- template<typename Derived >  
DynMat< typename Derived::Scalar > [qpp::kron](#) (const std::initializer\_list< Derived > &As)  
*Kronecker product (std::initializer\_list overload)*
- template<typename Derived >  
DynMat< typename Derived::Scalar > [qpp::kronpow](#) (const Eigen::MatrixBase< Derived > &A, std::size\_t  
n)  
*Kronecker power.*
- template<typename Derived >  
DynMat< typename Derived::Scalar > [qpp::reshape](#) (const Eigen::MatrixBase< Derived > &A, std::size\_t  
rows, std::size\_t cols)  
*Reshape.*
- template<typename Derived1 , typename Derived2 >  
DynMat< typename Derived1::Scalar > [qpp::comm](#) (const Eigen::MatrixBase< Derived1 > &A, const  
Eigen::MatrixBase< Derived2 > &B)

*Commutator.*

- template<typename Derived1 , typename Derived2 >  
DynMat< typename Derived1::Scalar > [qpp::anticomm](#) (const Eigen::MatrixBase< Derived1 > &A, const Eigen::MatrixBase< Derived2 > &B)

*Anti-commutator.*

- template<typename Derived >  
DynMat< typename Derived::Scalar > [qpp::prj](#) (const Eigen::MatrixBase< Derived > &V)

*Projector.*

- template<typename Derived >  
DynMat< typename Derived::Scalar > [qpp::grams](#) (const std::vector< Derived > &Vs)

*Gram-Schmidt orthogonalization (std::vector overload)*

- template<typename Derived >  
DynMat< typename Derived::Scalar > [qpp::grams](#) (const std::initializer\_list< Derived > &Vs)

*Gram-Schmidt orthogonalization (std::initializer\_list overload)*

- template<typename Derived >  
DynMat< typename Derived::Scalar > [qpp::grams](#) (const Eigen::MatrixBase< Derived > &A)

*Gram-Schmidt orthogonalization (Eigen expression (matrix) overload)*

- std::vector< std::size\_t > [qpp::n2multiidx](#) (std::size\_t n, const std::vector< std::size\_t > &dims)

*Non-negative integer index to multi-index.*

- std::size\_t [qpp::multiidx2n](#) (const std::vector< std::size\_t > &midx, const std::vector< std::size\_t > &dims)

*Multi-index to non-negative integer index.*

- ket [qpp::mket](#) (const std::vector< std::size\_t > &mask, const std::vector< std::size\_t > &dims)

*Multi-partite qudit ket (different dimensions overload)*

- ket [qpp::mket](#) (const std::vector< std::size\_t > &mask, std::size\_t d=2)

*Multi-partite qudit ket (same dimensions overload)*

- cmat [qpp::mprj](#) (const std::vector< std::size\_t > &mask, const std::vector< std::size\_t > &dims)

*Projector onto multi-partite qudit ket (different dimensions overload)*

- cmat [qpp::mprj](#) (const std::vector< std::size\_t > &mask, std::size\_t d=2)

*Projector onto multi-partite qudit ket (same dimensions overload)*

- std::vector< std::size\_t > [qpp::invperm](#) (const std::vector< std::size\_t > &perm)

*Inverse permutation.*

- std::vector< std::size\_t > [qpp::compperm](#) (const std::vector< std::size\_t > &perm, const std::vector< std::size\_t > &sigma)

*Compose permutations.*

- template<typename InputIterator >  
std::vector< double > [qpp::abssq](#) (InputIterator first, InputIterator last)

*Computes the absolut values squared of a range of complex numbers.*

- template<typename Derived >  
std::vector< double > [qpp::abssq](#) (const Eigen::MatrixBase< Derived > &V)

*Computes the absolut values squared of a column vector.*

- template<typename InputIterator >  
auto [qpp::sum](#) (InputIterator first, InputIterator last) -> typename InputIterator::value\_type

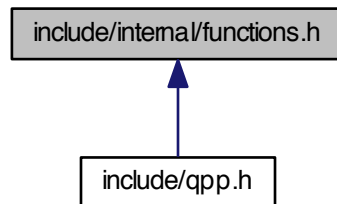
*Element-wise sum of a range.*

- template<typename InputIterator >  
auto [qpp::prod](#) (InputIterator first, InputIterator last) -> typename InputIterator::value\_type

*Element-wise product of a range.*

## 8.15 include/internal/functions.h File Reference

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



### Namespaces

- [qpp::internal](#)
- [qpp](#)

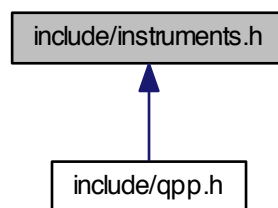
### Functions

- void [qpp::internal::\\_n2multiidx](#) (std::size\_t n, std::size\_t numdims, const std::size\_t \*dims, std::size\_t \*result)
- std::size\_t [qpp::internal::\\_multiidx2n](#) (const std::size\_t \*midx, std::size\_t numdims, const std::size\_t \*dims)
- template<typename Derived >  
bool [qpp::internal::\\_check\\_square\\_mat](#) (const Eigen::MatrixBase< Derived > &A)
- template<typename Derived >  
bool [qpp::internal::\\_check\\_vector](#) (const Eigen::MatrixBase< Derived > &A)
- template<typename Derived >  
bool [qpp::internal::\\_check\\_row\\_vector](#) (const Eigen::MatrixBase< Derived > &A)
- template<typename Derived >  
bool [qpp::internal::\\_check\\_col\\_vector](#) (const Eigen::MatrixBase< Derived > &A)
- template<typename T >  
bool [qpp::internal::\\_check\\_nonzero\\_size](#) (const T &x)
- bool [qpp::internal::\\_check\\_dims](#) (const std::vector< std::size\_t > &dims)
- template<typename Derived >  
bool [qpp::internal::\\_check\\_dims\\_match\\_mat](#) (const std::vector< std::size\_t > &dims, const Eigen::MatrixBase< Derived > &A)
- template<typename Derived >  
bool [qpp::internal::\\_check\\_dims\\_match\\_cvect](#) (const std::vector< std::size\_t > &dims, const Eigen::MatrixBase< Derived > &V)
- template<typename Derived >  
bool [qpp::internal::\\_check\\_dims\\_match\\_rvect](#) (const std::vector< std::size\_t > &dims, const Eigen::MatrixBase< Derived > &V)
- bool [qpp::internal::\\_check\\_eq\\_dims](#) (const std::vector< std::size\_t > &dims, std::size\_t dim)
- bool [qpp::internal::\\_check\\_subsys\\_match\\_dims](#) (const std::vector< std::size\_t > &subsys, const std::vector< std::size\_t > &dims)
- bool [qpp::internal::\\_check\\_perm](#) (const std::vector< std::size\_t > &perm)
- template<typename Derived1 , typename Derived2 >  
DynMat< typename Derived1::Scalar > [qpp::internal::\\_kron2](#) (const Eigen::MatrixBase< Derived1 > &A, const Eigen::MatrixBase< Derived2 > &B)

- `template<typename T >`  
`void qpp::internal::variadic\_vector\_emplace (std::vector< T > &)`
- `template<typename T , typename First , typename... Args>`  
`void qpp::internal::variadic\_vector\_emplace (std::vector< T > &v, First &&first, Args &&...args)`

## 8.16 include/instruments.h File Reference

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



### Namespaces

- [qpp](#)

### Functions

- `template<typename Derived >`  
`std::pair< std::vector< double >`  
`, std::vector< cmat > > qpp::measure (const Eigen::MatrixBase< Derived > &A, const std::vector< cmat >`  
`&Ks)`

*Measures the state A using the set of Kraus operators Ks.*

- `template<typename Derived >`  
`std::pair< std::vector< double >`  
`, std::vector< cmat > > qpp::measure (const Eigen::MatrixBase< Derived > &A, const std::initializer_list<`  
`cmat > &Ks)`

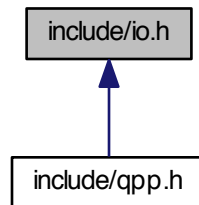
*Measures the state A using the set of Kraus operators Ks (std::initializer\_list overload)*

- `template<typename Derived >`  
`std::pair< std::vector< double >`  
`, std::vector< cmat > > qpp::measure (const Eigen::MatrixBase< Derived > &A, const cmat &M)`

*Measures the state A in the orthonormal basis specified by the eigenvectors of M.*

## 8.17 include/io.h File Reference

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



### Namespaces

- [qpp](#)

### Functions

- `template<typename InputIterator >`  
`void qpp::disp (const InputIterator &first, const InputIterator &last, const std::string &separator, const std::string &start="[", const std::string &end="]", std::ostream &os=std::cout)`  
*Displays a range. Does not add a newline.*
- `template<typename InputIterator >`  
`std::ostream & qpp::displn (const InputIterator &first, const InputIterator &last, const std::string &separator, const std::string &start="[", const std::string &end="]", std::ostream &os=std::cout)`  
*Displays a range. Adds a newline.*
- `template<typename T >`  
`std::ostream & qpp::disp (const T &x, const std::string &separator, const std::string &start="[", const std::string &end="]", std::ostream &os=std::cout)`  
*Displays a standard container that supports std::begin, std::end and forward iteration. Does not add a newline.*
- `template<typename T >`  
`std::ostream & qpp::displn (const T &x, const std::string &separator, const std::string &start="[", const std::string &end="]", std::ostream &os=std::cout)`  
*Displays a standard container that supports std::begin, std::end and forward iteration. Adds a newline.*
- `template<typename T >`  
`std::ostream & qpp::disp (const T *x, const std::size_t n, const std::string &separator, const std::string &start="[", const std::string &end="]", std::ostream &os=std::cout)`  
*Displays a C-style array. Does not add a newline.*
- `template<typename T >`  
`std::ostream & qpp::displn (const T *x, const std::size_t n, const std::string &separator, const std::string &start="[", const std::string &end="]", std::ostream &os=std::cout)`  
*Displays a C-style array. Adds a newline.*
- `template<typename Derived >`  
`std::ostream & qpp::disp (const Eigen::MatrixBase< Derived > &A, double chop=qpp::chop, std::ostream &os=std::cout)`  
*Displays an Eigen expression in matrix friendly form. Does not add a new line.*



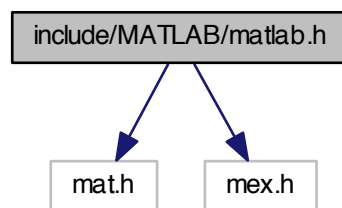
- `template<typename Derived >`  
`std::ostream & qpp::displn (const Eigen::MatrixBase< Derived > &A, double chop=qpp::chop, std::ostream &os=std::cout)`  
*Displays an Eigen expression in matrix friendly form. Adds a newline.*
- `std::ostream & qpp::disp (const cplx z, double chop=qpp::chop, std::ostream &os=std::cout)`  
*Displays a number (implicitly converted to `std::complex<double>`) in friendly form. Does not add a new line.*
- `std::ostream & qpp::displn (const cplx z, double chop=qpp::chop, std::ostream &os=std::cout)`  
*Displays a number (implicitly converted to `std::complex<double>`) in friendly form. Adds a new line.*
- `template<typename Derived >`  
`void qpp::save (const Eigen::MatrixBase< Derived > &A, const std::string &fname)`  
*Saves Eigen expression to a binary file (internal format) in double precision.*
- `template<typename Derived >`  
`DynMat< typename Derived::Scalar > qpp::load (const std::string &fname)`  
*Loads Eigen matrix from a binary file (internal format) in double precision.*

## 8.18 include/MATLAB/matlab.h File Reference

```
#include "mat.h"
```

```
#include "mex.h"
```

Include dependency graph for matlab.h:



### Namespaces

- `qpp`

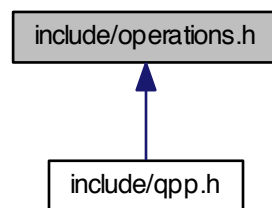
### Functions

- `template<typename Derived >`  
`Derived qpp::loadMATLABmatrix (const std::string &mat_file, const std::string &var_name)`  
*Loads an Eigen dynamic matrix from a MATLAB .mat file, generic version.*
- `template<>`  
`dmat qpp::loadMATLABmatrix (const std::string &mat_file, const std::string &var_name)`  
*Loads an Eigen dynamic matrix from a MATLAB .mat file, specialization for double matrices (`qpp::dmat`)*
- `template<>`  
`cmat qpp::loadMATLABmatrix (const std::string &mat_file, const std::string &var_name)`  
*Loads an Eigen dynamic matrix from a MATLAB .mat file, specialization for complex matrices (`qpp::cmat`)*

- `template<typename Derived >`  
`void qpp::saveMATLABmatrix (const Eigen::MatrixBase< Derived > &A, const std::string &mat_file, const std::string &var_name, const std::string &mode)`  
*Saves an Eigen dynamic matrix to a MATLAB .mat file, generic version.*
- `template<>`  
`void qpp::saveMATLABmatrix (const Eigen::MatrixBase< dmat > &A, const std::string &mat_file, const std::string &var_name, const std::string &mode)`  
*Saves an Eigen dynamic matrix to a MATLAB .mat file, specialization for double matrices ([qpp::dmat](#))*
- `template<>`  
`void qpp::saveMATLABmatrix (const Eigen::MatrixBase< cmat > &A, const std::string &mat_file, const std::string &var_name, const std::string &mode)`  
*Saves an Eigen dynamic matrix to a MATLAB .mat file, specialization for complex matrices ([qpp::cmat](#))*

## 8.19 include/operations.h File Reference

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



### Namespaces

- [qpp](#)

### Functions

- `template<typename Derived1 , typename Derived2 >`  
`DynMat< typename Derived1::Scalar > qpp::applyCTRL (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< std::size_t > &ctrl, const std::vector< std::size_t > &subsys, std::size_t n, std::size_t d=2)`  
*Applies the controlled-gate A to the part subsys of a multipartite state vector or density matrix.*
- `template<typename Derived1 , typename Derived2 >`  
`DynMat< typename Derived1::Scalar > qpp::apply (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< std::size_t > &subsys, std::size_t n, std::size_t d=2)`  
*Applies the gate A to the part subsys of a multipartite state vector or density matrix.*
- `template<typename Derived >`  
`cmat qpp::channel (const Eigen::MatrixBase< Derived > &rho, const std::vector< cmat > &Ks)`  
*Applies the channel specified by the set of Kraus operators Ks to the density matrix rho.*
- `template<typename Derived >`  
`cmat qpp::channel (const Eigen::MatrixBase< Derived > &rho, const std::vector< cmat > &Ks, const std::vector< std::size_t > &subsys, std::size_t n, std::size_t d=2)`

*Applies the channel specified by the set of Kraus operators  $K_s$  to the part of the density matrix  $\rho$  specified by  $\text{subsys}$ .*

- `cmat qpp::super (const std::vector< cmat > &Ks)`

*Superoperator matrix representation.*

- `cmat qpp::choi (const std::vector< cmat > &Ks)`

*Choi matrix representation.*

- `std::vector< cmat > qpp::choi2kraus (const cmat &A)`

*Extracts orthogonal Kraus operators from Choi matrix.*

- `template<typename Derived >  
DynMat< typename Derived::Scalar > qpp::ptrace1 (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`

*Partial trace.*

- `template<typename Derived >  
DynMat< typename Derived::Scalar > qpp::ptrace2 (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &dims)`

*Partial trace.*

- `template<typename Derived >  
DynMat< typename Derived::Scalar > qpp::ptrace (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &subsys, const std::vector< std::size_t > &dims)`

*Partial trace.*

- `template<typename Derived >  
DynMat< typename Derived::Scalar > qpp::ptranspose (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &subsys, const std::vector< std::size_t > &dims)`

*Partial transpose.*

- `template<typename Derived >  
DynMat< typename Derived::Scalar > qpp::syspermute (const Eigen::MatrixBase< Derived > &A, const std::vector< std::size_t > &perm, const std::vector< std::size_t > &dims)`

*System permutation.*

## 8.20 include/qpp.h File Reference

```
#include <algorithm>
#include <chrono>
#include <cmath>
#include <complex>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <exception>
#include <fstream>
#include <functional>
#include <initializer_list>
#include <iomanip>
#include <iostream>
#include <iterator>
#include <limits>
#include <numeric>
#include <ostream>
#include <random>
#include <sstream>
#include <stdexcept>
#include <string>
#include <tuple>
#include <type_traits>
#include <utility>
#include <vector>
#include <Eigen/Dense>
#include <Eigen/SVD>
#include "constants.h"
#include "types.h"
#include "classes/exception.h"
#include "classes/singleton.h"
#include "classes/states.h"
#include "classes/randevs.h"
#include "internal/functions.h"
#include "classes/init.h"
#include "functions.h"
#include "classes/gates.h"
#include "classes/codes.h"
#include "operations.h"
#include "entropies.h"
#include "io.h"
#include "entanglement.h"
#include "instruments.h"
#include "random.h"
#include "classes/timer.h"
#include "experimental/test.h"
#include "experimental/classes/qudit.h"
```

Include dependency graph for qpp.h:



## Namespaces

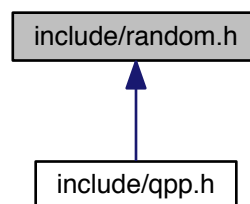
- [qpp](#)

## Variables

- const Init & [qpp::init](#) = Init::get\_instance()  
*[qpp::Init](#) const Singleton*
- const Codes & [qpp::codes](#) = Codes::get\_instance()  
*[qpp::Codes](#) const Singleton*
- const Gates & [qpp::gt](#) = Gates::get\_instance()  
*[qpp::Gates](#) const Singleton*
- const States & [qpp::st](#) = States::get\_instance()  
*[qpp::States](#) const Singleton*
- RandomDevices & [qpp::rdevs](#) = RandomDevices::get\_instance()  
*[qpp::RandomDevices](#) Singleton*

## 8.21 include/random.h File Reference

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



## Namespaces

- [qpp](#)

## Functions

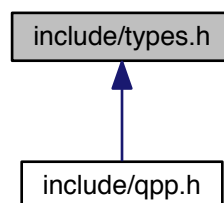
- template<typename Derived >  
Derived [qpp::rand](#) (std::size\_t rows, std::size\_t cols, double a=0, double b=1)  
*Generates a random matrix with entries uniformly distributed in the interval [a, b)*
- template<>  
dmat [qpp::rand](#) (std::size\_t rows, std::size\_t cols, double a, double b)  
*Generates a random real matrix with entries uniformly distributed in the interval [a, b), specialization for double matrices ([qpp::dmat](#))*
- template<>  
cmat [qpp::rand](#) (std::size\_t rows, std::size\_t cols, double a, double b)  
*Generates a random complex matrix with entries (both real and imaginary) uniformly distributed in the interval [a, b), specialization for complex matrices ([qpp::cmat](#))*
- double [qpp::rand](#) (double a=0, double b=1)  
*Generates a random real number uniformly distributed in the interval [a, b)*
- int [qpp::randint](#) (int a=std::numeric\_limits< int >::min(), int b=std::numeric\_limits< int >::max())

- Generates a random integer (int) uniformly distributed in the interval [a, b].*

  - `template<typename Derived >`  
`Derived qpp::randn (std::size_t rows, std::size_t cols, double mean=0, double sigma=1)`  
*Generates a random matrix with entries normally distributed in  $N(\text{mean}, \text{sigma})$*
- `template<>`  
`dmat qpp::randn (std::size_t rows, std::size_t cols, double mean, double sigma)`  
*Generates a random real matrix with entries normally distributed in  $N(\text{mean}, \text{sigma})$ , specialization for double matrices ([qpp::dmat](#))*
- `template<>`  
`cmat qpp::randn (std::size_t rows, std::size_t cols, double mean, double sigma)`  
*Generates a random complex matrix with entries (both real and imaginary) normally distributed in  $N(\text{mean}, \text{sigma})$ , specialization for complex matrices ([qpp::cmat](#))*
- `double qpp::randn (double mean=0, double sigma=1)`  
*Generates a random real number (double) normally distributed in  $N(\text{mean}, \text{sigma})$*
- `cmat qpp::randU (std::size_t D)`  
*Generates a random unitary matrix.*
- `cmat qpp::randV (std::size_t Din, std::size_t Dout)`  
*Generates a random isometry matrix.*
- `std::vector< cmat > qpp::randkraus (std::size_t N, std::size_t D)`  
*Generates a set of random Kraus operators.*
- `cmat qpp::randH (std::size_t D)`  
*Generates a random Hermitian matrix.*
- `ket qpp::randket (std::size_t D)`  
*Generates a random normalized ket (pure state vector)*
- `cmat qpp::randrho (std::size_t D)`  
*Generates a random density matrix.*
- `std::vector< std::size_t > qpp::randperm (std::size_t n)`  
*Generates a random uniformly distributed permutation.*

## 8.22 include/types.h File Reference

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



## Namespaces

- [qpp](#)

## Typedefs

- using `qpp::cplx` = `std::complex< double >`  
*Complex number in double precision.*
- template<typename Scalar >  
using `qpp::DynMat` = `Eigen::Matrix< Scalar, Eigen::Dynamic, Eigen::Dynamic >`  
*Dynamic Eigen matrix over the field specified by Scalar.*
- template<typename Scalar >  
using `qpp::DynColVect` = `Eigen::Matrix< Scalar, Eigen::Dynamic, 1 >`  
*Dynamic Eigen column vector over the field specified by Scalar.*
- template<typename Scalar >  
using `qpp::DynRowVect` = `Eigen::Matrix< Scalar, 1, Eigen::Dynamic >`  
*Dynamic Eigen row vector over the field specified by Scalar.*
- using `qpp::ket` = `DynColVect< cplx >`  
*Complex (double precision) dynamic Eigen column vector.*
- using `qpp::bra` = `DynRowVect< cplx >`  
*Complex (double precision) dynamic Eigen row vector.*
- using `qpp::cmat` = `DynMat< cplx >`  
*Complex (double precision) dynamic Eigen matrix.*
- using `qpp::dmat` = `DynMat< double >`  
*Real (double precision) dynamic Eigen matrix.*

## 8.23 mainpage.dox File Reference

# Index

- absm
  - qpp, [22](#)
- abssq
  - qpp, [23](#)
- adjoint
  - qpp, [24](#)
- anticomm
  - qpp, [24](#)
- apply
  - qpp, [25](#)
- bra
  - qpp, [21](#)
- CUSTOM\_EXCEPTION
  - qpp::Exception, [110](#)
- channel
  - qpp, [27](#), [28](#)
- choi
  - qpp, [29](#)
- choi2kraus
  - qpp, [30](#)
- chop
  - qpp, [92](#)
- cmat
  - qpp, [21](#)
- codes
  - qpp, [92](#)
- comm
  - qpp, [31](#)
- compperm
  - qpp, [31](#)
- concurrence
  - qpp, [33](#)
- conjugate
  - qpp, [33](#)
- cosm
  - qpp, [35](#)
- cplx
  - qpp, [21](#)
- cwise
  - qpp, [35](#)
- DIMS\_INVALID
  - qpp::Exception, [109](#)
- DIMS\_MISMATCH\_CVECTOR
  - qpp::Exception, [109](#)
- DIMS\_MISMATCH\_MATRIX
  - qpp::Exception, [109](#)
- DIMS\_MISMATCH\_RVECTOR
  - qpp::Exception, [109](#)
- DIMS\_MISMATCH\_VECTOR
  - qpp::Exception, [109](#)
- DIMS\_NOT\_EQUAL
  - qpp::Exception, [109](#)
- det
  - qpp, [36](#)
- disp
  - qpp, [36–38](#)
- displn
  - qpp, [39–41](#)
- dmat
  - qpp, [21](#)
- ee
  - qpp, [92](#)
- entanglement
  - qpp, [42](#)
- eps
  - qpp, [92](#)
- evals
  - qpp, [43](#)
- evects
  - qpp, [44](#)
- expm
  - qpp, [44](#)
- FIVE\_QUBIT
  - qpp::Codes, [106](#)
- funm
  - qpp, [45](#)
- gconcurrence
  - qpp, [45](#)
- grams
  - qpp, [46](#), [47](#)
- gt
  - qpp, [92](#)
- hevals
  - qpp, [48](#)
- hevects
  - qpp, [48](#)
- infty
  - qpp, [92](#)
- init
  - qpp, [93](#)
- inverse
  - qpp, [49](#)
- invperm



- qpp, 49
- ket
  - qpp, 22
- kron
  - qpp, 50, 51
- kronpow
  - qpp, 52
- load
  - qpp, 52
- logdet
  - qpp, 54
- logm
  - qpp, 54
- lognegativity
  - qpp, 55
- MATRIX\_MISMATCH\_SUBSYS
  - qpp::Exception, 109
- MATRIX\_NOT\_CVECTOR
  - qpp::Exception, 109
- MATRIX\_NOT\_RVECTOR
  - qpp::Exception, 109
- MATRIX\_NOT\_SQUARE
  - qpp::Exception, 109
- MATRIX\_NOT\_SQUARE\_OR\_CVECTOR
  - qpp::Exception, 109
- MATRIX\_NOT\_SQUARE\_OR\_RVECTOR
  - qpp::Exception, 109
- MATRIX\_NOT\_SQUARE\_OR\_VECTOR
  - qpp::Exception, 109
- MATRIX\_NOT\_VECTOR
  - qpp::Exception, 109
- maxn
  - qpp, 93
- measure
  - qpp, 56, 57
- mket
  - qpp, 58
- mprj
  - qpp, 59
- multiidx2n
  - qpp, 60
- n2multiidx
  - qpp, 60
- NINE\_QUBIT\_SHOR
  - qpp::Codes, 106
- NO\_CODEWORD
  - qpp::Exception, 110
- NOT\_BIPARTITE
  - qpp::Exception, 110
- NOT\_QUBIT\_GATE
  - qpp::Exception, 110
- NOT\_QUBIT\_SUBSYS
  - qpp::Exception, 110
- negativity
  - qpp, 61
- norm
  - qpp, 62
- OUT\_OF\_RANGE
  - qpp::Exception, 110
- omega
  - qpp, 62
- PERM\_INVALID
  - qpp::Exception, 109
- pi
  - qpp, 93
- powm
  - qpp, 63
- prj
  - qpp, 63
- prod
  - qpp, 64
- ptrace
  - qpp, 65
- ptrace1
  - qpp, 66
- ptrace2
  - qpp, 67
- ptranspose
  - qpp, 68
- qmutualinfo
  - qpp, 69
- qpp, 13
  - absm, 22
  - abssq, 23
  - adjoint, 24
  - anticomm, 24
  - apply, 25
  - bra, 21
  - channel, 27, 28
  - choi, 29
  - choi2kraus, 30
  - chop, 92
  - cmat, 21
  - codes, 92
  - comm, 31
  - compperm, 31
  - concurrence, 33
  - conjugate, 33
  - cosm, 35
  - cplx, 21
  - cwise, 35
  - det, 36
  - disp, 36–38
  - displn, 39–41
  - dmat, 21
  - ee, 92
  - entanglement, 42
  - eps, 92
  - evals, 43
  - evecs, 44
  - expm, 44

- funm, [45](#)
- gconcurrency, [45](#)
- grams, [46](#), [47](#)
- gt, [92](#)
- hevals, [48](#)
- hevects, [48](#)
- infty, [92](#)
- init, [93](#)
- inverse, [49](#)
- invperm, [49](#)
- ket, [22](#)
- kron, [50](#), [51](#)
- kronpow, [52](#)
- load, [52](#)
- logdet, [54](#)
- logm, [54](#)
- lognegativity, [55](#)
- maxn, [93](#)
- measure, [56](#), [57](#)
- mket, [58](#)
- mprj, [59](#)
- multiidx2n, [60](#)
- n2multiidx, [60](#)
- negativity, [61](#)
- norm, [62](#)
- omega, [62](#)
- pi, [93](#)
- powm, [63](#)
- prj, [63](#)
- prod, [64](#)
- ptrace, [65](#)
- ptrace1, [66](#)
- ptrace2, [67](#)
- ptranspose, [68](#)
- qmutualinfo, [69](#)
- rand, [70](#), [71](#)
- randint, [72](#)
- randket, [73](#)
- randkraus, [73](#)
- randn, [74](#), [75](#)
- randperm, [76](#)
- randrho, [76](#)
- rdevs, [93](#)
- renyi, [77](#)
- reshape, [78](#)
- save, [78](#)
- schatten, [80](#)
- schmidtcoeff, [80](#)
- schmidtprob, [81](#)
- shannon, [84](#)
- sinm, [85](#)
- spectralpowm, [85](#)
- sqrtn, [86](#)
- st, [93](#)
- sum, [86](#), [87](#)
- super, [87](#)
- svals, [88](#)
- syspermute, [89](#)
- trace, [90](#)
- transpose, [91](#)
- tsallis, [91](#)
- qpp::Codes
  - FIVE\_QUBIT, [106](#)
  - NINE\_QUBIT\_SHOR, [106](#)
  - SEVEN\_QUBIT\_STEANE, [106](#)
- qpp::Exception
  - CUSTOM\_EXCEPTION, [110](#)
  - DIMS\_INVALID, [109](#)
  - DIMS\_MISMATCH\_CVECTOR, [109](#)
  - DIMS\_MISMATCH\_MATRIX, [109](#)
  - DIMS\_MISMATCH\_RVECTOR, [109](#)
  - DIMS\_MISMATCH\_VECTOR, [109](#)
  - DIMS\_NOT\_EQUAL, [109](#)
  - MATRIX\_MISMATCH\_SUBSYS, [109](#)
  - MATRIX\_NOT\_CVECTOR, [109](#)
  - MATRIX\_NOT\_RVECTOR, [109](#)
  - MATRIX\_NOT\_SQUARE, [109](#)
  - MATRIX\_NOT\_SQUARE\_OR\_CVECTOR, [109](#)
  - MATRIX\_NOT\_SQUARE\_OR\_RVECTOR, [109](#)
  - MATRIX\_NOT\_SQUARE\_OR\_VECTOR, [109](#)
  - MATRIX\_NOT\_VECTOR, [109](#)
  - NO\_CODEWORD, [110](#)
  - NOT\_BIPARTITE, [110](#)
  - NOT\_QUBIT\_GATE, [110](#)
  - NOT\_QUBIT\_SUBSYS, [110](#)
  - OUT\_OF\_RANGE, [110](#)
  - PERM\_INVALID, [109](#)
  - SUBSYS\_MISMATCH\_DIMS, [109](#)
  - TYPE\_MISMATCH, [110](#)
  - UNDEFINED\_TYPE, [110](#)
  - UNKNOWN\_EXCEPTION, [109](#)
  - ZERO\_SIZE, [109](#)
- rand
  - qpp, [70](#), [71](#)
- randint
  - qpp, [72](#)
- randket
  - qpp, [73](#)
- randkraus
  - qpp, [73](#)
- randn
  - qpp, [74](#), [75](#)
- randperm
  - qpp, [76](#)
- randrho
  - qpp, [76](#)
- rdevs
  - qpp, [93](#)
- renyi
  - qpp, [77](#)
- reshape
  - qpp, [78](#)
- SEVEN\_QUBIT\_STEANE
  - qpp::Codes, [106](#)
- SUBSYS\_MISMATCH\_DIMS

- qpp::Exception, [109](#)
- save
  - qpp, [78](#)
- schatten
  - qpp, [80](#)
- schmidtcoeff
  - qpp, [80](#)
- schmidtprob
  - qpp, [81](#)
- shannon
  - qpp, [84](#)
- sinm
  - qpp, [85](#)
- spectralpowm
  - qpp, [85](#)
- sqrtn
  - qpp, [86](#)
- st
  - qpp, [93](#)
- sum
  - qpp, [86](#), [87](#)
- super
  - qpp, [87](#)
- svals
  - qpp, [88](#)
- syspermute
  - qpp, [89](#)
- TYPE\_MISMATCH
  - qpp::Exception, [110](#)
- trace
  - qpp, [90](#)
- transpose
  - qpp, [91](#)
- tsallis
  - qpp, [91](#)
- UNDEFINED\_TYPE
  - qpp::Exception, [110](#)
- UNKNOWN\_EXCEPTION
  - qpp::Exception, [109](#)
- ZERO\_SIZE
  - qpp::Exception, [109](#)