# Quantum++

v0.8.8.2

Generated by Doxygen 1.8.10

Mon Oct 3 2016 14:34:41

# Contents

# Chapter 1

# Quantum++

### Version 0.8.8.2 - 3 October 2016

Quantum++ is a C++11 general purpose quantum computing library, composed solely of template header files. Quantum++ is written in standard C++11 and has very low external dependencies, using only the Eigen 3 linear algebra header-only template library and, if available, the OpenMP multi-processing library.

Quantum++ is not restricted to qubit systems or specific quantum information processing tasks, being capable of simulating arbitrary quantum processes. The main design factors taken in consideration were the ease of use, high portability, and high performance.

If you are interesting in contributing, please contact me. To contribute, you need to have a good knowledge of C++ (preferably C++11), including templates and the standard library, a basic knowledge of quantum computing and linear algebra, and working experience with Eigen 3.

For additional Eigen 3 documentation see http://eigen.tuxfamily.org/dox/. For a simple Eigen 3 quick ASCII reference see http://eigen.tuxfamily.org/dox/AsciiQuickReference.txt.



### Building instructions

#### Configuration:

- Compiler: g++ version 4.8.2 or later (for good C++11 support)
- Eigen 3 library located in `$HOME/eigen`
- Quantum++ library located in `$HOME/qpp`
- MATLAB compiler include header files: `/Applications/MATLAB_R2016a.app/extern/include`
- MATLAB compiler shared library files: `/Applications/MATLAB_R2016a.app/bin/maci64`

#### Building without a build system

- Example file: `$HOME/qpp/examples/minimal.cpp`
- Output executable: `$HOME/qpp/examples/minimal`
- Must run the commands below from inside the directory `$HOME/qpp/examples`

**Release version (without MATLAB support):**

```
g++ -pedantic -std=c++11 -Wall -Wextra -Weffc++ -fopenmp \
     -O3 -DNDEBUG -DEIGEN_NO_DEBUG \
     -isystem $HOME/eigen -I $HOME/qpp/include \
     minimal.cpp -o minimal
```

**Debug version (without MATLAB support):**

```
g++ -pedantic -std=c++11 -Wall -Wextra -Weffc++ -fopenmp \
     -g3 -DDEBUG \
     -isystem $HOME/eigen -I $HOME/qpp/include \
      minimal.cpp -o minimal
```

**Release version (with MATLAB support):**

```
g++ -pedantic -std=c++11 -Wall -Wextra -Weffc++ -fopenmp \
     -O3 -DNDEBUG -DEIGEN_NO_DEBUG \
     -isystem $HOME/eigen -I $HOME/qpp/include \
     -I/Applications/MATLAB_R2016a.app/extern/include \
     -L/Applications/MATLAB_R2016a.app/bin/maci64 \
     -lmx -lmat minimal.cpp -o minimal
```

**Debug version (with MATLAB support):**

```
g++ -pedantic -std=c++11 -Wall -Wextra -Weffc++ -fopenmp \
     -g3 -DDEBUG \
     -isystem $HOME/eigen -I $HOME/qpp/include \
     -I /Applications/MATLAB_R2016a.app/extern/include \
     -L /Applications/MATLAB_R2016a.app/bin/maci64 \
     -lmx -lmat minimal.cpp -o minimal
```

#### Building using cmake (version 3.0 or later)

The current version of the repository has a `./CMakeLists.txt` configuration file for building examples using `cmake`. To build an example using `cmake`, I recommend an out-of-source build, i.e., from the root of the project (where `./include` is located), type

```
mkdir ./build
cd ./build
cmake ..
make
```

The above commands build the relase version (default) executable `qpp`, from the source file `./examples/minimal.cpp`, without MATLAB support (default), inside the directory `./build`. To build a different configuration, e.g. debug version with MATLAB support, type from the root of the project

```
cd ./build
rm -rf *
cmake -DCMAKE_BUILD_TYPE=Debug -DWITH_MATLAB=ON ..
make
```

Or, to disable OpenMP support (enabled by default), type

```
cd ./build
rm -rf *
cmake -DWITH_OPENMP=OFF ..
make
```

To change the name of the example file, the location of the Eigen 3 library or the location of MATLAB installation, edit the ./CMakeLists.txt file. See also ./CMakeLists.txt for additional options. Do not forget to remove everything from the ./build directory before a fresh build!

#### Additional remarks

- The C++ compiler must be C++11 compliant.
- If using Windows, I recommend compiling under cygwin via cmake and g++. See also http://stackoverflow.com/questions/28997206/cygwin-support-for-c11-in-g4-9-2 for a bug related to lack of support for some C++11 math functions, and how to fix it. Quick fix: patch the standard library header file <cmath> using the provided patch ./cmath_cygwin.patch.
- If your compiler does not support OpenMP (as it is the case e.g with clang++), disable OpenMP in your build, as otherwise the linker may not find the gomp library.
- If you run the program on OS X with MATLAB support, make sure that the environment variable DYLD_LIBRARY_PATH is set to point to the MATLAB compiler library location, see the run_OSX_MATLAB script. Otherwise, you will get a runtime error like dyld: Library not loaded: @rpath/libmat.dylib.

```
* I recommend running via a script, as otherwise setting the
'DYLD_LIBRARY_PATH' globally may interfere with
[macports](https://www.macports.org/)' [cmake](http://www.cmake.org/)
installation (in case you use [cmake](http://www.cmake.org/) from
[macports](https://www.macports.org/)). If you use a script,
then the environment variable is local to the script and
does not interfere with the rest of the system.

* Example of script, assumed to be located in the root directory
of Quantum++:

        #!/bin/sh

        MATLAB=/Applications/MATLAB_R2016a.app
        export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$MATLAB/bin/maci64

        ./build/qpp
```

- If you build a debug version with g++ under OS X and use gdb to step inside template functions you may want to add -fno-weak compiler flag. See http://stackoverflow.com/questions/23330641/gnu-gdb-can-not-step-into-template-functions-os-x-mavericks for more details about this problem.

# Chapter 2

# Namespace Index

## 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 3

# Hierarchical Index

## 3.1 Class Hierarchy

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

# Chapter 4

# Class Index

## 4.1 Class List

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

# Chapter 5

# File Index

## 5.1 File List

Here is a list of all files with brief descriptions:

# Chapter 6

# Namespace Documentation

## 6.1 qpp Namespace Reference

Quantum++ main namespace.

### Namespaces

- experimental

  *Experimental/test functions/classes, do not use or modify.*
- internal

  *Internal utility functions, do not use/modify.*

### 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 IDisplay

  *Abstract class (interface) that mandates the definition of virtual std::ostream& display(std::ostream& os) const.*
- class Init

  *const Singleton class that performs additional initializations/cleanups*
- struct is_complex

  *Checks whether the type is a complex type.*
- struct is_complex< std::complex< T > >

  *Checks whether the type is a complex number type, specialization for complex types.*
- struct is_iterable

  *Checks whether T is compatible with an STL-like iterable container.*
- struct is_iterable< T, to_void< decltype(std::declval< T >().begin()), decltype(std::declval< T >().end()), typename T::value_type > >

  *Checks whether T is compatible with an STL-like iterable container, specialization for STL-like iterable containers.*
- struct is_matrix_expression

  *Checks whether the type is an Eigen matrix expression.*
- struct is_matrix_expression< typename Eigen::MatrixBase< Derived > >

  *Checks whether the type is an Eigen matrix expression, specialization for Eigen matrix expressions.*

- class RandomDevices

  *Singeleton class that manages the source of randomness in the library.*
- class States

  *const Singleton class that implements most commonly used states*
- class Timer

  *Chronometer.*

## Typedefs

- template<typename... >
  using to_void = void

  *Alias template that implements the proposal for void_t.*
- using idx = std::size_t

  *Non-negative integer index.*
- using bigint = long long int

  *Big integer.*
- using ubigint = unsigned long long int

  *Non-negative big integer.*
- using cplx = std::complex< double >

  *Complex number in double precision.*
- using ket = Eigen::VectorXcd

  *Complex (double precision) dynamic Eigen column vector.*
- using bra = Eigen::RowVectorXcd

  *Complex (double precision) dynamic Eigen row vector.*
- using cmat = Eigen::MatrixXcd

  *Complex (double precision) dynamic Eigen matrix.*
- using dmat = Eigen::MatrixXd

  *Real (double precision) dynamic Eigen matrix.*
- template<typename Scalar >
  using dyn_mat = Eigen::Matrix< Scalar, Eigen::Dynamic, Eigen::Dynamic >

  *Dynamic Eigen matrix over the field specified by Scalar.*
- template<typename Scalar >
  using dyn_col_vect = Eigen::Matrix< Scalar, Eigen::Dynamic, 1 >

  *Dynamic Eigen column vector over the field specified by Scalar.*
- template<typename Scalar >
  using dyn_row_vect = Eigen::Matrix< Scalar, 1, Eigen::Dynamic >

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

## Functions

- constexpr cplx operator""_i (unsigned long long int x) noexcept

  *User-defined literal for complex $i = \sqrt{-1}$ (integer overload)*
- constexpr cplx operator""_i (long double x) noexcept

  *User-defined literal for complex $i = \sqrt{-1}$ (real overload)*
- cplx omega (idx D)

  *D-th root of unity.*
- template<typename Derived >
  dyn_col_vect< double > schmidtcoeffs (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &dims)

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

- template<typename Derived >
  cmat schmidtA (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &dims)

  *Schmidt basis on Alice side.*
- template<typename Derived >
  cmat schmidtB (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &dims)

  *Schmidt basis on Bob side.*
- template<typename Derived >
  std::vector< double > schmidtprobs (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &dims)

  *Schmidt probabilities of the bi-partite pure state A.*
- template<typename Derived >
  double entanglement (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &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< idx > &dims)

  *Negativity of the bi-partite mixed state A.*
- template<typename Derived >
  double lognegativity (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &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 entropy (const Eigen::MatrixBase< Derived > &A)

  *von-Neumann entropy of the density matrix A*
- double entropy (const std::vector< double > &prob)

  *Shannon entropy of the probability distribution prob.*
- template<typename Derived >
  double renyi (const Eigen::MatrixBase< Derived > &A, double alpha)

  *Renyi- $\alpha$ entropy of the density matrix A, for $\alpha \geq 0$.*
- double renyi (const std::vector< double > &prob, double alpha)

  *Renyi- $\alpha$ entropy of the probability distribution prob, for $\alpha \geq 0$.*
- template<typename Derived >
  double tsallis (const Eigen::MatrixBase< Derived > &A, double q)

  *Tsallis- $q$ entropy of the density matrix A, for $q \geq 0$.*
- double tsallis (const std::vector< double > &prob, double q)

  *Tsallis- $q$ entropy of the probability distribution prob, for $q \geq 0$.*
- template<typename Derived >
  double qmutualinfo (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &subsysA, const std::vector< idx > &subsysB, const std::vector< idx > &dims)

  *Quantum mutual information between 2 subsystems of a composite system.*
- template<typename Derived >
  double qmutualinfo (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &subsysA, const std::vector< idx > &subsysB, idx d=2)

  *Quantum mutual information between 2 subsystems of a composite system.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > transpose (const Eigen::MatrixBase< Derived > &A)

  *Transpose.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > conjugate (const Eigen::MatrixBase< Derived > &A)

*Complex conjugate.*

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

  *Adjoint.*
- template<typename Derived >
  dyn_mat< 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 >
  std::pair< dyn_col_vect< cplx >, cmat > eig (const Eigen::MatrixBase< Derived > &A)

  *Full eigen decomposition.*
- template<typename Derived >
  dyn_col_vect< cplx > evals (const Eigen::MatrixBase< Derived > &A)

  *Eigenvalues.*
- template<typename Derived >
  cmat evects (const Eigen::MatrixBase< Derived > &A)

  *Eigenvectors.*
- template<typename Derived >
  std::pair< dyn_col_vect< double >, cmat > heig (const Eigen::MatrixBase< Derived > &A)

  *Full eigen decomposition of Hermitian expression.*
- template<typename Derived >
  dyn_col_vect< double > hevals (const Eigen::MatrixBase< Derived > &A)

  *Hermitian eigenvalues.*
- template<typename Derived >
  cmat hevects (const Eigen::MatrixBase< Derived > &A)

  *Hermitian eigenvectors.*
- template<typename Derived >
  std::tuple< cmat, dyn_col_vect< double >, cmat > svd (const Eigen::MatrixBase< Derived > &A)

  *Full singular value decomposition.*
- template<typename Derived >
  dyn_col_vect< 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 >
  dyn_mat< typename Derived::Scalar > powm (const Eigen::MatrixBase< Derived > &A, idx n)

  *Matrix power.*
- template<typename Derived >
  double schatten (const Eigen::MatrixBase< Derived > &A, double p)

  *Schatten matrix norm.*
- template<typename OutputScalar , typename Derived >
  dyn_mat< OutputScalar > cwise (const Eigen::MatrixBase< Derived > &A, OutputScalar(*f)( const typename Derived::Scalar &))

  *Functor.*
- template<typename T >
  dyn_mat< typename T::Scalar > kron (const T &head)

  *Kronecker product.*
- template<typename T , typename... Args>
  dyn_mat< typename T::Scalar > kron (const T &head, const Args &...tail)

  *Kronecker product.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > kron (const std::vector< Derived > &As)

  *Kronecker product.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > kron (const std::initializer_list< Derived > &As)

  *Kronecker product.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > kronpow (const Eigen::MatrixBase< Derived > &A, idx n)

  *Kronecker power.*

- template<typename T >
  dyn_mat< typename T::Scalar > dirsum (const T &head)

  *Direct sum.*
- template<typename T , typename... Args>
  dyn_mat< typename T::Scalar > dirsum (const T &head, const Args &...tail)

  *Direct sum.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > dirsum (const std::vector< Derived > &As)

  *Direct sum.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > dirsum (const std::initializer_list< Derived > &As)

  *Direct sum.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > dirsumpow (const Eigen::MatrixBase< Derived > &A, idx n)

  *Direct sum power.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > reshape (const Eigen::MatrixBase< Derived > &A, idx rows, idx cols)

  *Reshape.*
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > comm (const Eigen::MatrixBase< Derived1 > &A, const Eigen::↩
  MatrixBase< Derived2 > &B)

  *Commutator.*
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > anticomm (const Eigen::MatrixBase< Derived1 > &A, const Eigen::MatrixBase< Derived2 > &B)

  *Anti-commutator.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > prj (const Eigen::MatrixBase< Derived > &V)

  *Projector.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > grams (const std::vector< Derived > &Vs)

  *Gram-Schmidt orthogonalization.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > grams (const std::initializer_list< Derived > &Vs)

  *Gram-Schmidt orthogonalization.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > grams (const Eigen::MatrixBase< Derived > &A)

  *Gram-Schmidt orthogonalization.*
- std::vector< idx > n2multiidx (idx n, const std::vector< idx > &dims)

  *Non-negative integer index to multi-index.*
- idx multiidx2n (const std::vector< idx > &midx, const std::vector< idx > &dims)

  *Multi-index to non-negative integer index.*
- ket mket (const std::vector< idx > &mask, const std::vector< idx > &dims)

  *Multi-partite qudit ket.*
- ket mket (const std::vector< idx > &mask, idx d=2)

  *Multi-partite qudit ket.*
- cmat mprj (const std::vector< idx > &mask, const std::vector< idx > &dims)

  *Projector onto multi-partite qudit ket.*
- cmat mprj (const std::vector< idx > &mask, idx d=2)

  *Projector onto multi-partite qudit ket.*
- template<typename InputIterator >
  std::vector< double > abssq (InputIterator first, InputIterator last)

*Computes the absolute values squared of an STL-like range of complex numbers.*

- template<typename Container >
  std::vector< double > abssq (const Container &c, typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Computes the absolute values squared of an STL-like container.*
- template<typename Derived >
  std::vector< double > abssq (const Eigen::MatrixBase< Derived > &A)

  *Computes the absolute values squared of an Eigen expression.*
- template<typename InputIterator >
  std::iterator_traits< InputIterator >::value_type sum (InputIterator first, InputIterator last)

  *Element-wise sum of an STL-like range.*
- template<typename Container >
  Container::value_type sum (const Container &c, typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Element-wise sum of the elements of an STL-like container.*
- template<typename InputIterator >
  std::iterator_traits< InputIterator >::value_type prod (InputIterator first, InputIterator last)

  *Element-wise product of an STL-like range.*
- template<typename Container >
  Container::value_type prod (const Container &c, typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Element-wise product of the elements of an STL-like container.*
- template<typename Derived >
  dyn_col_vect< typename Derived::Scalar > rho2pure (const Eigen::MatrixBase< Derived > &A)

  *Finds the pure state representation of a matrix proportional to a projector onto a pure state.*
- template<typename T >
  std::vector< T > complement (std::vector< T > subsys, idx N)

  *Constructs the complement of a subsystem vector.*
- template<typename Derived >
  std::vector< double > rho2bloch (const Eigen::MatrixBase< Derived > &A)

  *Computes the 3-dimensional real Bloch vector corresponding to the qubit density matrix A.*
- cmat bloch2rho (const std::vector< double > &r)

  *Computes the density matrix corresponding to the 3-dimensional real Bloch vector r.*
- template<typename Derived >
  internal::IOManipEigen disp (const Eigen::MatrixBase< Derived > &A, double chop=qpp::chop)

  *Eigen expression ostream manipulator.*
- internal::IOManipEigen disp (cplx z, double chop=qpp::chop)

  *Complex number ostream manipulator.*
- template<typename InputIterator >
  internal::IOManipRange< InputIterator > disp (InputIterator first, InputIterator last, const std::string &separator, const std::string &start="[", const std::string &end="]")

  *Range ostream manipulator.*
- template<typename Container >
  internal::IOManipRange< typename Container::const_iterator > disp (const Container &c, const std::string &separator, const std::string &start="[", const std::string &end="]", typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Standard container ostream manipulator. The container must support std::begin(), std::end() and forward iteration.*
- template<typename PointerType >
  internal::IOManipPointer< PointerType > disp (const PointerType *p, idx n, const std::string &separator, const std::string &start="[", const std::string &end="]")

  *C-style pointer ostream manipulator.*
- 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 >
  dyn_mat< typename Derived::Scalar > load (const std::string &fname)

  *Loads Eigen matrix from a binary file (internal format) in double precision.*
- template<typename Derived >
  dyn_col_vect< typename Derived::Scalar > ip (const Eigen::MatrixBase< Derived > &phi, const Eigen::↩ MatrixBase< Derived > &psi, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Generalized inner product.*
- template<typename Derived >
  dyn_col_vect< typename Derived::Scalar > ip (const Eigen::MatrixBase< Derived > &phi, const Eigen::↩ MatrixBase< Derived > &psi, const std::vector< idx > &subsys, idx d=2)

  *Generalized inner product.*
- template<typename Derived >
  std::tuple< idx, 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::tuple< idx, 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.*
- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > measure (const Eigen::MatrixBase< Derived > &A, const cmat &U)

  *Measures the state A in the orthonormal basis specified by the unitary matrix U.*
- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > measure (const Eigen::MatrixBase< Derived > &A, const std::vector< cmat > &Ks, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Measures the part subsys of the multi-partite state vector or density matrix A using the set of Kraus operators Ks.*
- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > measure (const Eigen::MatrixBase< Derived > &A, const std::initializer_list< cmat > &Ks, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Measures the part subsys of the multi-partite state vector or density matrix A using the set of Kraus operators Ks.*
- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > measure (const Eigen::MatrixBase< Derived > &A, const std::vector< cmat > &Ks, const std::vector< idx > &subsys, idx d=2)

  *Measures the part subsys of the multi-partite state vector or density matrix A using the set of Kraus operators Ks.*
- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > measure (const Eigen::MatrixBase< Derived > &A, const std::initializer_list< cmat > &Ks, const std::vector< idx > &subsys, idx d=2)

  *Measures the part subsys of the multi-partite state vector or density matrix A using the set of Kraus operators Ks.*
- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > measure (const Eigen::MatrixBase< Derived > &A, const cmat &V, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Measures the part subsys of the multi-partite state vector or density matrix A in the orthonormal basis or rank-1 POVM specified by the matrix V.*
- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > measure (const Eigen::MatrixBase< Derived > &A, const cmat &V, const std::vector< idx > &subsys, idx d=2)

  *Measures the part subsys of the multi-partite state vector or density matrix A in the orthonormal basis or rank-1 POVM specified by the matrix V.*
- template<typename Derived >
  std::tuple< std::vector< idx >, double, cmat > measure_seq (const Eigen::MatrixBase< Derived > &A, std::vector< idx > subsys, std::vector< idx > dims)

  *Sequentially measures the part subsys of the multi-partite state vector or density matrix A in the computational basis.*

- template<typename Derived >
  std::tuple< std::vector< idx >, double, cmat > measure_seq (const Eigen::MatrixBase< Derived > &A, std::vector< idx > subsys, idx d=2)

  *Sequentially measures the part subsys of the multi-partite state vector or density matrix A in the computational basis.*
- template<typename Derived >
  Derived loadMATLABmatrix (const std::string &, const std::string &)

  *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 > &, const std::string &, const std::string &, const std::string &)

  *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)*
- std::vector< int > x2contfrac (double x, idx n, idx cut=1e5)

  *Simple continued fraction expansion.*
- double contfrac2x (const std::vector< int > &cf, idx n)

  *Real representation of a simple continued fraction.*
- double contfrac2x (const std::vector< int > &cf)

  *Real representation of a simple continued fraction.*
- bigint gcd (bigint m, bigint n)

  *Greatest common divisor of two integers.*
- bigint gcd (const std::vector< bigint > &ns)

  *Greatest common divisor of a list of integers.*
- bigint lcm (bigint m, bigint n)

  *Least common multiple of two integers.*
- bigint lcm (const std::vector< bigint > &ns)

  *Least common multiple of a list of integers.*
- std::vector< idx > invperm (const std::vector< idx > &perm)

  *Inverse permutation.*
- std::vector< idx > compperm (const std::vector< idx > &perm, const std::vector< idx > &sigma)

  *Compose permutations.*
- std::vector< bigint > factors (bigint n)

  *Prime factor decomposition.*
- bool isprime (bigint n)

  *Primality test.*
- bigint modpow (bigint a, bigint n, bigint p)

  *Fast integer power modulo p based on the SQUARE-AND-MULTIPLY algorithm.*
- std::tuple< bigint, bigint, bigint > egcd (bigint m, bigint n)

  *Extended greatest common divisor of two integers.*
- bigint modinv (bigint a, bigint p)

*Modular inverse of a mod p.*

- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > applyCTRL (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< idx > &ctrl, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Applies the controlled-gate A to the part subsys of the multi-partite state vector or density matrix state.*
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > applyCTRL (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< idx > &ctrl, const std::vector< idx > &subsys, idx d=2)

  *Applies the controlled-gate A to the part subsys of the multi-partite state vector or density matrix state.*
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > apply (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Applies the gate A to the part subsys of the multi-partite state vector or density matrix state.*
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > apply (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< idx > &subsys, idx d=2)

  *Applies the gate A to the part subsys of the multi-partite state vector or density matrix state.*
- template<typename Derived >
  cmat apply (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 apply (const Eigen::MatrixBase< Derived > &rho, const std::vector< cmat > &Ks, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Applies the channel specified by the set of Kraus operators Ks to the part subsys of the multi-partite density matrix rho.*
- template<typename Derived >
  cmat apply (const Eigen::MatrixBase< Derived > &rho, const std::vector< cmat > &Ks, const std::vector< idx > &subsys, idx d=2)

  *Applies the channel specified by the set of Kraus operators Ks to the part subsys of the multi-partite density matrix rho.*
- cmat kraus2super (const std::vector< cmat > &Ks)

  *Superoperator matrix.*
- cmat kraus2choi (const std::vector< cmat > &Ks)

  *Choi matrix.*
- std::vector< cmat > choi2kraus (const cmat &A)

  *Orthogonal Kraus operators from Choi matrix.*
- cmat choi2super (const cmat &A)

  *Converts Choi matrix to superoperator matrix.*
- cmat super2choi (const cmat &A)

  *Converts superoperator matrix to Choi matrix.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > ptrace1 (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &dims)

  *Partial trace.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > ptrace2 (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &dims)

  *Partial trace.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > ptrace (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Partial trace.*

- template<typename Derived >
  dyn_mat< typename Derived::Scalar > ptrace (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &subsys, idx d=2)

  *Partial trace.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > ptranspose (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Partial transpose.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > ptranspose (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &subsys, idx d=2)

  *Partial transpose.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > syspermute (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &perm, const std::vector< idx > &dims)

  *Subsystem permutation.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > syspermute (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &perm, idx d=2)

  *Subsystem permutation.*
- double rand (double a=0, double b=1)

  *Generates a random real number uniformly distributed in the interval [a, b)*
- bigint rand (bigint a=std::numeric_limits< bigint >::min(), bigint b=std::numeric_limits< bigint >::max())

  *Generates a random big integer uniformly distributed in the interval [a, b].*
- ubigint rand (ubigint a=std::numeric_limits< ubigint >::min(), ubigint b=std::numeric_limits< ubigint >::max())

  *Generates a random non-negative big integer uniformly distributed in the interval [a, b].*
- idx randidx (idx a=std::numeric_limits< idx >::min(), idx b=std::numeric_limits< idx >::max())

  *Generates a random index (idx) uniformly distributed in the interval [a, b].*
- template<typename Derived >
  Derived rand (idx rows, idx cols, double a=0, double b=1)

  *Generates a random matrix with entries uniformly distributed in the interval [a, b)*
- template<>
  dmat rand (idx rows, idx 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 (idx rows, idx 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)*
- template<typename Derived >
  Derived randn (idx rows, idx cols, double mean=0, double sigma=1)

  *Generates a random matrix with entries normally distributed in N(mean, sigma)*
- template<>
  dmat randn (idx rows, idx cols, double mean, double sigma)

  *Generates a random real matrix with entries normally distributed in N(mean, sigma), specialization for double matrices (qpp::dmat)*
- template<>
  cmat randn (idx rows, idx cols, double mean, double sigma)

  *Generates a random complex matrix with entries (both real and imaginary) normally distributed in N(mean, 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(mean, sigma)*
- cmat randU (idx D)

*Generates a random unitary matrix.*

- cmat randV (idx Din, idx Dout)

  *Generates a random isometry matrix.*
- std::vector< cmat > randkraus (idx N, idx D)

  *Generates a set of random Kraus operators.*
- cmat randH (idx D)

  *Generates a random Hermitian matrix.*
- ket randket (idx D)

  *Generates a random normalized ket (pure state vector)*
- cmat randrho (idx D)

  *Generates a random density matrix.*
- std::vector< idx > randperm (idx n)

  *Generates a random uniformly distributed permutation.*
- std::vector< double > uniform (idx N)

  *Uniform probability distribution vector.*
- std::vector< double > marginalX (const dmat &probXY)

  *Marginal distribution.*
- std::vector< double > marginalY (const dmat &probXY)

  *Marginal distribution.*
- template<typename Container >
  double avg (const std::vector< double > &prob, const Container &X, typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Average.*
- template<typename Container >
  double cov (const dmat &probXY, const Container &X, const Container &Y, typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Covariance.*
- template<typename Container >
  double var (const std::vector< double > &prob, const Container &X, typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Variance.*
- template<typename Container >
  double sigma (const std::vector< double > &prob, const Container &X, typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Standard deviation.*
- template<typename Container >
  double cor (const dmat &probXY, const Container &X, const Container &Y, typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Correlation.*

## Variables

- constexpr double chop = 1e-10

  *Used in qpp::disp() for setting to zero numbers that have their absolute value smaller than qpp::chop.*
- constexpr double eps = 1e-12

  *Used to decide whether a number or expression in double precision is zero or not.*
- constexpr idx maxn = 64

  *Maximum number of allowed qu(d)its (subsystems)*
- constexpr double pi = 3.14159265358979323846264338327950288 4

  $\pi$
- constexpr double ee = 2.71828182845904523536028747135266249 7

  *Base of natural logarithm,* $e$.
- constexpr double infty = std::numeric_limits<double>::infinity()

  *Used to denote infinity in double precision.*

### 6.1.1 Detailed Description

Quantum++ main namespace.

### 6.1.2 Typedef Documentation

#### 6.1.2.1 using qpp::bigint = typedef long long int

Big integer.

#### 6.1.2.2 using qpp::bra = typedef Eigen::RowVectorXcd

Complex (double precision) dynamic Eigen row vector.

#### 6.1.2.3 using qpp::cmat = typedef Eigen::MatrixXcd

Complex (double precision) dynamic Eigen matrix.

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

Complex number in double precision.

#### 6.1.2.5 using qpp::dmat = typedef Eigen::MatrixXd

Real (double precision) dynamic Eigen matrix.

#### 6.1.2.6 template<typename Scalar > using qpp::dyn_col_vect = typedef Eigen::Matrix<Scalar, Eigen::Dynamic, 1>

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

Example:

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

#### 6.1.2.7 template<typename Scalar > using qpp::dyn_mat = typedef Eigen::Matrix<Scalar, Eigen::Dynamic, Eigen::Dynamic>

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

Example:

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

#### 6.1.2.8 template<typename Scalar > using qpp::dyn_row_vect = typedef Eigen::Matrix<Scalar, 1, Eigen::Dynamic>

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

Example:

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

#### 6.1.2.9 using qpp::idx = typedef std::size_t

Non-negative integer index.

#### 6.1.2.10 using qpp::ket = typedef Eigen::VectorXcd

Complex (double precision) dynamic Eigen column vector.

#### 6.1.2.11 template<typename... > using qpp::to_void = typedef void

Alias template that implements the proposal for void_t.

**See also**

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n3911

#### 6.1.2.12 using qpp::ubigint = typedef unsigned long long int

Non-negative big integer.

### 6.1.3 Function Documentation

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

Matrix absolut value.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Matrix absolut value of *A*

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

Computes the absolute values squared of an STL-like range of complex numbers.

**Parameters**

| | |
|---|---|
| *first* | Iterator to the first element of the range |
| *last* | Iterator to the last element of the range |

**Returns**

Real vector consisting of the range absolut values squared

#### 6.1.3.3 template<typename Container > std::vector<double> qpp::abssq ( const Container & *c*, typename std::enable_if< is_iterable< Container >::value >::type * = nullptr )

Computes the absolute values squared of an STL-like container.

**Parameters**

| | |
|---|---|
| *c* | STL-like container |

**Returns**

Real vector consisting of the container's absolut values squared

#### 6.1.3.4 template<typename Derived > std::vector<double> qpp::abssq ( const Eigen::MatrixBase< Derived > & *A* )

Computes the absolute values squared of an Eigen expression.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Real vector consisting of the absolut values squared

#### 6.1.3.5 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::adjoint ( const Eigen::MatrixBase< Derived > & *A* )

Adjoint.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

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

#### 6.1.3.6 template<typename Derived1 , typename Derived2 > dyn_mat<typename Derived1::Scalar> qpp::anticomm ( const Eigen::MatrixBase< Derived1 > & *A*, const Eigen::MatrixBase< Derived2 > & *B* )

Anti-commutator.

**See also**

qpp::comm()

Anti-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**

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

#### 6.1.3.7 template<typename Derived1 , typename Derived2 > dyn_mat<typename Derived1::Scalar> qpp::apply ( const Eigen::MatrixBase< Derived1 > & *state*, const Eigen::MatrixBase< Derived2 > & *A*, const std::vector< idx > & *subsys*, const std::vector< idx > & *dims* )

Applies the gate *A* to the part *subsys* of the multi-partite state vector or density matrix *state*.

**Note**

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

**Parameters**

| | |
|---|---|
| *state* | Eigen expression |
| *A* | Eigen expression |
| *subsys* | Subsystem indexes where the gate *A* is applied |
| *dims* | Dimensions of the multi-partite system |

**Returns**

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

#### 6.1.3.8 template<typename Derived1 , typename Derived2 > dyn_mat<typename Derived1::Scalar> qpp::apply ( const Eigen::MatrixBase< Derived1 > & *state*, const Eigen::MatrixBase< Derived2 > & *A*, const std::vector< idx > & *subsys*, idx *d* = 2 )

Applies the gate *A* to the part *subsys* of the multi-partite state vector or density matrix *state*.

**Note**

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

**Parameters**

| | |
|---|---|
| *state* | Eigen expression |
| *A* | Eigen expression |
| *subsys* | Subsystem indexes where the gate *A* is applied |
| *d* | Subsystem dimensions |

**Returns**

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

#### 6.1.3.9 template<typename Derived > cmat qpp::apply ( 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**

| | |
|---|---|
| *rho* | Eigen expression |
| *Ks* | Set of Kraus operators |

**Returns**

Output density matrix after the action of the channel

#### 6.1.3.10 template<typename Derived > cmat qpp::apply ( const Eigen::MatrixBase< Derived > & *rho*, const std::vector< cmat > & *Ks*, const std::vector< idx > & *subsys*, const std::vector< idx > & *dims* )

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

**Parameters**

| | |
|---|---|
| *rho* | Eigen expression |
| *Ks* | Set of Kraus operators |
| *subsys* | Subsystem indexes where the Kraus operators *Ks* are applied |
| *dims* | Dimensions of the multi-partite system |

**Returns**

Output density matrix after the action of the channel

#### 6.1.3.11 template<typename Derived > cmat qpp::apply ( const Eigen::MatrixBase< Derived > & *rho*, const std::vector< cmat > & *Ks*, const std::vector< idx > & *subsys*, idx *d* = 2 )

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

**Parameters**

| | |
|---|---|
| *rho* | Eigen expression |
| *Ks* | Set of Kraus operators |
| *subsys* | Subsystem indexes where the Kraus operators *Ks* are applied |
| *d* | Subsystem dimensions |

**Returns**

Output density matrix after the action of the channel

#### 6.1.3.12 template<typename Derived1 , typename Derived2 > dyn_mat<typename Derived1::Scalar> qpp::applyCTRL ( const Eigen::MatrixBase< Derived1 > & *state*, const Eigen::MatrixBase< Derived2 > & *A*, const std::vector< idx > & *ctrl*, const std::vector< idx > & *subsys*, const std::vector< idx > & *dims* )

Applies the controlled-gate *A* to the part *subsys* of the multi-partite state vector or density matrix *state*.

**See also**

qpp::Gates::CTRL()

**Note**

The dimension of the gate *A* must match the dimension of *subsys*. Also, all control subsystems in *ctrl* must have the same dimension.

**Parameters**

| | |
|---|---|
| *state* | Eigen expression |
| *A* | Eigen expression |
| *ctrl* | Control subsystem indexes |
| *subsys* | Subsystem indexes where the gate *A* is applied |
| *dims* | Dimensions of the multi-partite system |

**Returns**

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

#### 6.1.3.13 template<typename Derived1 , typename Derived2 > dyn_mat<typename Derived1::Scalar> qpp::applyCTRL ( const Eigen::MatrixBase< Derived1 > & *state*, const Eigen::MatrixBase< Derived2 > & *A*, const std::vector< idx > & *ctrl*, const std::vector< idx > & *subsys*, idx *d* = 2 )

Applies the controlled-gate *A* to the part *subsys* of the multi-partite state vector or density matrix *state*.

**See also**

qpp::Gates::CTRL()

**Note**

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

**Parameters**

| | |
|---|---|
| *state* | Eigen expression |
| *A* | Eigen expression |
| *ctrl* | Control subsystem indexes |
| *subsys* | Subsystem indexes where the gate *A* is applied |
| *d* | Subsystem dimensions |

**Returns**

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

#### 6.1.3.14 template<typename Container > double qpp::avg ( const std::vector< double > & *prob*, const Container & *X*, typename std::enable_if< is_iterable< Container >::value >::type * = nullptr )

Average.

**Parameters**

| | |
|---|---|
| *prob* | Real probability vector representing the probability distribution of *X* |
| *X* | Random variable values represented by an STL-like container |

**Returns**

Average of *X*

#### 6.1.3.15 cmat qpp::bloch2rho ( const std::vector< double > & *r* ) [inline]

Computes the density matrix corresponding to the 3-dimensional real Bloch vector *r*.

**See also**

qpp::rho2bloch()

**Parameters**

| | |
|---|---|
| *r* | 3-dimensional real vector |

**Returns**

Qubit density matrix

#### 6.1.3.16 std::vector<cmat> qpp::choi2kraus ( const cmat & *A* ) [inline]

Orthogonal Kraus operators from Choi matrix.

**See also**

qpp::kraus2choi()

Extracts a set of orthogonal (under Hilbert-Schmidt operator norm) Kraus operators from the Choi matrix *A*

**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 orthogonal Kraus operators

#### 6.1.3.17 cmat qpp::choi2super ( const cmat & *A* ) [inline]

Converts Choi matrix to superoperator matrix.

**See also**

qpp::super2choi()

**Parameters**

| | |
|---|---|
| *A* | Choi matrix |

**Returns**

Superoperator matrix

#### 6.1.3.18 template<typename Derived1 , typename Derived2 > dyn_mat<typename Derived1::Scalar> qpp::comm ( const Eigen::MatrixBase< Derived1 > & *A*, const Eigen::MatrixBase< Derived2 > & *B* )

Commutator.

**See also**

qpp::anticomm()

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*

#### 6.1.3.19 template<typename T > std::vector<T> qpp::complement ( std::vector< T > *subsys*, idx *N* )

Constructs the complement of a subsystem vector.

**Parameters**

| | |
|---|---|
| *subsys* | Subsystem vector |
| *N* | Total number of systems |

**Returns**

The complement of *subsys* with respect to the set $\{0, 1, \ldots, N-1\}$

#### 6.1.3.20 std::vector<idx> qpp::compperm ( const std::vector< idx > & *perm*, const std::vector< idx > & *sigma* ) [inline]

Compose permutations.

**Parameters**

| | |
|---|---|
| *perm* | Permutation |
| *sigma* | Permutation |

**Returns**

Composition of the permutations *perm* ∘ *sigma* = perm(sigma)

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

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

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Wootters concurrence

#### 6.1.3.22 template<typename Derived > dyn_mat<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*

#### 6.1.3.23 double qpp::contfrac2x ( const std::vector< int > & *cf*, idx *n* ) [inline]

Real representation of a simple continued fraction.

**See also**

qpp::x2contfrac()

**Parameters**

| | |
|---|---|
| *cf* | Integer vector containing the simple continued fraction expansion |
| *n* | Number of terms considered in the continued fraction expansion. If *n* is greater than the size of *cf*,then all terms in *cf* are considered. |

**Returns**

Real representation of the simple continued fraction

#### 6.1.3.24 double qpp::contfrac2x ( const std::vector< int > & *cf* ) [inline]

Real representation of a simple continued fraction.

**See also**

qpp::x2contfrac()

**Parameters**

| | |
|---|---|
| *cf* | Integer vector containing the simple continued fraction expansion |

**Returns**

Real representation of the simple continued fraction

#### 6.1.3.25 template<typename Container > double qpp::cor ( const dmat & *probXY*, const Container & *X*, const Container & *Y*, typename std::enable_if< is_iterable< Container >::value >::type * = nullptr )

Correlation.

**Parameters**

| | |
|---|---|
| *probXY* | Real matrix representing the joint probability distribution of *X* and *Y* in lexicographical order (*X* labels the rows, *Y* labels the columns) |
| *X* | Random variable values represented by an STL-like container |
| *Y* | Random variable values represented by an STL-like container |

**Returns**

Correlation of *X* and *Y*

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

Matrix cos.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Matrix cosine of *A*

#### 6.1.3.27 template<typename Container > double qpp::cov ( const dmat & *probXY*, const Container & *X*, const Container & *Y*, typename std::enable_if< is_iterable< Container >::value >::type * = nullptr )

Covariance.

**Parameters**

| | |
|---|---|
| *probXY* | Real matrix representing the joint probability distribution of *X* and *Y* in lexicographical order (*X* labels the rows, *Y* labels the columns) |
| *X* | Random variable values represented by an STL-like container |
| *Y* | Random variable values represented by an STL-like container |

**Returns**

Covariance of *X* and *Y*

#### 6.1.3.28 template<typename OutputScalar , typename Derived > dyn_mat<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 *OutputScalar* |

**Returns**

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

#### 6.1.3.29 template<typename Derived > Derived::Scalar qpp::det ( const Eigen::MatrixBase< Derived > & *A* )

Determinant.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

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

#### 6.1.3.30 template<typename T > dyn_mat<typename T::Scalar> qpp::dirsum ( const T & *head* )

Direct sum.

**See also**

qpp::dirsumpow()

Used to stop the recursion for the variadic template version of qpp::dirsum()

**Parameters**

| | |
|---|---|
| *head* | Eigen expression |

**Returns**

Its argument *head*

#### 6.1.3.31 template<typename T , typename... Args> dyn_mat<typename T::Scalar> qpp::dirsum ( const T & *head*, const Args &... *tail* )

Direct sum.

**See also**

qpp::dirsumpow()

**Parameters**

| | |
|---|---|
| *head* | Eigen expression |
| *tail* | Variadic Eigen expression (zero or more parameters) |

**Returns**

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

#### 6.1.3.32 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::dirsum ( const std::vector< Derived > & *As* )

Direct sum.

**See also**

qpp::dirsumpow()

**Parameters**

| | |
|---|---|
| *As* | std::vector of Eigen expressions |

**Returns**

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

#### 6.1.3.33 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::dirsum ( const std::initializer_list< Derived > & *As* )

Direct sum.

**See also**

qpp::dirsumpow()

**Parameters**

| | |
|---|---|
| *As* | std::initializer_list of Eigen expressions, such as *{A1*, A2, ... ,Ak} |

**Returns**

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

#### 6.1.3.34 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::dirsumpow ( const Eigen::MatrixBase< Derived > & *A*, idx *n* )

Direct sum power.

**See also**

qpp::dirsum()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *n* | Non-negative integer |

**Returns**

Direct sum of *A* with itself *n* times $A^{\oplus n}$, as a dynamic matrix over the same scalar field as *A*

#### 6.1.3.35 template<typename Derived > internal::IOManipEigen qpp::disp ( const Eigen::MatrixBase< Derived > & *A*, double *chop* = qpp::chop )

Eigen expression ostream manipulator.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *chop* | Set to zero the elements smaller in absolute value than *chop* |

**Returns**

Instance of qpp::internal::internal::IOManipEigen

#### 6.1.3.36 internal::IOManipEigen qpp::disp ( cplx *z*, double *chop* = qpp::chop ) [inline]

Complex number ostream manipulator.

**Parameters**

| | |
|---|---|
| *z* | Complex number (or any other type implicitly cast-able to std::complex<double>) |
| *chop* | Set to zero the elements smaller in absolute value than *chop* |

**Returns**

Instance of qpp::internal::internal::IOManipEigen

#### 6.1.3.37 template<typename InputIterator > internal::IOManipRange<InputIterator> qpp::disp ( InputIterator *first*, InputIterator *last*, const std::string & *separator*, const std::string & *start* = "[", const std::string & *end* = "]" )

Range ostream manipulator.

**Parameters**

| | |
|---|---|
| *first* | Iterator to the first element of the range |
| *last* | Iterator to the last element of the range |
| *separator* | Separator |
| *start* | Left marking |
| *end* | Right marking |

**Returns**

Instance of qpp::internal::internal::IOManipRange

#### 6.1.3.38 template<typename Container > internal::IOManipRange<typename Container::const_iterator> qpp::disp ( const Container & *c*, const std::string & *separator*, const std::string & *start* = "[", const std::string & *end* = "]", typename std::enable_if< is_iterable< Container >::value >::type * = nullptr )

Standard container ostream manipulator. The container must support std::begin(), std::end() and forward iteration.

**Parameters**

| | |
|---|---|
| *c* | Container |
| *separator* | Separator |
| *start* | Left marking |
| *end* | Right marking |

**Returns**

Instance of qpp::internal::internal::IOManipRange

#### 6.1.3.39 template<typename PointerType > internal::IOManipPointer<PointerType> qpp::disp ( const PointerType * *p*, idx *n*, const std::string & *separator*, const std::string & *start* = "[", const std::string & *end* = "]" )

C-style pointer ostream manipulator.

**Parameters**

| | |
|---|---|
| *p* | Pointer to the first element |
| *n* | Number of elements to be displayed |
| *separator* | Separator |
| *start* | Left marking |
| *end* | Right marking |

**Returns**

Instance of qpp::internal::internal::IOManipPointer

#### 6.1.3.40 std::tuple<bigint, bigint, bigint> qpp::egcd ( bigint *m*, bigint *n* ) [inline]

Extended greatest common divisor of two integers.

**See also**

qpp::gcd()

**Parameters**

| | |
|---|---|
| *m* | Integer |
| *n* | Integer |

**Returns**

Tuple of: 1. Integer $a$, 2. Integer $b$, and 3. Non-negative integer $gcd(a, b)$ such that $am + bn = gcd(a, b)$

#### 6.1.3.41 template<typename Derived > std::pair<dyn_col_vect < cplx>, cmat> qpp::eig ( const Eigen::MatrixBase< Derived > & *A* )

Full eigen decomposition.

**See also**

qpp::heig()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Pair of: 1. Eigenvalues of *A*, as a complex dynamic column vector, and 2. Eigenvectors of *A*, as columns of a complex dynamic matrix

#### 6.1.3.42 template<typename Derived > double qpp::entanglement ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *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::entropy()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *dims* | Dimensions of the bi-partite system |

**Returns**

Entanglement, with the logarithm in base 2

#### 6.1.3.43 template<typename Derived > double qpp::entropy ( const Eigen::MatrixBase< Derived > & *A* )

von-Neumann entropy of the density matrix *A*

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

von-Neumann entropy, with the logarithm in base 2

#### 6.1.3.44 double qpp::entropy ( const std::vector< double > & *prob* ) [inline]

Shannon entropy of the probability distribution *prob*.

**Parameters**

| | |
|---|---|
| *prob* | Real probability vector |

**Returns**

Shannon entropy, with the logarithm in base 2

#### 6.1.3.45 template<typename Derived > dyn_col_vect<cplx> qpp::evals ( const Eigen::MatrixBase< Derived > & *A* )

Eigenvalues.

**See also**

qpp::hevals()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

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

#### 6.1.3.46 template<typename Derived > cmat qpp::evects ( const Eigen::MatrixBase< Derived > & *A* )

Eigenvectors.

**See also**

qpp::hevects()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Eigenvectors of *A*, as columns of a complex dynamic matrix

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

Matrix exponential.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Matrix exponential of *A*

#### 6.1.3.48 std::vector<bigint> qpp::factors ( bigint *n* ) [inline]

Prime factor decomposition.

**Note**

Runs in $\mathcal{O}(\sqrt{n})$ time complexity

**Parameters**

| | |
|---|---|
| *n* | Integer different from 0, 1 or -1 |

**Returns**

Integer vector containing the factors

#### 6.1.3.49 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**

$f(A)$

#### 6.1.3.50 bigint qpp::gcd ( bigint *m*, bigint *n* ) [inline]

Greatest common divisor of two integers.

**See also**

qpp::lcm()

**Parameters**

| | |
|---|---|
| *m* | Integer |
| *n* | Integer |

**Returns**

Greatest common divisor of *m* and *n*

#### 6.1.3.51 bigint qpp::gcd ( const std::vector< bigint > & *ns* ) [inline]

Greatest common divisor of a list of integers.

**See also**

qpp::lcm()

**Parameters**

| | |
|---|---|
| *ns* | List of integers |

**Returns**

Greatest common divisor of all numbers in *ns*

#### 6.1.3.52 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

#### 6.1.3.53 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::grams ( const std::vector< Derived > & *Vs* )

Gram-Schmidt orthogonalization.

**Parameters**

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

**Returns**

Gram-Schmidt vectors of *Vs* as columns of a dynamic matrix over the same scalar field as its arguments

#### 6.1.3.54 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::grams ( const std::initializer_list< Derived > & *Vs* )

Gram-Schmidt orthogonalization.

**Parameters**

| | |
|---|---|
| *Vs* | std::initializer_list of Eigen expressions as column vectors |

**Returns**

Gram-Schmidt vectors of *Vs* as columns of a dynamic matrix over the same scalar field as its arguments

#### 6.1.3.55 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::grams ( const Eigen::MatrixBase< Derived > & *A* )

Gram-Schmidt orthogonalization.

**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*

#### 6.1.3.56 template<typename Derived > std::pair<dyn_col_vect < double>, cmat> qpp::heig ( const Eigen::MatrixBase< Derived > & *A* )

Full eigen decomposition of Hermitian expression.

**See also**

qpp::eig()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Pair of: 1. Eigenvalues of *A*, as a real dynamic column vector, and 2. Eigenvectors of *A*, as columns of a complex dynamic matrix

#### 6.1.3.57 template<typename Derived > dyn_col_vect<double> qpp::hevals ( const Eigen::MatrixBase< Derived > & *A* )

Hermitian eigenvalues.

**See also**

qpp::evals()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

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

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

Hermitian eigenvectors.

**See also**

qpp::evects()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

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

#### 6.1.3.59 template<typename Derived > dyn_mat<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*

#### 6.1.3.60 std::vector<idx> qpp::invperm ( const std::vector< idx > & *perm* ) [inline]

Inverse permutation.

**Parameters**

| | |
|---|---|
| *perm* | Permutation |

**Returns**

Inverse of the permutation *perm*

#### 6.1.3.61 template<typename Derived > dyn_col_vect<typename Derived::Scalar> qpp::ip ( const Eigen::MatrixBase< Derived > & *phi*, const Eigen::MatrixBase< Derived > & *psi*, const std::vector< idx > & *subsys*, const std::vector< idx > & *dims* )

Generalized inner product.

**Parameters**

| | |
|---|---|
| *phi* | Column vector Eigen expression |
| *psi* | Column vector Eigen expression |
| *subsys* | Subsystem indexes over which *phi* is defined |

| | |
|---|---|
| *dims* | Dimensions of the multi-partite system |

**Returns**

The inner product $\langle \phi_{subsys} | \psi \rangle$, as a scalar or column vector over the remaining Hilbert space

#### 6.1.3.62 template<typename Derived > dyn_col_vect<typename Derived::Scalar> qpp::ip ( const Eigen::MatrixBase< Derived > & *phi*, const Eigen::MatrixBase< Derived > & *psi*, const std::vector< idx > & *subsys*, idx *d* = 2 )

Generalized inner product.

**Parameters**

| | |
|---|---|
| *phi* | Column vector Eigen expression |
| *psi* | Column vector Eigen expression |
| *subsys* | Subsystem indexes over which *phi* is defined |
| *d* | Subsystem dimensions |

**Returns**

The inner product $\langle \phi_{subsys} | \psi \rangle$, as a scalar or column vector over the remaining Hilbert space

#### 6.1.3.63 bool qpp::isprime ( bigint *n* ) [inline]

Primality test.

**Note**

Runs in $\mathcal{O}(\sqrt{n})$ time complexity. Use Miller-Rabin or something similar if you need high performance.

**Parameters**

| | |
|---|---|
| *n* | Integer different for 0, 1 or -1 |

**Returns**

True if the number is prime, false otherwise

#### 6.1.3.64 cmat qpp::kraus2choi ( const std::vector< cmat > & *Ks* ) [inline]

Choi matrix.

**See also**

qpp::choi2kraus()

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**

| | |
|---|---|
| *Ks* | Set of Kraus operators |

**Returns**

Choi matrix

#### 6.1.3.65 cmat qpp::kraus2super ( const std::vector< cmat > & *Ks* ) [inline]

Superoperator matrix.

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**

| | |
|---|---|
| *Ks* | Set of Kraus operators |

**Returns**

Superoperator matrix

#### 6.1.3.66 template<typename T > dyn_mat<typename T::Scalar> qpp::kron ( const T & *head* )

Kronecker product.

**See also**

qpp::kronpow()

Used to stop the recursion for the variadic template version of qpp::kron()

**Parameters**

| | |
|---|---|
| *head* | Eigen expression |

**Returns**

Its argument *head*

#### 6.1.3.67 template<typename T , typename... Args> dyn_mat<typename T::Scalar> qpp::kron ( const T & *head*, const Args &... *tail* )

Kronecker product.

**See also**

qpp::kronpow()

**Parameters**

| | |
|---|---|
| *head* | Eigen expression |
| *tail* | 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

#### 6.1.3.68 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::kron ( const std::vector< Derived > & *As* )

Kronecker product.

**See also**

qpp::kronpow()

**Parameters**

| | |
|---|---|
| *As* | 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

#### 6.1.3.69 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::kron ( const std::initializer_list< Derived > & *As* )

Kronecker product.

**See also**

qpp::kronpow()

**Parameters**

| | |
|---|---|
| *As* | 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

#### 6.1.3.70 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::kronpow ( const Eigen::MatrixBase< Derived > & *A*, idx *n* )

Kronecker power.

**See also**

qpp::kron()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *n* | 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*

#### 6.1.3.71 bigint qpp::lcm ( bigint *m*, bigint *n* ) [inline]

Least common multiple of two integers.

**See also**

qpp::gcd()

**Parameters**

| | |
|---|---|
| *m* | Integer |
| *n* | Integer |

**Returns**

Least common multiple of *m* and *n*

#### 6.1.3.72 bigint qpp::lcm ( const std::vector< bigint > & *ns* ) [inline]

Least common multiple of a list of integers.

**See also**

qpp::gcd()

**Parameters**

| | |
|---|---|
| *ns* | List of integers |

**Returns**

Least common multiple of all numbers in *ns*

#### 6.1.3.73 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::load ( const std::string & *fname* )

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

**See also**

qpp::save()

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");
```

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *fname* | Output file name |

#### 6.1.3.74 template<typename Derived > Derived qpp::loadMATLABmatrix ( const std::string & , const std::string & )

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

**See also**

qpp::saveMATLABmatrix()

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)

#### 6.1.3.75 template<> dmat qpp::loadMATLABmatrix ( const std::string & *mat_file*, const std::string & *var_name* ) [inline]

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

**See also**

qpp::saveMATLABmatrix()

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**

| | |
|---|---|
| *mat_file* | MATALB .mat file |
| *var_name* | Variable name in the .mat file representing the matrix to be loaded |

**Returns**

Eigen double dynamic matrix (qpp::dmat)

#### 6.1.3.76 template<> cmat qpp::loadMATLABmatrix ( const std::string & *mat_file*, const std::string & *var_name* ) [inline]

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

**See also**

qpp::saveMATLABmatrix()

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**

| | |
|---|---|
| *mat_file* | MATALB .mat file |
| *var_name* | Variable name in the .mat file representing the matrix to be loaded |

**Returns**

Eigen complex dynamic matrix (qpp::cmat)

#### 6.1.3.77 template<typename Derived > Derived::Scalar qpp::logdet ( const Eigen::MatrixBase< Derived > & *A* )

Logarithm of the determinant.

Useful when the determinant overflows/underflows

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

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

#### 6.1.3.78 template<typename Derived > cmat qpp::logm ( const Eigen::MatrixBase< Derived > & *A* )

Matrix logarithm.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Matrix logarithm of *A*

#### 6.1.3.79 template<typename Derived > double qpp::lognegativity ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *dims* )

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

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *dims* | Dimensions of the bi-partite system |

**Returns**

Logarithmic negativity, with the logarithm in base 2

#### 6.1.3.80 std::vector<double> qpp::marginalX ( const dmat & *probXY* ) [inline]

Marginal distribution.

**Parameters**

| | |
|---|---|
| *probXY* | Real matrix representing the joint probability distribution of *X* and *Y* in lexicographical order (*X* labels the rows, *Y* labels the columns) |

**Returns**

Real vector consisting of the marginal distribution of *X*

#### 6.1.3.81 std::vector<double> qpp::marginalY ( const dmat & *probXY* ) [inline]

Marginal distribution.

**Parameters**

| | |
|---|---|
| *probXY* | Real matrix representing the joint probability distribution of *X* and *Y* in lexicographical order (*X* labels the rows, *Y* labels the columns) |

**Returns**

Real vector consisting of the marginal distribution of *Y*

#### 6.1.3.82 template<typename Derived > std::tuple<idx, 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**

Tuple of: 1. Result of the measurement, 2. Vector of outcome probabilities, and 3. Vector of post-measurement normalized states

#### 6.1.3.83 template<typename Derived > std::tuple<idx, 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*.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *Ks* | Set of Kraus operators |

**Returns**

Tuple of: 1. Result of the measurement, 2. Vector of outcome probabilities, and 3. Vector of post-measurement normalized states

#### 6.1.3.84 template<typename Derived > std::tuple<idx, std::vector<double>, std::vector<cmat> > qpp::measure ( const Eigen::MatrixBase< Derived > & *A*, const cmat & *U* )

Measures the state *A* in the orthonormal basis specified by the unitary matrix *U*.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *U* | Unitary matrix whose columns represent the measurement basis vectors |

**Returns**

Tuple of: 1. Result of the measurement, 2. Vector of outcome probabilities, and 3. Vector of post-measurement normalized states

#### 6.1.3.85 template<typename Derived > std::tuple<idx, std::vector<double>, std::vector<cmat> > qpp::measure ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< cmat > & *Ks*, const std::vector< idx > & *subsys*, const std::vector< idx > & *dims* )

Measures the part *subsys* of the multi-partite state vector or density matrix *A* using the set of Kraus operators *Ks*.

**See also**

qpp::measure_seq()

**Note**

The dimension of all *Ks* must match the dimension of *subsys*. The measurement is destructive, i.e. the measured subsystems are traced away.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *Ks* | Set of Kraus operators |
| *subsys* | Subsystem indexes that are measured |
| *dims* | Dimensions of the multi-partite system |

**Returns**

Tuple of: 1. Result of the measurement, 2. Vector of outcome probabilities, and 3. Vector of post-measurement normalized states

#### 6.1.3.86 template<typename Derived > std::tuple<idx, std::vector<double>, std::vector<cmat> > qpp::measure ( const Eigen::MatrixBase< Derived > & *A*, const std::initializer_list< cmat > & *Ks*, const std::vector< idx > & *subsys*, const std::vector< idx > & *dims* )

Measures the part *subsys* of the multi-partite state vector or density matrix *A* using the set of Kraus operators *Ks*.

**See also**

qpp::measure_seq()

**Note**

The dimension of all *Ks* must match the dimension of *subsys*. The measurement is destructive, i.e. the measured subsystems are traced away.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *Ks* | Set of Kraus operators |
| *subsys* | Subsystem indexes that are measured |
| *dims* | Dimensions of the multi-partite system |

**Returns**

Tuple of: 1. Result of the measurement, 2. Vector of outcome probabilities, and 3. Vector of post-measurement normalized states

#### 6.1.3.87 template<typename Derived > std::tuple<idx, std::vector<double>, std::vector<cmat> > qpp::measure ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< cmat > & *Ks*, const std::vector< idx > & *subsys*, idx *d* = 2 )

Measures the part *subsys* of the multi-partite state vector or density matrix *A* using the set of Kraus operators *Ks*.

**See also**

qpp::measure_seq()

**Note**

The dimension of all *Ks* must match the dimension of *subsys*. The measurement is destructive, i.e. the measured subsystems are traced away.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *Ks* | Set of Kraus operators |
| *subsys* | Subsystem indexes that are measured |
| *d* | Subsystem dimensions |

**Returns**

Tuple of: 1. Result of the measurement, 2. Vector of outcome probabilities, and 3. Vector of post-measurement normalized states

#### 6.1.3.88 template<typename Derived > std::tuple<idx, std::vector<double>, std::vector<cmat> > qpp::measure ( const Eigen::MatrixBase< Derived > & *A*, const std::initializer_list< cmat > & *Ks*, const std::vector< idx > & *subsys*, idx *d* = 2 )

Measures the part *subsys* of the multi-partite state vector or density matrix *A* using the set of Kraus operators *Ks*.

**See also**

qpp::measure_seq()

**Note**

The dimension of all *Ks* must match the dimension of *subsys*. The measurement is destructive, i.e. the measured subsystems are traced away.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *Ks* | Set of Kraus operators |
| *subsys* | Subsystem indexes that are measured |
| *d* | Subsystem dimensions |

**Returns**

Tuple of: 1. Result of the measurement, 2. Vector of outcome probabilities, and 3. Vector of post-measurement normalized states

#### 6.1.3.89 template<typename Derived > std::tuple<idx, std::vector<double>, std::vector<cmat> > qpp::measure ( const Eigen::MatrixBase< Derived > & *A*, const cmat & *V*, const std::vector< idx > & *subsys*, const std::vector< idx > & *dims* )

Measures the part *subsys* of the multi-partite state vector or density matrix *A* in the orthonormal basis or rank-1 POVM specified by the matrix *V*.

**See also**

qpp::measure_seq()

**Note**

The dimension of *V* must match the dimension of *subsys*. The measurement is destructive, i.e. the measured subsystems are traced away.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *V* | Matrix whose columns represent the measurement basis vectors or the bra parts of the rank-1 POVM |
| *subsys* | Subsystem indexes that are measured |
| *dims* | Dimensions of the multi-partite system |

**Returns**

Tuple of: 1. Result of the measurement, 2. Vector of outcome probabilities, and 3. Vector of post-measurement normalized states

#### 6.1.3.90 template<typename Derived > std::tuple<idx, std::vector<double>, std::vector<cmat> > qpp::measure ( const Eigen::MatrixBase< Derived > & *A*, const cmat & *V*, const std::vector< idx > & *subsys*, idx *d* = 2 )

Measures the part *subsys* of the multi-partite state vector or density matrix *A* in the orthonormal basis or rank-1 POVM specified by the matrix *V*.

**See also**

qpp::measure_seq()

**Note**

The dimension of *V* must match the dimension of *subsys*. The measurement is destructive, i.e. the measured subsystems are traced away.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *V* | Matrix whose columns represent the measurement basis vectors or the bra parts of the rank-1 POVM |
| *subsys* | Subsystem indexes that are measured |
| *d* | Subsystem dimensions |

**Returns**

Tuple of: 1. Result of the measurement, 2. Vector of outcome probabilities, and 3. Vector of post-measurement normalized states

#### 6.1.3.91 template<typename Derived > std::tuple<std::vector<idx>, double, cmat> qpp::measure_seq ( const Eigen::MatrixBase< Derived > & *A*, std::vector< idx > *subsys*, std::vector< idx > *dims* )

Sequentially measures the part *subsys* of the multi-partite state vector or density matrix *A* in the computational basis.

**See also**

qpp::measure()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *subsys* | Subsystem indexes that are measured |
| *dims* | Dimensions of the multi-partite system |

**Returns**

Tuple of: 1. Vector of outcome results of the measurement (ordered in increasing order with respect to *subsys*, i.e. first measurement result corresponds to the subsystem with the smallest index), 2. Outcome probability, and 3. Post-measurement normalized state

#### 6.1.3.92 template<typename Derived > std::tuple<std::vector<idx>, double, cmat> qpp::measure_seq ( const Eigen::MatrixBase< Derived > & *A*, std::vector< idx > *subsys*, idx *d* = 2 )

Sequentially measures the part *subsys* of the multi-partite state vector or density matrix *A* in the computational basis.

**See also**

qpp::measure()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *subsys* | Subsystem indexes that are measured |
| *d* | Subsystem dimensions |

**Returns**

Tuple of: 1. Vector of outcome results of the measurement (ordered in increasing order with respect to *subsys*, i.e. first measurement result corresponds to the subsystem with the smallest index), 2. Outcome probability, and 3. Post-measurement normalized state

#### 6.1.3.93 ket qpp::mket ( const std::vector< idx > & *mask*, const std::vector< idx > & *dims* ) [inline]

Multi-partite qudit ket.

Constructs the multi-partite qudit ket $|\mathrm{mask}\rangle$, 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**

| | |
|---|---|
| *mask* | std::vector of non-negative integers |
| *dims* | Dimensions of the multi-partite system |

**Returns**

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

#### 6.1.3.94 ket qpp::mket ( const std::vector< idx > & *mask*, idx *d* = 2 ) [inline]

Multi-partite qudit ket.

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

**Parameters**

| | |
|---|---|
| *mask* | std::vector of non-negative integers |
| *d* | Subsystem dimensions |

**Returns**

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

#### 6.1.3.95 bigint qpp::modinv ( bigint *a*, bigint *p* ) [inline]

Modular inverse of *a* mod *p*.

**See also**

qpp::egcd()

**Note**

*a* and *p* must be co-prime

**Parameters**

| | |
|---|---|
| *m* | Non-negative integer |
| *p* | Non-negative integer |

**Returns**

Modular inverse $a^{-1}$ of *a* mod *p*

#### 6.1.3.96 bigint qpp::modpow ( bigint *a*, bigint *n*, bigint *p* ) [inline]

Fast integer power modulo *p* based on the SQUARE-AND-MULTIPLY algorithm.

Computes $a^n \mathrm{mod} p$

**Parameters**

| | |
|---|---|
| *a* | Non-negative integer |
| *n* | Non-negative integer |
| *p* | Strictly positive integer |

**Returns**

$a^n \bmod p$

#### 6.1.3.97 cmat qpp::mprj ( const std::vector< idx > & *mask*, const std::vector< idx > & *dims* ) [inline]

Projector onto multi-partite qudit ket.

Constructs the projector onto the multi-partite qudit ket $|\mathrm{mask}\rangle$, 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**

| | |
|---|---|
| *mask* | std::vector of non-negative integers |
| *dims* | Dimensions of the multi-partite system |

**Returns**

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

#### 6.1.3.98 cmat qpp::mprj ( const std::vector< idx > & *mask*, idx *d* = 2 ) [inline]

Projector onto multi-partite qudit ket.

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

**Parameters**

| | |
|---|---|
| *mask* | std::vector of non-negative integers |
| *d* | Subsystem dimensions |

**Returns**

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

#### 6.1.3.99 idx qpp::multiidx2n ( const std::vector< idx > & *midx*, const std::vector< idx > & *dims* ) [inline]

Multi-index to non-negative integer index.

**See also**

qpp::n2multiidx()

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

**Parameters**

| | |
|---|---|
| *midx* | Multi-index |
| *dims* | Dimensions of the multi-partite system |

**Returns**

Non-negative integer index

#### 6.1.3.100 std::vector<idx> qpp::n2multiidx ( idx *n*, const std::vector< idx > & *dims* ) [inline]

Non-negative integer index to multi-index.

**See also**

qpp::multiidx2n()

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

**Parameters**

| | |
|---|---|
| *n* | Non-negative integer index |
| *dims* | Dimensions of the multi-partite system |

**Returns**

Multi-index of the same size as *dims*

#### 6.1.3.101 template<typename Derived > double qpp::negativity ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *dims* )

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

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *dims* | Dimensions of the bi-partite system |

**Returns**

Negativity

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

Frobenius norm.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Frobenius norm of *A*

#### 6.1.3.103 cplx qpp::omega ( idx *D* ) [inline]

D-th root of unity.

**Parameters**

| | |
|---|---|
| *D* | Non-negative integer |

**Returns**

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

#### 6.1.3.104 constexpr cplx qpp::operator""_i ( unsigned long long int *x* ) [inline], [noexcept]

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

Example:

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

#### 6.1.3.105 constexpr cplx qpp::operator""_i ( long double *x* ) [inline], [noexcept]

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

Example:

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

#### 6.1.3.106 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::powm ( const Eigen::MatrixBase< Derived > & *A*, idx *n* )

Matrix power.

**See also**

qpp::spectralpowm()

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*

#### 6.1.3.107 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::prj ( const Eigen::MatrixBase< Derived > & *V* )

Projector.

Normalized projector onto state vector

**Parameters**

| | |
|---|---|
| *V* | 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*

#### 6.1.3.108 template<typename Derived > Derived::Scalar qpp::prod ( const Eigen::MatrixBase< Derived > & *A* )

Element-wise product of *A*.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Element-wise product of *A*, as a scalar over the same scalar field as *A*

#### 6.1.3.109 template<typename InputIterator > std::iterator_traits<InputIterator>::value_type qpp::prod ( InputIterator *first*, InputIterator *last* )

Element-wise product of an STL-like range.

**Parameters**

| | |
|---|---|
| *first* | Iterator to the first element of the range |
| *last* | Iterator to the last element of the range |

**Returns**

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

#### 6.1.3.110 template<typename Container > Container::value_type qpp::prod ( const Container & *c*, typename std::enable_if< **is_iterable**< Container >::value >::type * = `nullptr` )

Element-wise product of the elements of an STL-like container.

**Parameters**

| | |
|---|---|
| *c* | STL-like container |

**Returns**

Element-wise product of the elements of the container, as a scalar over the same scalar field as the container

#### 6.1.3.111 template<typename Derived > **dyn_mat**<typename Derived::Scalar> qpp::ptrace ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< **idx** > & *subsys*, const std::vector< **idx** > & *dims* )

Partial trace.

**See also**

qpp::ptrace1(), qpp::ptrace2()

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

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *subsys* | Subsystem indexes |
| *dims* | Dimensions of the multi-partite system |

**Returns**

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

#### 6.1.3.112 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::ptrace ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *subsys*, idx *d* = 2 )

Partial trace.

**See also**

qpp::ptrace1(), qpp::ptrace2()

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

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *subsys* | Subsystem indexes |
| *d* | Subsystem dimensions |

**Returns**

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

#### 6.1.3.113 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::ptrace1 ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *dims* )

Partial trace.

**See also**

qpp::ptrace2()

Partial trace over the first subsystem of bi-partite state vector or density matrix

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *dims* | Dimensions of the bi-partite system (must be a std::vector with 2 elements) |

**Returns**

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

#### 6.1.3.114 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::ptrace2 ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *dims* )

Partial trace.

**See also**

qpp::ptrace1()

Partial trace over the second subsystem of bi-partite state vector or density matrix

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *dims* | Dimensions of the bi-partite system (must be a std::vector with 2 elements) |

**Returns**

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

#### 6.1.3.115 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::ptranspose ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *subsys*, const std::vector< idx > & *dims* )

Partial transpose.

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

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *subsys* | Subsystem indexes |
| *dims* | Dimensions of the multi-partite system |

**Returns**

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

#### 6.1.3.116 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::ptranspose ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *subsys*, idx *d* = 2 )

Partial transpose.

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

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *subsys* | Subsystem indexes |
| *d* | Subsystem dimensions |

**Returns**

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

#### 6.1.3.117 template<typename Derived > double qpp::qmutualinfo ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *subsysA*, const std::vector< idx > & *subsysB*, const std::vector< idx > & *dims* )

Quantum mutual information between 2 subsystems of a composite system.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *subsysA* | Indexes of the first subsystem |
| *subsysB* | Indexes of the second subsystem |
| *dims* | Dimensions of the multi-partite system |

**Returns**

Mutual information between the 2 subsystems

#### 6.1.3.118 template<typename Derived > double qpp::qmutualinfo ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *subsysA*, const std::vector< idx > & *subsysB*, idx *d* = 2 )

Quantum mutual information between 2 subsystems of a composite system.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *subsysA* | Indexes of the first subsystem |
| *subsysB* | Indexes of the second subsystem |
| *d* | Subsystem dimensions |

**Returns**

Mutual information between the 2 subsystems

#### 6.1.3.119 double qpp::rand ( double *a* = 0, double *b* = 1 ) [inline]

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

**Parameters**

| | |
|---|---|
| *a* | Beginning of the interval, belongs to it |
| *b* | End of the interval, does not belong to it |

**Returns**

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

#### 6.1.3.120 bigint qpp::rand ( bigint *a* = std::numeric_limits<bigint>::min(), bigint *b* = std::numeric_limits<bigint>::max() ) [inline]

Generates a random big integer uniformly distributed in the interval [a, b].

**Parameters**

| | |
|---|---|
| *a* | Beginning of the interval, belongs to it |
| *b* | End of the interval, belongs to it |

**Returns**

Random big integer uniformly distributed in the interval [a, b]

#### 6.1.3.121 ubigint qpp::rand ( ubigint *a* = std::numeric_limits<ubigint>::min(), ubigint *b* = std::numeric_limits<ubigint>::max() ) [inline]

Generates a random non-negative big integer uniformly distributed in the interval [a, b].

**Parameters**

| | |
|---|---|
| *a* | Beginning of the interval, belongs to it |
| *b* | End of the interval, belongs to it |

**Returns**

Random non-negative big integer uniformly distributed in the interval [a, b]

#### 6.1.3.122 template<typename Derived > Derived qpp::rand ( idx *rows*, idx *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

#### 6.1.3.123 template<> dmat qpp::rand ( idx *rows*, idx *cols*, double *a*, double *b* ) [inline]

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**

| | |
|---|---|
| *rows* | Number of rows of the random generated matrix |
| *cols* | Number of columns of the random generated matrix |
| *a* | Beginning of the interval, belongs to it |
| *b* | End of the interval, does not belong to it |

**Returns**

Random real matrix

#### 6.1.3.124 template<> cmat qpp::rand ( idx *rows*, idx *cols*, double *a*, double *b* ) [inline]

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**

| | |
|---|---|
| *rows* | Number of rows of the random generated matrix |
| *cols* | Number of columns of the random generated matrix |
| *a* | Beginning of the interval, belongs to it |
| *b* | End of the interval, does not belong to it |

**Returns**

Random complex matrix

#### 6.1.3.125 cmat qpp::randH ( idx *D* ) [inline]

Generates a random Hermitian matrix.

**Parameters**

| | |
|---|---|
| *D* | Dimension of the Hilbert space |

**Returns**

Random Hermitian matrix

#### 6.1.3.126 idx qpp::randidx ( idx *a* = std::numeric_limits<idx>::min(), idx *b* = std::numeric_limits<idx>::max() ) [inline]

Generates a random index (idx) uniformly distributed in the interval [a, b].

**Parameters**

| | |
|---|---|
| *a* | Beginning of the interval, belongs to it |
| *b* | End of the interval, belongs to it |

**Returns**

Random index (idx) uniformly distributed in the interval [a, b]

#### 6.1.3.127 ket qpp::randket ( idx *D* ) [inline]

Generates a random normalized ket (pure state vector)

**Parameters**

| | |
|---|---|
| *D* | Dimension of the Hilbert space |

**Returns**

Random normalized ket

#### 6.1.3.128 std::vector<cmat> qpp::randkraus ( idx *N*, idx *D* ) [inline]

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

#### 6.1.3.129 template<typename Derived > Derived qpp::randn ( idx *rows*, idx *cols*, double *mean* = 0, double *sigma* = 1 )

Generates a random matrix with entries normally distributed in N(mean, sigma)

If complex, then both real and imaginary parts are normally distributed in N(mean, sigma)

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

#### 6.1.3.130 template<> dmat qpp::randn ( idx *rows*, idx *cols*, double *mean*, double *sigma* ) [inline]

Generates a random real matrix with entries normally distributed in N(mean, 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**

| | |
|---|---|
| *rows* | Number of rows of the random generated matrix |
| *cols* | Number of columns of the random generated matrix |
| *mean* | Mean |
| *sigma* | Standard deviation |

**Returns**

Random real matrix

#### 6.1.3.131 template<> cmat qpp::randn ( idx *rows*, idx *cols*, double *mean*, double *sigma* ) [inline]

Generates a random complex matrix with entries (both real and imaginary) normally distributed in N(mean, 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**

| | |
|---|---|
| *rows* | Number of rows of the random generated matrix |
| *cols* | Number of columns of the random generated matrix |
| *mean* | Mean |
| *sigma* | Standard deviation |

**Returns**

Random complex matrix

#### 6.1.3.132 double qpp::randn ( double *mean* = 0, double *sigma* = 1 ) [inline]

Generates a random real number (double) normally distributed in N(mean, sigma)

**Parameters**

| | |
|---|---|
| *mean* | Mean |
| *sigma* | Standard deviation |

**Returns**

Random real number normally distributed in N(mean, sigma)

#### 6.1.3.133 std::vector<idx> qpp::randperm ( idx *n* ) [inline]

Generates a random uniformly distributed permutation.

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

**Parameters**

| | |
|---|---|
| *n* | Size of the permutation |

**Returns**

Random permutation of size *n*

#### 6.1.3.134 cmat qpp::randrho ( idx *D* ) [inline]

Generates a random density matrix.

**Parameters**

| | |
|---|---|
| *D* | Dimension of the Hilbert space |

**Returns**

Random density matrix

#### 6.1.3.135 cmat qpp::randU ( idx *D* ) [inline]

Generates a random unitary matrix.

**Parameters**

| | |
|---|---|
| *D* | Dimension of the Hilbert space |

**Returns**

Random unitary

#### 6.1.3.136 cmat qpp::randV ( idx *Din*, idx *Dout* ) [inline]

Generates a random isometry matrix.

**Parameters**

| | |
|---|---|
| *Din* | Size of the input Hilbert space |
| *Dout* | Size of the output Hilbert space |

**Returns**

Random isometry matrix

#### 6.1.3.137 template<typename Derived > double qpp::renyi ( const Eigen::MatrixBase< Derived > & *A*, double *alpha* )

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

**Note**

When $\alpha \to 1$ the Renyi entropy converges to the von-Neumann entropy, with the logarithm in base 2

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *alpha* | Non-negative real number, use qpp::infty for $\alpha = \infty$ |

**Returns**

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

#### 6.1.3.138 double qpp::renyi ( const std::vector< double > & *prob*, double *alpha* ) [inline]

Renyi- $\alpha$ entropy of the probability distribution *prob*, for $\alpha \geq 0$.

**Note**

When $\alpha \to 1$ the Renyi entropy converges to the Shannon entropy, with the logarithm in base 2

**Parameters**

| | |
|---|---|
| *prob* | Real probability vector |
| *alpha* | Non-negative real number, use qpp::infty for $\alpha = \infty$ |

**Returns**

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

#### 6.1.3.139 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::reshape ( const Eigen::MatrixBase< Derived > & *A*, idx *rows*, idx *cols* )

Reshape.

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

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *rows* | Number of rows of the reshaped matrix |
| *cols* | 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*

#### 6.1.3.140 template<typename Derived > std::vector<double> qpp::rho2bloch ( const Eigen::MatrixBase< Derived > & *A* )

Computes the 3-dimensional real Bloch vector corresponding to the qubit density matrix *A*.

**See also**

qpp::bloch2rho()

**Note**

It is implicitly assumed that the density matrix is Hermitian

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

3-dimensional Bloch vector

#### 6.1.3.141 template<typename Derived > dyn_col_vect<typename Derived::Scalar> qpp::rho2pure ( const Eigen::MatrixBase< Derived > & *A* )

Finds the pure state representation of a matrix proportional to a projector onto a pure state.

**Note**

No purity check is done, the input state *A* must have rank one, otherwise the function returns the first non-zero eigenvector of *A*

**Parameters**

| | |
|---|---|
| *A* | Eigen expression, assumed to be proportional to a projector onto a pure state, i.e. *A* is assumed to have rank one |

**Returns**

The unique non-zero eigenvector of *A*, as a dynamic column vector over the same scalar field as *A*

#### 6.1.3.142 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::load()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *fname* | Output file name |

#### 6.1.3.143 template<typename Derived > void qpp::saveMATLABmatrix ( const Eigen::MatrixBase< Derived > & , const std::string & , const std::string & , const std::string & )

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

**See also**

qpp::loadMATLABmatrix()

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)

#### 6.1.3.144 template<> void qpp::saveMATLABmatrix ( const Eigen::MatrixBase< dmat > & *A*, const std::string & *mat_file*, const std::string & *var_name*, const std::string & *mode* ) [inline]

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

**See also**

qpp::loadMATLABmatrix()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression over the complex field |
| *mat_file* | MATALB .mat file |
| *var_name* | Variable name in the .mat file representing the matrix to be saved |
| *mode* | Saving mode (append, overwrite etc.), see MATLAB *matOpen()* documentation for details |

#### 6.1.3.145 template<> void qpp::saveMATLABmatrix ( const Eigen::MatrixBase< cmat > & *A*, const std::string & *mat_file*, const std::string & *var_name*, const std::string & *mode* ) [inline]

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

**See also**

qpp::loadMATLABmatrix()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression over the complex field |
| *mat_file* | MATALB .mat file |
| *var_name* | Variable name in the .mat file representing the matrix to be saved |
| *mode* | Saving mode (append, overwrite etc.), see MATLAB *matOpen()* documentation for details |

#### 6.1.3.146 template<typename Derived > double qpp::schatten ( const Eigen::MatrixBase< Derived > & *A*, double *p* )

Schatten matrix norm.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *p* | Real number, greater or equal to 1, use qpp::infty for $p = \infty$ |

**Returns**

Schatten-*p* matrix norm of *A*

#### 6.1.3.147 template<typename Derived > cmat qpp::schmidtA ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *dims* )

Schmidt basis on Alice side.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *dims* | Dimensions of the bi-partite system |

**Returns**

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

#### 6.1.3.148 template<typename Derived > cmat qpp::schmidtB ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *dims* )

Schmidt basis on Bob side.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *dims* | Dimensions of the bi-partite system |

**Returns**

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

#### 6.1.3.149 template<typename Derived > dyn_col_vect<double> qpp::schmidtcoeffs ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *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::schmidtprobs()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *dims* | Dimensions of the bi-partite system |

**Returns**

Schmidt coefficients of *A*, as a real dynamic column vector

#### 6.1.3.150 template<typename Derived > std::vector<double> qpp::schmidtprobs ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *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::schmidtcoeffs()

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *dims* | Dimensions of the bi-partite system |

**Returns**

Real vector consisting of the Schmidt probabilites of *A*

#### 6.1.3.151 template<typename Container > double qpp::sigma ( const std::vector< double > & *prob*, const Container & *X*, typename std::enable_if< is_iterable< Container >::value >::type * = nullptr )

Standard deviation.

**Parameters**

| | |
|---|---|
| *prob* | Real probability vector representing the probability distribution of *X* |
| *X* | Random variable values represented by an STL-like container |

**Returns**

Standard deviation of *X*

#### 6.1.3.152 template<typename Derived > cmat qpp::sinm ( const Eigen::MatrixBase< Derived > & *A* )

Matrix sin.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Matrix sine of *A*

#### 6.1.3.153 template<typename Derived > cmat qpp::spectralpowm ( const Eigen::MatrixBase< Derived > & *A*, const cplx *z* )

Matrix power.

**See also**

qpp::powm()

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 $A^z$

#### 6.1.3.154 template<typename Derived > cmat qpp::sqrtm ( const Eigen::MatrixBase< Derived > & *A* )

Matrix square root.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Matrix square root of *A*

#### 6.1.3.155 template<typename Derived > Derived::Scalar qpp::sum ( const Eigen::MatrixBase< Derived > & *A* )

Element-wise sum of *A*.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

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

#### 6.1.3.156 template<typename InputIterator > std::iterator_traits<InputIterator>::value_type qpp::sum ( InputIterator *first*, InputIterator *last* )

Element-wise sum of an STL-like range.

**Parameters**

| | |
|---|---|
| *first* | Iterator to the first element of the range |
| *last* | Iterator to the last element of the range |

**Returns**

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

#### 6.1.3.157 template<typename Container > Container::value_type qpp::sum ( const Container & *c*, typename std::enable_if< is_iterable< Container >::value >::type * = nullptr )

Element-wise sum of the elements of an STL-like container.

**Parameters**

| | |
|---|---|
| *c* | STL-like container |

**Returns**

Element-wise sum of the elements of the container, as a scalar over the same scalar field as the container

#### 6.1.3.158 cmat qpp::super2choi ( const cmat & *A* ) [inline]

Converts superoperator matrix to Choi matrix.

**See also**

qpp::choi2super()

**Parameters**

| | |
|---|---|
| *A* | Superoperator matrix |

**Returns**

Choi matrix

#### 6.1.3.159 template<typename Derived > dyn_col_vect<double> qpp::svals ( const Eigen::MatrixBase< Derived > & *A* )

Singular values.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Singular values of *A*, ordered in decreasing order, as a real dynamic column vector

#### 6.1.3.160 template<typename Derived > std::tuple<cmat, dyn_col_vect < double>, cmat> qpp::svd ( const Eigen::MatrixBase< Derived > & *A* )

Full singular value decomposition.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Tuple of: 1. Left sigular vectors of *A*, as columns of a complex dynamic matrix, 2. Singular values of *A*, ordered in decreasing order, as a real dynamic column vector, and 3. Right singular vectors of *A*, as columns of a complex dynamic matrix

#### 6.1.3.161 template<typename Derived > cmat qpp::svdU ( const Eigen::MatrixBase< Derived > & *A* )

Left singular vectors.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

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

#### 6.1.3.162 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*

#### 6.1.3.163 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::syspermute ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *perm*, const std::vector< idx > & *dims* )

Subsystem permutation.

Permutes the subsystems of a state vector or density matrix. The qubit *perm*[*i*] is permuted to the location *i*.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *perm* | Permutation |
| *dims* | Dimensions of the multi-partite system |

**Returns**

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

#### 6.1.3.164 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::syspermute ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *perm*, idx *d* = 2 )

Subsystem permutation.

Permutes the subsystems of a state vector or density matrix. The qubit *perm*[*i*] is permuted to the location *i*.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *perm* | Permutation |
| *d* | Subsystem dimensions |

**Returns**

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

#### 6.1.3.165 template<typename Derived > Derived::Scalar qpp::trace ( const Eigen::MatrixBase< Derived > & *A* )

Trace.

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

**Returns**

Trace of *A*, as a scalar over the same scalar field as *A*

#### 6.1.3.166 template<typename Derived > dyn_mat<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*

#### 6.1.3.167 template<typename Derived > double qpp::tsallis ( const Eigen::MatrixBase< Derived > & *A*, double *q* )

Tsallis- $q$ entropy of the density matrix *A*, for $q \geq 0$.

**Note**

When $q \to 1$ the Tsallis entropy converges to the von-Neumann entropy, with the logarithm in base $e$

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |
| *q* | Non-negative real number |

**Returns**

Tsallis- $q$ entropy

#### 6.1.3.168 double qpp::tsallis ( const std::vector< double > & *prob*, double *q* ) [inline]

Tsallis- $q$ entropy of the probability distribution *prob*, for $q \geq 0$.

**Note**

When $q \to 1$ the Tsallis entropy converges to the Shannon entropy, with the logarithm in base $e$

**Parameters**

| | |
|---|---|
| *prob* | Real probability vector |
| *q* | Non-negative real number |

**Returns**

Tsallis- $q$ entropy

#### 6.1.3.169 std::vector<double> qpp::uniform ( idx *N* ) [inline]

Uniform probability distribution vector.

**Parameters**

| | |
|---|---|
| *N* | Size of the alphabet |

**Returns**

Real vector consisting of a uniform distribution of size *N*

#### 6.1.3.170 template<typename Container > double qpp::var ( const std::vector< double > & *prob*, const Container & *X*, typename std::enable_if< is_iterable< Container >::value >::type * = nullptr )

Variance.

**Parameters**

| | |
|---|---|
| *prob* | Real probability vector representing the probability distribution of *X* |
| *X* | Random variable values represented by an STL-like container |

**Returns**

Variance of *X*

#### 6.1.3.171 std::vector<int> qpp::x2contfrac ( double *x*, idx *n*, idx *cut* = 1e5 ) [inline]

Simple continued fraction expansion.

**See also**

qpp::contfrac2x()

**Parameters**

| | |
|---|---|
| *x* | Real number |
| *n* | Maximum number of terms in the expansion |
| *cut* | Stop the expansion when the next term is greater than *cut* |

**Returns**

Integer vector containing the simple continued fraction expansion of *x*. If there are *m* less than *n* terms in the expansion, a shorter vector with *m* components is returned.

### 6.1.4 Variable Documentation

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

Used in qpp::disp() for setting to zero numbers that have their absolute value smaller than qpp::chop.

#### 6.1.4.2 constexpr double qpp::ee = 2.71828182845904523536028747135266249 97

Base of natural logarithm, $e$.

#### 6.1.4.3 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
```

#### 6.1.4.4 constexpr double qpp::infty = std::numeric_limits<double>::infinity()

Used to denote infinity in double precision.

#### 6.1.4.5 constexpr idx qpp::maxn = 64

Maximum number of allowed qu(d)its (subsystems)

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

#### 6.1.4.6 constexpr double qpp::pi = 3.14159265358979323846264338327950288 4

$\pi$

## 6.2 qpp::experimental Namespace Reference

Experimental/test functions/classes, do not use or modify.

### 6.2.1 Detailed Description

Experimental/test functions/classes, do not use or modify.

## 6.3 qpp::internal Namespace Reference

Internal utility functions, do not use/modify.

### Namespaces

- _details

### Classes

- class IOManipEigen
- class IOManipPointer
- class IOManipRange
- class Singleton

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

### Functions

- void _n2multiidx (idx n, idx numdims, const idx *dims, idx *result) noexcept
- idx _multiidx2n (const idx *midx, idx numdims, const idx *dims) noexcept
- 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_rvector (const Eigen::MatrixBase< Derived > &A)

- template<typename Derived >
  bool _check_cvector (const Eigen::MatrixBase< Derived > &A)
- template<typename T >
  bool _check_nonzero_size (const T &x) noexcept
- template<typename T1 , typename T2 >
  bool _check_matching_sizes (const T1 &lhs, const T2 &rhs) noexcept
- bool _check_dims (const std::vector< idx > &dims)
- template<typename Derived >
  bool _check_dims_match_mat (const std::vector< idx > &dims, const Eigen::MatrixBase< Derived > &A)
- template<typename Derived >
  bool _check_dims_match_cvect (const std::vector< idx > &dims, const Eigen::MatrixBase< Derived > &V)
- template<typename Derived >
  bool _check_dims_match_rvect (const std::vector< idx > &dims, const Eigen::MatrixBase< Derived > &V)
- bool _check_eq_dims (const std::vector< idx > &dims, idx dim) noexcept
- bool _check_subsys_match_dims (const std::vector< idx > &subsys, const std::vector< idx > &dims)
- template<typename Derived >
  bool _check_qubit_matrix (const Eigen::MatrixBase< Derived > &A) noexcept
- template<typename Derived >
  bool _check_qubit_cvector (const Eigen::MatrixBase< Derived > &V) noexcept
- template<typename Derived >
  bool _check_qubit_rvector (const Eigen::MatrixBase< Derived > &V) noexcept
- template<typename Derived >
  bool _check_qubit_vector (const Eigen::MatrixBase< Derived > &V) noexcept
- bool _check_perm (const std::vector< idx > &perm)
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > _kron2 (const Eigen::MatrixBase< Derived1 > &A, const Eigen::MatrixBase< Derived2 > &B)
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > _dirsum2 (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 utility functions, do not use/modify.

### 6.3.2 Function Documentation

#### 6.3.2.1 template<typename Derived > bool qpp::internal::_check_cvector ( const Eigen::MatrixBase< Derived > & *A* )

#### 6.3.2.2 bool qpp::internal::_check_dims ( const std::vector< idx > & *dims* ) [inline]

#### 6.3.2.3 template<typename Derived > bool qpp::internal::_check_dims_match_cvect ( const std::vector< idx > & *dims,* const Eigen::MatrixBase< Derived > & *V* )

#### 6.3.2.4 template<typename Derived > bool qpp::internal::_check_dims_match_mat ( const std::vector< idx > & *dims,* const Eigen::MatrixBase< Derived > & *A* )

#### 6.3.2.5 template<typename Derived > bool qpp::internal::_check_dims_match_rvect ( const std::vector< idx > & *dims,* const Eigen::MatrixBase< Derived > & *V* )

6.3.2.6 bool qpp::internal::_check_eq_dims ( const std::vector< idx > & *dims*, idx *dim* ) [inline], [noexcept]

6.3.2.7 template<typename T1 , typename T2 > bool qpp::internal::_check_matching_sizes ( const T1 & *lhs*, const T2 & *rhs* ) [noexcept]

6.3.2.8 template<typename T > bool qpp::internal::_check_nonzero_size ( const T & *x* ) [noexcept]

6.3.2.9 bool qpp::internal::_check_perm ( const std::vector< idx > & *perm* ) [inline]

6.3.2.10 template<typename Derived > bool qpp::internal::_check_qubit_cvector ( const Eigen::MatrixBase< Derived > & *V* ) [noexcept]

6.3.2.11 template<typename Derived > bool qpp::internal::_check_qubit_matrix ( const Eigen::MatrixBase< Derived > & *A* ) [noexcept]

6.3.2.12 template<typename Derived > bool qpp::internal::_check_qubit_rvector ( const Eigen::MatrixBase< Derived > & *V* ) [noexcept]

6.3.2.13 template<typename Derived > bool qpp::internal::_check_qubit_vector ( const Eigen::MatrixBase< Derived > & *V* ) [noexcept]

6.3.2.14 template<typename Derived > bool qpp::internal::_check_rvector ( const Eigen::MatrixBase< Derived > & *A* )

6.3.2.15 template<typename Derived > bool qpp::internal::_check_square_mat ( const Eigen::MatrixBase< Derived > & *A* )

6.3.2.16 bool qpp::internal::_check_subsys_match_dims ( const std::vector< idx > & *subsys*, const std::vector< idx > & *dims* ) [inline]

6.3.2.17 template<typename Derived > bool qpp::internal::_check_vector ( const Eigen::MatrixBase< Derived > & *A* )

6.3.2.18 template<typename Derived1 , typename Derived2 > dyn_mat<typename Derived1::Scalar> qpp::internal::_dirsum2 ( const Eigen::MatrixBase< Derived1 > & *A*, const Eigen::MatrixBase< Derived2 > & *B* )

6.3.2.19 template<typename Derived1 , typename Derived2 > dyn_mat<typename Derived1::Scalar> qpp::internal::_kron2 ( const Eigen::MatrixBase< Derived1 > & *A*, const Eigen::MatrixBase< Derived2 > & *B* )

6.3.2.20 idx qpp::internal::_multiidx2n ( const idx * *midx*, idx *numdims*, const idx * *dims* ) [inline], [noexcept]

6.3.2.21 void qpp::internal::_n2multiidx ( idx *n*, idx *numdims*, const idx * *dims*, idx * *result* ) [inline], [noexcept]

6.3.2.22 template<typename T > void qpp::internal::variadic_vector_emplace ( std::vector< T > & )

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

## 6.4 qpp::internal::_details Namespace Reference

### Classes

- struct _Display_Impl

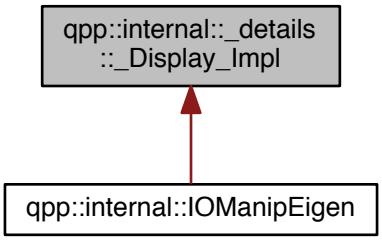
# Chapter 7

# Class Documentation

## 7.1 qpp::internal::_details::_Display_Impl Struct Reference

`#include <internal/classes/iomanip.h>`

Inheritance diagram for qpp::internal::_details::_Display_Impl:




### Public Member Functions

- template<typename T >
  std::ostream & _display_impl (const T &_A, std::ostream &_os, double _chop=qpp::chop) const

### 7.1.1 Member Function Documentation

#### 7.1.1.1 template<typename T > std::ostream& qpp::internal::_details::_Display_Impl::_display_impl ( const T & *_A*, std::ostream & *_os*, double *_chop* = qpp::chop ) const [inline]

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

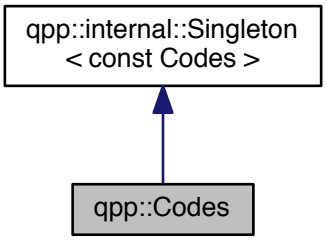
- internal/classes/iomanip.h

## 7.2 qpp::Codes Class Reference

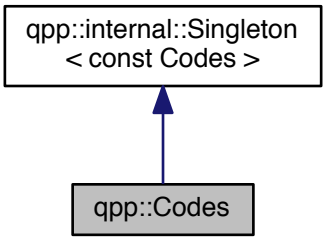
const Singleton class that defines quantum error correcting codes

`#include <classes/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, idx i) const

  *Returns the codeword of the specified code type.*

## Private Member Functions

- Codes ()

  *Default constructor.*
- ~Codes ()=default

  *Default destructor.*

### Friends

- class internal::Singleton< const Codes >

### Additional Inherited Members

## 7.2.1 Detailed Description

const Singleton class that defines quantum error correcting codes

## 7.2.2 Member Enumeration Documentation

#### 7.2.2.1 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

## 7.2.3 Constructor & Destructor Documentation

#### 7.2.3.1 qpp::Codes::Codes ( ) [inline], [private]

Default constructor.

#### 7.2.3.2 qpp::Codes::~Codes ( ) [private], [default]

Default destructor.

## 7.2.4 Member Function Documentation

#### 7.2.4.1 ket qpp::Codes::codeword ( Type *type*, idx *i* ) const [inline]

Returns the codeword of the specified code type.

**See also**

qpp::Codes::Type

**Parameters**

| | |
|---|---|
| *type* | Code type |
| *i* | Codeword index |

**Returns**

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

### 7.2.5 Friends And Related Function Documentation

#### 7.2.5.1 friend class internal::Singleton< const Codes > [friend]

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

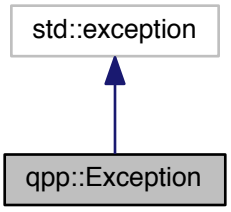
- classes/codes.h

## 7.3 qpp::Exception Class Reference

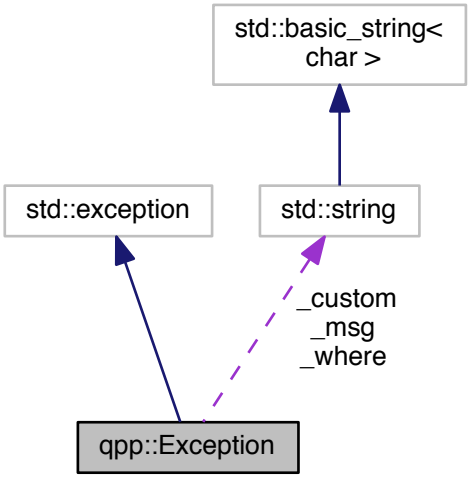
Generates custom exceptions, used when validating function parameters.

`#include <classes/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::PERM_MISMATCH_DIMS, Type::NOT_QUBIT_MATRIX,
  Type::NOT_QUBIT_CVECTOR, Type::NOT_QUBIT_RVECTOR, Type::NOT_QUBIT_VECTOR, Type::NOT_QUBIT_SUBSYS,
  Type::NOT_BIPARTITE, Type::NO_CODEWORD, Type::OUT_OF_RANGE, Type::TYPE_MISMATCH,
  Type::SIZE_MISMATCH, Type::UNDEFINED_TYPE, Type::CUSTOM_EXCEPTION }

  *Exception types, add more 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

- void _construct_exception_msg ()

  *Constructs the exception description from its type.*

### Private Attributes

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

## 7.3.1 Detailed Description

Generates custom exceptions, used when validating function parameters.

Customize this class if more exceptions are needed

## 7.3.2 Member Enumeration Documentation

### 7.3.2.1 enum qpp::Exception::Type [strong]

Exception types, add more here if needed.

**See also**

qpp::Exception::_construct_exception_msg()

**Enumerator**

***UNKNOWN_EXCEPTION*** Unknown exception

***ZERO_SIZE*** Zero sized object, e.g. empty Eigen::Matrix or std::vector<> with no elements

***MATRIX_NOT_SQUARE*** Eigen::Matrix is not square

***MATRIX_NOT_CVECTOR*** Eigen::Matrix is not a column vector

***MATRIX_NOT_RVECTOR*** Eigen::Matrix is not a row vector

***MATRIX_NOT_VECTOR*** Eigen::Matrix is not a row/column vector

***MATRIX_NOT_SQUARE_OR_CVECTOR*** Eigen::Matrix is not square nor a column vector

***MATRIX_NOT_SQUARE_OR_RVECTOR*** Eigen::Matrix is not square nor a row vector

***MATRIX_NOT_SQUARE_OR_VECTOR*** Eigen::Matrix is not square nor a row/column vector

***MATRIX_MISMATCH_SUBSYS*** Matrix size mismatch subsystem sizes (e.g. in qpp::apply())

***DIMS_INVALID*** std::vector<idx> of dimensions has zero size or contains zeros

***DIMS_NOT_EQUAL*** Local/global dimensions are not equal

***DIMS_MISMATCH_MATRIX*** Product of the elements of std::vector<idx> of dimensions is not equal to the number of rows of Eigen::Matrix (assumed to be a square matrix)

***DIMS_MISMATCH_CVECTOR*** Product of the elements of std::vector<idx> of dimensions is not equal to the number of elements of Eigen::Matrix (assumed to be a column vector)

***DIMS_MISMATCH_RVECTOR*** Product of the elements of std::vector<idx> of dimensions is not equal to the number of elements of Eigen::Matrix (assumed to be a row vector)

***DIMS_MISMATCH_VECTOR*** Product of the elements of std::vector<idx> of dimensions is not equal to the number of elements of Eigen::Matrix (assumed to be a row/column vector)

***SUBSYS_MISMATCH_DIMS*** std::vector<idx> of subsystem labels has duplicates, or has entries that are larger than the size of the std::vector<idx> of dimensions

***PERM_INVALID*** std::vector<idx> does note represent a valid permutation

***PERM_MISMATCH_DIMS*** Size of the std::vector<idx> representing the permutation is different from the size of the std::vector<idx> of dimensions

***NOT_QUBIT_MATRIX*** Eigen::Matrix is not 2 x 2

***NOT_QUBIT_CVECTOR*** Eigen::Matrix is not 2 x 1

***NOT_QUBIT_RVECTOR*** Eigen::Matrix is not 1 x 2

***NOT_QUBIT_VECTOR*** Eigen::Matrix is not 1 x 2 nor 2 x 1

***NOT_QUBIT_SUBSYS*** Subsystems are not 2-dimensional

***NOT_BIPARTITE*** std::vector<idx> of dimensions has size different from 2

***NO_CODEWORD*** Codeword does not exist, thrown when calling qpp::Codes::codeword() with invalid index *i*

***OUT_OF_RANGE*** Parameter out of range

***TYPE_MISMATCH*** Scalar types do not match

***SIZE_MISMATCH*** Sizes do not match

***UNDEFINED_TYPE*** Templated specialization not defined for this type

***CUSTOM_EXCEPTION*** Custom exception, user must provide a custom message

### 7.3.3 Constructor & Destructor Documentation

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

Constructs an exception.

**Parameters**

| | |
|---|---|
| *where* | Text representing where the exception occured |
| *type* | Exception type, defined in qpp::Exception::Type |

#### 7.3.3.2 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**

| | |
|---|---|
| *where* | Text representing where the exception occured |
| *custom* | Exception description |

### 7.3.4 Member Function Documentation

#### 7.3.4.1 void qpp::Exception::_construct_exception_msg ( ) [inline], [private]

Constructs the exception description from its type.

**See also**

qpp::Exception::Type

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

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

Overrides std::exception::what()

**Returns**

Exception description

### 7.3.5 Member Data Documentation

#### 7.3.5.1 std::string qpp::Exception::_custom [private]

#### 7.3.5.2 std::string qpp::Exception::_msg [private]

#### 7.3.5.3 Type qpp::Exception::_type [private]

#### 7.3.5.4 std::string qpp::Exception::_where [private]

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

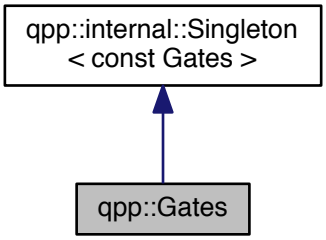
- classes/exception.h

## 7.4 qpp::Gates Class Reference

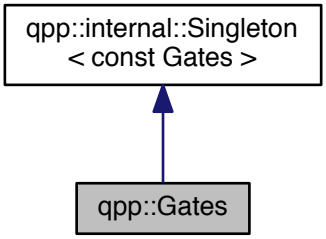
const Singleton class that implements most commonly used gates

`#include <classes/gates.h>`

Inheritance diagram for qpp::Gates:




Collaboration diagram for qpp::Gates:




## Public Member Functions

- cmat Rn (double theta, const std::vector< double > &n) const

  *Rotation of theta about the 3-dimensional real unit vector n.*
- cmat Zd (idx D) const

  *Generalized Z gate for qudits.*
- cmat Fd (idx D) const

  *Fourier transform gate for qudits.*
- cmat Xd (idx D) const

  *Generalized X gate for qudits.*
- template<typename Derived = Eigen::MatrixXcd>
  Derived Id (idx D) const

  *Identity gate.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > CTRL (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &ctrl, const std::vector< idx > &subsys, idx n, idx d=2) const

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

- template<typename Derived >
  dyn_mat< typename Derived::Scalar > expandout (const Eigen::MatrixBase< Derived > &A, idx pos, const std::vector< idx > &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 CNOT {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.*
- ~Gates ()=default

  *Default destructor.*

### Friends

- class internal::Singleton< const Gates >

### Additional Inherited Members

#### 7.4.1 Detailed Description

const Singleton class that implements most commonly used gates

### 7.4.2 Constructor & Destructor Documentation

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

Initializes the gates.

#### 7.4.2.2 qpp::Gates::~Gates ( ) [private], [default]

Default destructor.

### 7.4.3 Member Function Documentation

#### 7.4.3.1 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::Gates::CTRL ( const Eigen::MatrixBase< Derived > & *A*, const std::vector< idx > & *ctrl*, const std::vector< idx > & *subsys*, idx *n*, idx *d* = 2 ) const [inline]

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

**See also**

qpp::applyCTRL()

**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* | Subsystem dimensions |

**Returns**

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

#### 7.4.3.2 template<typename Derived > dyn_mat<typename Derived::Scalar> qpp::Gates::expandout ( const Eigen::MatrixBase< Derived > & *A*, idx *pos*, const std::vector< idx > & *dims* ) const [inline]

Expands out.

**See also**

qpp::kron()

Expands out *A* as a matrix in a multi-partite system. Faster than using qpp::kron(I, I, ..., I, A, I, ..., I).

**Parameters**

| | |
|---|---|
| *A* | Eigen expression |

| | |
|---|---|
| *pos* | Position |
| *dims* | 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*

#### 7.4.3.3 cmat qpp::Gates::Fd ( idx *D* ) const [inline]

Fourier transform gate for qudits.

**Note**

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

**Parameters**

| | |
|---|---|
| *D* | Dimension of the Hilbert space |

**Returns**

Fourier transform gate for qudits

#### 7.4.3.4 template<typename Derived = Eigen::MatrixXcd> Derived qpp::Gates::Id ( idx *D* ) const [inline]

Identity gate.

**Note**

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

**Parameters**

| | |
|---|---|
| *D* | Dimension of the Hilbert space |

**Returns**

Identity gate

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

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

**Parameters**

| | |
|---|---|
| *theta* | Rotation angle |
| *n* | 3-dimensional real unit vector |

**Returns**

Rotation gate

#### 7.4.3.6 cmat qpp::Gates::Xd ( idx *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

#### 7.4.3.7 cmat qpp::Gates::Zd ( idx *D* ) const [inline]

Generalized Z gate for qudits.

**Note**

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

**Parameters**

| | |
|---|---|
| *D* | Dimension of the Hilbert space |

**Returns**

Generalized Z gate for qudits

### 7.4.4 Friends And Related Function Documentation

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

### 7.4.5 Member Data Documentation

#### 7.4.5.1 cmat qpp::Gates::CNOT {cmat::Identity(4, 4)}

Controlled-NOT control target gate.

#### 7.4.5.2 cmat qpp::Gates::CNOTba {cmat::Zero(4, 4)}

Controlled-NOT target control gate.

#### 7.4.5.3 cmat qpp::Gates::CZ {cmat::Identity(4, 4)}

Controlled-Phase gate.

#### 7.4.5.4 cmat qpp::Gates::FRED {cmat::Identity(8, 8)}

Fredkin gate.

#### 7.4.5.5 cmat qpp::Gates::H {cmat::Zero(2, 2)}

Hadamard gate.

#### 7.4.5.6 cmat qpp::Gates::Id2 {cmat::Identity(2, 2)}

Identity gate.

#### 7.4.5.7 cmat qpp::Gates::S {cmat::Zero(2, 2)}

S gate.

#### 7.4.5.8 cmat qpp::Gates::SWAP {cmat::Identity(4, 4)}

SWAP gate.

#### 7.4.5.9 cmat qpp::Gates::T {cmat::Zero(2, 2)}

T gate.

#### 7.4.5.10 cmat qpp::Gates::TOF {cmat::Identity(8, 8)}

Toffoli gate.

#### 7.4.5.11 cmat qpp::Gates::X {cmat::Zero(2, 2)}

Pauli Sigma-X gate.

#### 7.4.5.12 cmat qpp::Gates::Y {cmat::Zero(2, 2)}

Pauli Sigma-Y gate.

#### 7.4.5.13 cmat qpp::Gates::Z {cmat::Zero(2, 2)}

Pauli Sigma-Z gate.

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

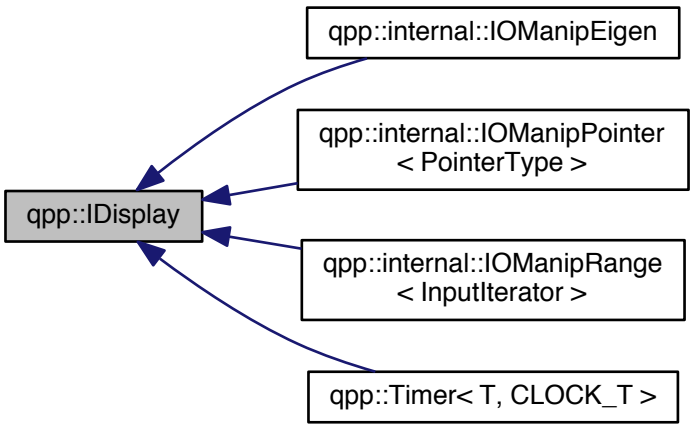
- classes/gates.h

## 7.5 qpp::IDisplay Class Reference

Abstract class (interface) that mandates the definition of virtual std::ostream& display(std::ostream& os) const.

`#include <classes/idisplay.h>`

Inheritance diagram for qpp::IDisplay:




## Public Member Functions

- IDisplay ()=default

  *Default constructor.*
- IDisplay (const IDisplay &)=default

  *Default copy constructor.*
- IDisplay (IDisplay &&)=default

  *Default move constructor.*
- IDisplay & operator= (const IDisplay &)=default

  *Default copy assignment operator.*
- IDisplay & operator= (IDisplay &&)=default

  *Default move assignment operator.*
- virtual ~IDisplay ()=default

  *Default virtual destructor.*

## Private Member Functions

- virtual std::ostream & display (std::ostream &os) const =0

  *Must be overridden by all derived classes.*

## Friends

- std::ostream & operator<< (std::ostream &os, const IDisplay &rhs)

  *Overloads the extraction operator.*

### 7.5.1 Detailed Description

Abstract class (interface) that mandates the definition of virtual std::ostream& display(std::ostream& os) const.

This class defines friend inline std::ostream& operator<< (std::ostream& os, const qpp::IDisplay& rhs). The latter delegates the work to the pure private virtual function qpp::IDisplay::display() which has to be overridden by all derived classes.

### 7.5.2 Constructor & Destructor Documentation

#### 7.5.2.1 qpp::IDisplay::IDisplay ( ) [default]

Default constructor.

#### 7.5.2.2 qpp::IDisplay::IDisplay ( const IDisplay & ) [default]

Default copy constructor.

#### 7.5.2.3 qpp::IDisplay::IDisplay ( IDisplay && ) [default]

Default move constructor.

#### 7.5.2.4 virtual qpp::IDisplay::~IDisplay ( ) [virtual], [default]

Default virtual destructor.

### 7.5.3 Member Function Documentation

#### 7.5.3.1 virtual std::ostream& qpp::IDisplay::display ( std::ostream & *os* ) const [private], [pure virtual]

Must be overridden by all derived classes.

The actual stream extraction processing is performed by the overriden member function in the derived class. This function is automatically invoked by friend inline std::ostream& operator<<(std::ostream& os, const IDisplay& rhs).

Implemented in qpp::internal::IOManipEigen, qpp::internal::IOManipPointer< PointerType >, qpp::internal::IOManipRange< InputIterator >, and qpp::Timer< T, CLOCK_T >.

#### 7.5.3.2 IDisplay& qpp::IDisplay::operator= ( const IDisplay & ) [default]

Default copy assignment operator.

#### 7.5.3.3 IDisplay& qpp::IDisplay::operator= ( IDisplay && ) [default]

Default move assignment operator.

### 7.5.4 Friends And Related Function Documentation

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

Overloads the extraction operator.

Delegates the work to the virtual function qpp::IDisplay::display()

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

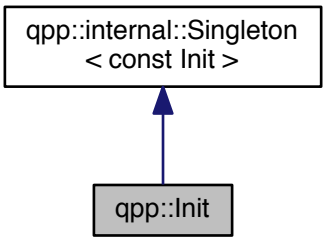
- classes/idisplay.h

## 7.6 qpp::Init Class Reference

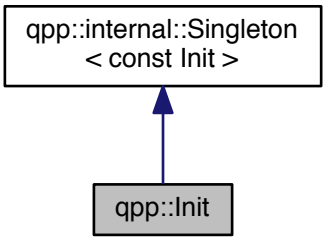
const Singleton class that performs additional initializations/cleanups

`#include <classes/init.h>`

Inheritance diagram for qpp::Init:




Collaboration diagram for qpp::Init:




### Private Member Functions

- Init ()
  *Additional initializations.*
- ~Init ()
  *Cleanups.*

### Friends

- class internal::Singleton< const Init >

### Additional Inherited Members

### 7.6.1 Detailed Description

const Singleton class that performs additional initializations/cleanups

### 7.6.2 Constructor & Destructor Documentation

#### 7.6.2.1 qpp::Init::Init ( ) [inline], [private]

Additional initializations.

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

Cleanups.

### 7.6.3 Friends And Related Function Documentation

#### 7.6.3.1 friend class internal::Singleton< const Init > [friend]

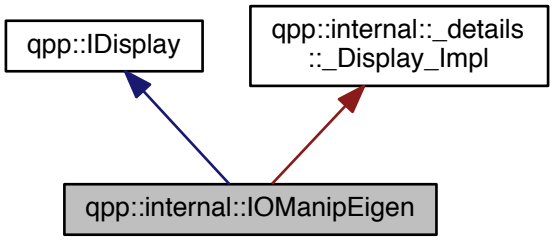
The documentation for this class was generated from the following file:

- classes/init.h

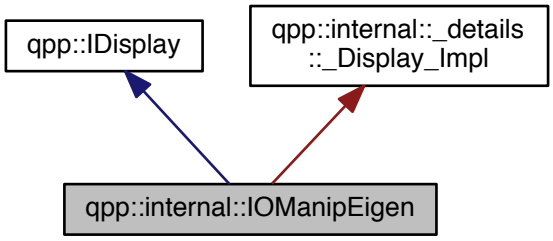
## 7.7 qpp::internal::IOManipEigen Class Reference

`#include <internal/classes/iomanip.h>`

Inheritance diagram for qpp::internal::IOManipEigen:



Collaboration diagram for qpp::internal::IOManipEigen:




### Public Member Functions

- template<typename Derived >
  IOManipEigen (const Eigen::MatrixBase< Derived > &A, double chop=qpp::chop)
- IOManipEigen (const cplx z, double chop=qpp::chop)

### Private Member Functions

- std::ostream & display (std::ostream &os) const override

  *Must be overridden by all derived classes.*

### Private Attributes

- cmat _A
- double _chop

## 7.7.1 Constructor & Destructor Documentation

#### 7.7.1.1 template<typename Derived > qpp::internal::IOManipEigen::IOManipEigen ( const Eigen::MatrixBase< Derived > & *A*, double *chop* = qpp::chop ) [inline], [explicit]

#### 7.7.1.2 qpp::internal::IOManipEigen::IOManipEigen ( const cplx *z*, double *chop* = qpp::chop ) [inline], [explicit]

## 7.7.2 Member Function Documentation

#### 7.7.2.1 std::ostream& qpp::internal::IOManipEigen::display ( std::ostream & *os* ) const [inline], [override], [private], [virtual]

Must be overridden by all derived classes.

The actual stream extraction processing is performed by the overriden member function in the derived class. This function is automatically invoked by friend inline std::ostream& operator<<(std::ostream& os, const IDisplay& rhs).

Implements qpp::IDisplay.

### 7.7.3 Member Data Documentation

#### 7.7.3.1 cmat qpp::internal::IOManipEigen::_A [private]

#### 7.7.3.2 double qpp::internal::IOManipEigen::_chop [private]

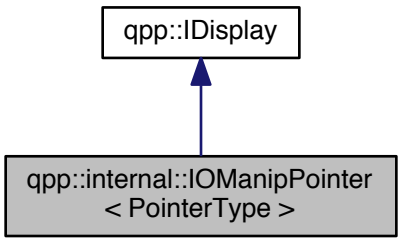
The documentation for this class was generated from the following file:

- internal/classes/iomanip.h

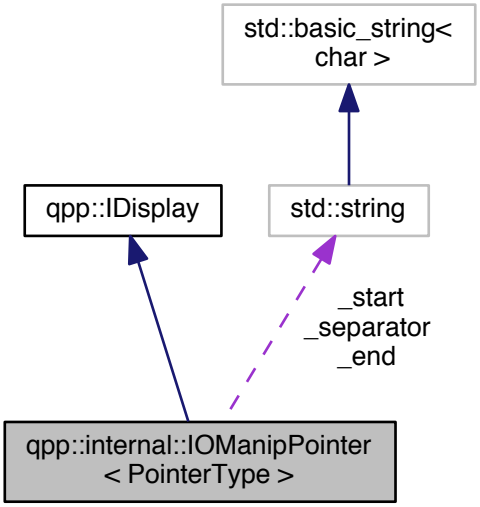
## 7.8 qpp::internal::IOManipPointer< PointerType > Class Template Reference

`#include <internal/classes/iomanip.h>`

Inheritance diagram for qpp::internal::IOManipPointer< PointerType >:




Collaboration diagram for qpp::internal::IOManipPointer< PointerType >:



### Public Member Functions

- IOManipPointer (const PointerType *p, idx n, const std::string &separator, const std::string &start="[", const std::string &end="]")
- IOManipPointer (const IOManipPointer &)=default
- IOManipPointer & operator= (const IOManipPointer &)=default

### Private Member Functions

- std::ostream & display (std::ostream &os) const override

  *Must be overridden by all derived classes.*

### Private Attributes

- const PointerType * _p
- idx _n
- std::string _separator
- std::string _start
- std::string _end

## 7.8.1 Constructor & Destructor Documentation

### 7.8.1.1 template<typename PointerType> qpp::internal::IOManipPointer< PointerType >::IOManipPointer ( const PointerType * *p*, idx *n*, const std::string & *separator*, const std::string & *start* = "[", const std::string & *end* = "]" ) [inline], [explicit]

### 7.8.1.2 template<typename PointerType> qpp::internal::IOManipPointer< PointerType >::IOManipPointer ( const IOManipPointer< PointerType > & ) [default]

## 7.8.2 Member Function Documentation

### 7.8.2.1 template<typename PointerType> std::ostream& qpp::internal::IOManipPointer< PointerType >::display ( std::ostream & *os* ) const [inline], [override], [private], [virtual]

Must be overridden by all derived classes.

The actual stream extraction processing is performed by the overriden member function in the derived class. This function is automatically invoked by friend inline std::ostream& operator<<(std::ostream& os, const IDisplay& rhs).

Implements qpp::IDisplay.

### 7.8.2.2 template<typename PointerType> IOManipPointer& qpp::internal::IOManipPointer< PointerType >::operator= ( const IOManipPointer< PointerType > & ) [default]

## 7.8.3 Member Data Documentation

### 7.8.3.1 template<typename PointerType> std::string qpp::internal::IOManipPointer< PointerType >::_end [private]

### 7.8.3.2 template<typename PointerType> idx qpp::internal::IOManipPointer< PointerType >::_n [private]

### 7.8.3.3 template<typename PointerType> const PointerType* qpp::internal::IOManipPointer< PointerType >::_p [private]

#### 7.8.3.4 template<typename PointerType> std::string qpp::internal::IOManipPointer< PointerType >::_separator [private]

#### 7.8.3.5 template<typename PointerType> std::string qpp::internal::IOManipPointer< PointerType >::_start [private]

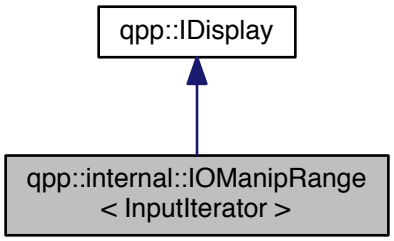
The documentation for this class was generated from the following file:

- internal/classes/iomanip.h

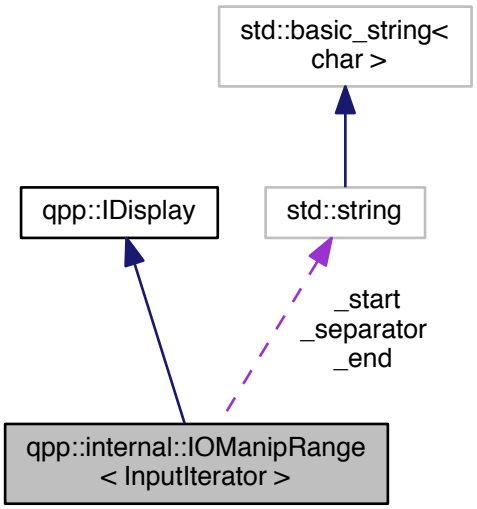
## 7.9 qpp::internal::IOManipRange< InputIterator > Class Template Reference

`#include <internal/classes/iomanip.h>`

Inheritance diagram for qpp::internal::IOManipRange< InputIterator >:




Collaboration diagram for qpp::internal::IOManipRange< InputIterator >:



### Public Member Functions

- IOManipRange (InputIterator first, InputIterator last, const std::string &separator, const std::string &start="[", const std::string &end="]")
- IOManipRange (const IOManipRange &)=default
- IOManipRange & operator= (const IOManipRange &)=default

### Private Member Functions

- std::ostream & display (std::ostream &os) const override

  *Must be overridden by all derived classes.*

### Private Attributes

- InputIterator _first
- InputIterator _last
- std::string _separator
- std::string _start
- std::string _end

## 7.9.1 Constructor & Destructor Documentation

### 7.9.1.1 template<typename InputIterator> qpp::internal::IOManipRange< InputIterator >::IOManipRange ( InputIterator *first,* InputIterator *last,* const std::string & *separator,* const std::string & *start* = "[", const std::string & *end* = "]" ) [inline], [explicit]

### 7.9.1.2 template<typename InputIterator> qpp::internal::IOManipRange< InputIterator >::IOManipRange ( const IOManipRange< InputIterator > & ) [default]

## 7.9.2 Member Function Documentation

### 7.9.2.1 template<typename InputIterator> std::ostream& qpp::internal::IOManipRange< InputIterator >::display ( std::ostream & *os* ) const [inline], [override], [private], [virtual]

Must be overridden by all derived classes.

The actual stream extraction processing is performed by the overriden member function in the derived class. This function is automatically invoked by friend inline std::ostream& operator<<(std::ostream& os, const IDisplay& rhs).

Implements qpp::IDisplay.

### 7.9.2.2 template<typename InputIterator> IOManipRange& qpp::internal::IOManipRange< InputIterator >::operator= ( const IOManipRange< InputIterator > & ) [default]

## 7.9.3 Member Data Documentation

### 7.9.3.1 template<typename InputIterator> std::string qpp::internal::IOManipRange< InputIterator >::_end [private]

### 7.9.3.2 template<typename InputIterator> InputIterator qpp::internal::IOManipRange< InputIterator >::_first [private]

### 7.9.3.3 template<typename InputIterator> InputIterator qpp::internal::IOManipRange< InputIterator >::_last [private]

#### 7.9.3.4 template<typename InputIterator> std::string qpp::internal::IOManipRange< InputIterator >::_separator [private]

#### 7.9.3.5 template<typename InputIterator> std::string qpp::internal::IOManipRange< InputIterator >::_start [private]

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

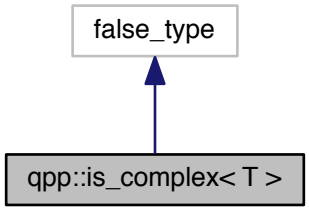
- internal/classes/iomanip.h

## 7.10 qpp::is_complex< T > Struct Template Reference

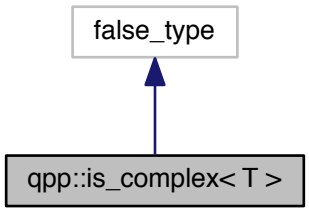
Checks whether the type is a complex type.

`#include <traits.h>`

Inheritance diagram for qpp::is_complex< T >:




Collaboration diagram for qpp::is_complex< T >:




### 7.10.1 Detailed Description

template<typename T>struct qpp::is_complex< T >

Checks whether the type is a complex type.

Provides the member constant *value* which is equal to *true*, if the type is a complex type (i.e. *std::complex<T>*)

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

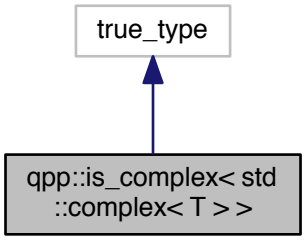
- traits.h

## 7.11 qpp::is_complex< std::complex< T > > Struct Template Reference

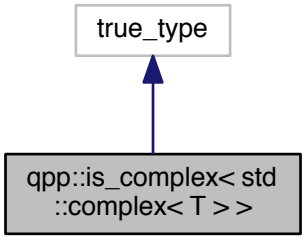
Checks whether the type is a complex number type, specialization for complex types.

`#include <traits.h>`

Inheritance diagram for qpp::is_complex< std::complex< T > >:




Collaboration diagram for qpp::is_complex< std::complex< T > >:




### 7.11.1 Detailed Description

**template<typename T>struct qpp::is_complex< std::complex< T > >**

Checks whether the type is a complex number type, specialization for complex types.

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

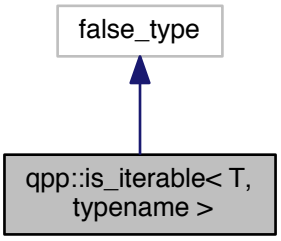
- traits.h

## 7.12 qpp::is_iterable< T, typename > Struct Template Reference

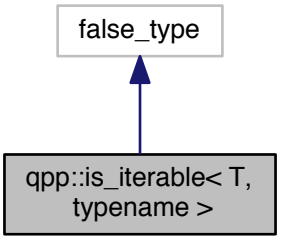
Checks whether *T* is compatible with an STL-like iterable container.

`#include <traits.h>`

Inheritance diagram for qpp::is_iterable< T, typename >:




Collaboration diagram for qpp::is_iterable< T, typename >:




### 7.12.1 Detailed Description

**template<typename T, typename = void>struct qpp::is_iterable< T, typename >**

Checks whether *T* is compatible with an STL-like iterable container.

Provides the member constant *value* which is equal to *true*, if *T* is compatible with an iterable container, i.e. provides at least *begin()* and *end()* member functions. Otherwise, *value* is equal to *false*.

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

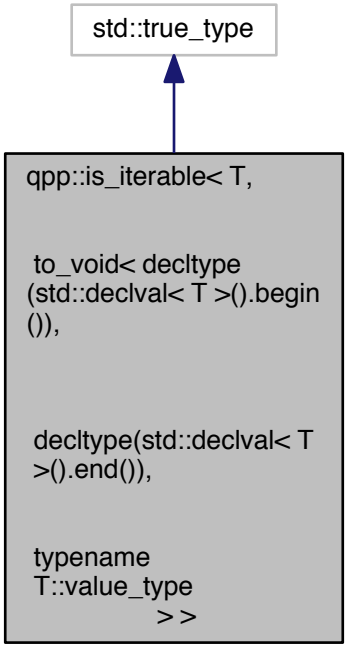
- traits.h

## 7.13 qpp::is_iterable< T, to_void< decltype(std::declval< T >().begin()), decltype(std::declval< T >().end()), typename T::value_type > > Struct Template Reference

Checks whether *T* is compatible with an STL-like iterable container, specialization for STL-like iterable containers.

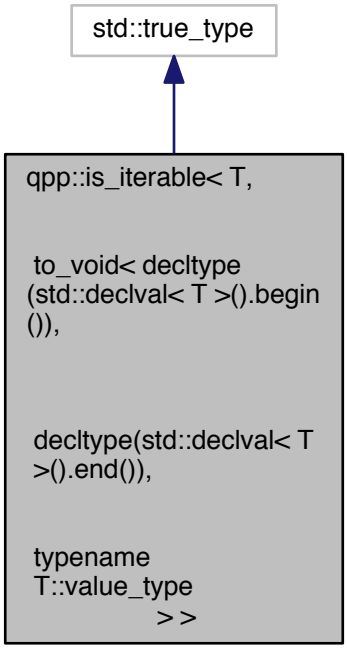
`#include <traits.h>`

Inheritance diagram for qpp::is_iterable< T, to_void< decltype(std::declval< T >().begin()), decltype(std::declval< T >().end()), typename T::value_type > >:




Collaboration diagram for qpp::is_iterable< T, to_void< decltype(std::declval< T >().begin()), decltype(std::

:declval< T >().end()), typename T::value_type > >:




### 7.13.1 Detailed Description

**template<typename T>struct qpp::is_iterable< T, to_void< decltype(std::declval< T >().begin()), decltype(std::declval< T >().end()), typename T::value_type > >**

Checks whether *T* is compatible with an STL-like iterable container, specialization for STL-like iterable containers.

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

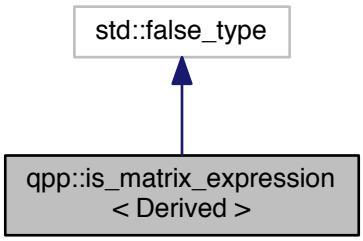
- traits.h

## 7.14 qpp::is_matrix_expression< Derived > Struct Template Reference

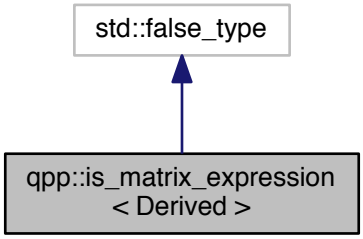
Checks whether the type is an Eigen matrix expression.

`#include <traits.h>`

Inheritance diagram for qpp::is_matrix_expression< Derived >:




Collaboration diagram for qpp::is_matrix_expression< Derived >:




### 7.14.1 Detailed Description

**template<typename Derived>struct qpp::is_matrix_expression< Derived >**

Checks whether the type is an Eigen matrix expression.

Provides the member constant *value* which is equal to *true*, if the type is an Eigen matrix expression of type *Eigen::MatrixBase<Derived>*. Otherwise, *value* is equal to *false*.

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

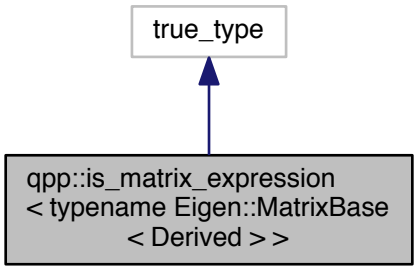
- traits.h

## 7.15 qpp::is_matrix_expression< typename Eigen::MatrixBase< Derived > > Struct Template Reference

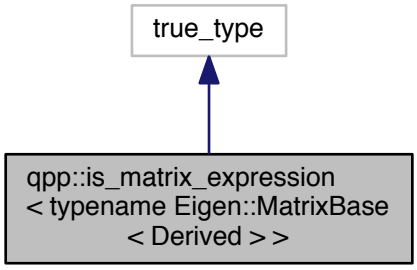
Checks whether the type is an Eigen matrix expression, specialization for Eigen matrix expressions.

`#include <traits.h>`

Inheritance diagram for qpp::is_matrix_expression< typename Eigen::MatrixBase< Derived > >:




Collaboration diagram for qpp::is_matrix_expression< typename Eigen::MatrixBase< Derived > >:




### 7.15.1 Detailed Description

**template<typename Derived>struct qpp::is_matrix_expression< typename Eigen::MatrixBase< Derived > >**

Checks whether the type is an Eigen matrix expression, specialization for Eigen matrix expressions.

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

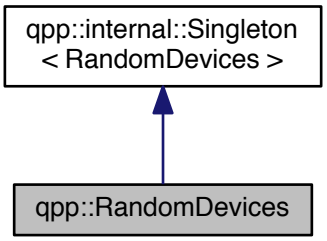
- traits.h

## 7.16 qpp::RandomDevices Class Reference

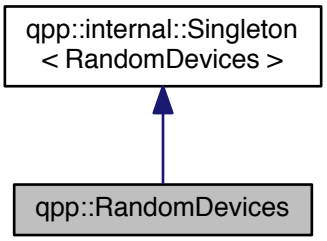
Singeleton class that manages the source of randomness in the library.

`#include <classes/random_devices.h>`

Inheritance diagram for qpp::RandomDevices:




Collaboration diagram for qpp::RandomDevices:




## Public Attributes

- std::mt19937 _rng

  *Mersenne twister random number generator.*

## Private Member Functions

- RandomDevices ()

  *Initializes and seeds the random number generators.*
- ~RandomDevices ()=default

  *Default destructor.*

## Private Attributes

- std::random_device _rd

  *used to seed std::mt19937 _rng*

### Friends

- class internal::Singleton< RandomDevices >

### Additional Inherited Members

### 7.16.1 Detailed Description

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

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.

#### Warning

This class DOES NOT seed the standard C number generator used by Eigen::Matrix::Random(), since it is not thread safe. Do not use Eigen::Matrix::Random() or functions that depend on the C style random number engine, but use qpp::rand() instead!

### 7.16.2 Constructor & Destructor Documentation

#### 7.16.2.1 qpp::RandomDevices::RandomDevices ( ) [inline], [private]

Initializes and seeds the random number generators.

#### 7.16.2.2 qpp::RandomDevices::~RandomDevices ( ) [private], [default]

Default destructor.

### 7.16.3 Friends And Related Function Documentation

#### 7.16.3.1 friend class internal::Singleton< RandomDevices > [friend]

### 7.16.4 Member Data Documentation

#### 7.16.4.1 std::random_device qpp::RandomDevices::_rd [private]

used to seed std::mt19937 _rng

#### 7.16.4.2 std::mt19937 qpp::RandomDevices::_rng

Mersenne twister random number generator.

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

- classes/random_devices.h

## 7.17 qpp::internal::Singleton< T > Class Template Reference

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

```
#include <internal/classes/singleton.h>
```

### Static Public Member Functions

- static T & get_instance () noexcept(std::is_nothrow_constructible< T >::value)
- static T & get_thread_local_instance () noexcept(std::is_nothrow_constructible< T >::value)

### Protected Member Functions

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

## 7.17.1 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 and destructor of your class as private. To get an instance, use the static member function qpp::internal::Singleton::get_instance() (qpp::internal::Singleton::get_thread_local_instance()), which returns a reference (thread_local 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()
    {
        // Implement the destructor here
    }
};

MySingleton& mySingleton = MySingleton::get_instance(); // Get an instance
thread_local MySingleton& tls = MySingleton::get_thread_local_instance();
// Get a thread_local instance
```

**See also**

Code of qpp::Codes, qpp::Gates, qpp::Init, qpp::RandomDevices, qpp::States or qpp.h for real world examples of usage.

## 7.17.2 Constructor & Destructor Documentation

### 7.17.2.1 template<typename T> qpp::internal::Singleton< T >::Singleton ( ) [protected], [default], [noexcept]

### 7.17.2.2 template<typename T> qpp::internal::Singleton< T >::Singleton ( const Singleton< T > & ) [protected], [delete]

### 7.17.2.3 template<typename T> virtual qpp::internal::Singleton< T >::~Singleton ( ) [protected], [virtual], [default]

### 7.17.3 Member Function Documentation

#### 7.17.3.1 template<typename T> static T& qpp::internal::Singleton< T >::get_instance ( ) [inline], [static], [noexcept]

#### 7.17.3.2 template<typename T> static T& qpp::internal::Singleton< T >::get_thread_local_instance ( ) [inline], [static], [noexcept]

#### 7.17.3.3 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:

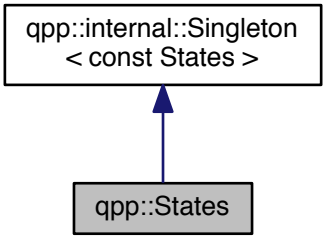
- internal/classes/singleton.h

## 7.18 qpp::States Class Reference

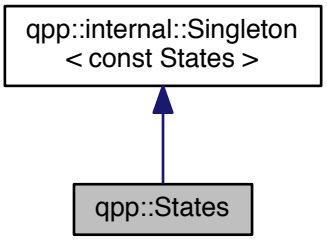
const Singleton class that implements most commonly used states

`#include <classes/states.h>`

Inheritance diagram for qpp::States:




Collaboration diagram for qpp::States:



## Public Attributes

- ket x0 {ket::Zero(2)}

  *Pauli Sigma-X 0-eigenstate |+>*
- ket x1 {ket::Zero(2)}

  *Pauli Sigma-X 1-eigenstate |->*
- ket y0 {ket::Zero(2)}

  *Pauli Sigma-Y 0-eigenstate |y+>*
- ket y1 {ket::Zero(2)}

  *Pauli Sigma-Y 1-eigenstate |y->*
- ket z0 {ket::Zero(2)}

  *Pauli Sigma-Z 0-eigenstate |0>*
- ket z1 {ket::Zero(2)}

  *Pauli Sigma-Z 1-eigenstate |1>*
- cmat px0 {cmat::Zero(2, 2)}

  *Projector onto the Pauli Sigma-X 0-eigenstate |+><+|.*
- cmat px1 {cmat::Zero(2, 2)}

  *Projector onto the Pauli Sigma-X 1-eigenstate |-><-|.*
- cmat py0 {cmat::Zero(2, 2)}

  *Projector onto the Pauli Sigma-Y 0-eigenstate |y+><y+|.*
- cmat py1 {cmat::Zero(2, 2)}

  *Projector onto the Pauli Sigma-Y 1-eigenstate |y-><y-|.*
- cmat pz0 {cmat::Zero(2, 2)}

  *Projector onto the Pauli Sigma-Z 0-eigenstate |0><0|.*
- cmat pz1 {cmat::Zero(2, 2)}

  *Projector onto the Pauli Sigma-Z 1-eigenstate |1><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 ()
- ∼States ()=default

  *Default destructor.*

### Friends

- class internal::Singleton< const States >

### Additional Inherited Members

## 7.18.1 Detailed Description

const Singleton class that implements most commonly used states

## 7.18.2 Constructor & Destructor Documentation

#### 7.18.2.1 qpp::States::States ( ) [inline], [private]

Initialize the states

#### 7.18.2.2 qpp::States::∼States ( ) [private], [default]

Default destructor.

## 7.18.3 Friends And Related Function Documentation

#### 7.18.3.1 friend class internal::Singleton< const States > [friend]

## 7.18.4 Member Data Documentation

#### 7.18.4.1 ket qpp::States::b00 {ket::Zero(4)}

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

#### 7.18.4.2 ket qpp::States::b01 {ket::Zero(4)}

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

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

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

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

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

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

GHZ state.

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

Projector onto the Bell-00 state.

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

Projector onto the Bell-01 state.

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

Projector onto the Bell-10 state.

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

Projector onto the Bell-11 state.

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

Projector onto the GHZ state.

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

Projector onto the W state.

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

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

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

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

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

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

#### 7.18.4.15 cmat qpp::States::py1 {cmat::Zero(2, 2)}

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

#### 7.18.4.16 cmat qpp::States::pz0 {cmat::Zero(2, 2)}

Projector onto the Pauli Sigma-Z 0-eigenstate $|0\rangle\langle0|$.

#### 7.18.4.17 cmat qpp::States::pz1 {cmat::Zero(2, 2)}

Projector onto the Pauli Sigma-Z 1-eigenstate |1><1|.

#### 7.18.4.18 ket qpp::States::W {ket::Zero(8)}

W state.

#### 7.18.4.19 ket qpp::States::x0 {ket::Zero(2)}

Pauli Sigma-X 0-eigenstate |+>

#### 7.18.4.20 ket qpp::States::x1 {ket::Zero(2)}

Pauli Sigma-X 1-eigenstate |->

#### 7.18.4.21 ket qpp::States::y0 {ket::Zero(2)}

Pauli Sigma-Y 0-eigenstate |y+>

#### 7.18.4.22 ket qpp::States::y1 {ket::Zero(2)}

Pauli Sigma-Y 1-eigenstate |y->

#### 7.18.4.23 ket qpp::States::z0 {ket::Zero(2)}

Pauli Sigma-Z 0-eigenstate |0>

#### 7.18.4.24 ket qpp::States::z1 {ket::Zero(2)}

Pauli Sigma-Z 1-eigenstate |1>

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

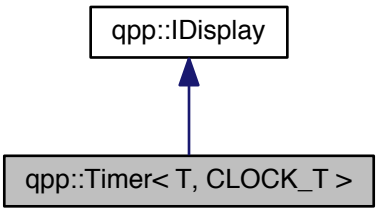
- classes/states.h

## 7.19 qpp::Timer< T, CLOCK_T > Class Template Reference

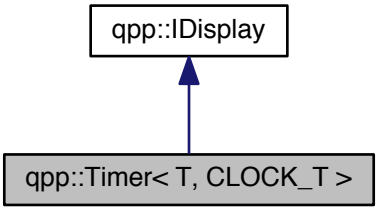
Chronometer.

`#include <classes/timer.h>`

Inheritance diagram for qpp::Timer< T, CLOCK_T >:




Collaboration diagram for qpp::Timer< T, CLOCK_T >:




## Public Member Functions

- Timer () noexcept

  *Constructs an instance with the current time as the starting point.*
- void tic () noexcept

  *Resets the chronometer.*
- const Timer & toc () noexcept

  *Stops the chronometer.*
- double tics () const noexcept

  *Time passed in the duration specified by T.*
- template<typename U = T>
  U get_duration () const noexcept

  *Duration specified by U.*
- Timer (const Timer &)=default

  *Default copy constructor.*
- Timer (Timer &&)=default

  *Default move constructor.*
- Timer & operator= (const Timer &)=default

  *Default copy assignment operator.*
- Timer & operator= (Timer &&)=default

*Default move assignment operator.*

- virtual ~Timer ()=default

  *Default virtual destructor.*

### Protected Attributes

- CLOCK_T::time_point _start
- CLOCK_T::time_point _end

### Private Member Functions

- std::ostream & display (std::ostream &os) const override

  *qpp::IDisplay::display() override*

## 7.19.1 Detailed Description

**template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock>class qpp::Timer< T, CLOCK_T >**

Chronometer.

**Template Parameters**

| | |
|---|---|
| *T* | Tics duration, default is std::chrono::duration<double, 1> i.e. seconds in double precision |
| *CLOCK_T* | Clock's type, default is std::chrono::steady_clock, not affected by wall clock changes during runtime |

## 7.19.2 Constructor & Destructor Documentation

### 7.19.2.1 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> qpp::Timer< T, CLOCK_T >::Timer ( ) [inline], [noexcept]

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

### 7.19.2.2 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> qpp::Timer< T, CLOCK_T >::Timer ( const Timer< T, CLOCK_T > & ) [default]

Default copy constructor.

### 7.19.2.3 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> qpp::Timer< T, CLOCK_T >::Timer ( Timer< T, CLOCK_T > && ) [default]

Default move constructor.

### 7.19.2.4 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> virtual qpp::Timer< T, CLOCK_T >::~Timer ( ) [virtual], [default]

Default virtual destructor.

### 7.19.3 Member Function Documentation

#### 7.19.3.1 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> std::ostream& qpp::Timer< T, CLOCK_T >::display ( std::ostream & *os* ) const [inline], [override], [private], [virtual]

qpp::IDisplay::display() override

**Parameters**

| | |
|---|---|
| *os* | Output stream |

**Returns**

Writes to the output stream the number of tics (specified by T) that passed between the instantiation/reset and invocation of qpp::Timer::toc().

Implements qpp::IDisplay.

#### 7.19.3.2 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> template<typename U = T> U qpp::Timer< T, CLOCK_T >::get_duration ( ) const [inline], [noexcept]

Duration specified by U.

**Template Parameters**

| | |
|---|---|
| *U* | Duration, default is T, which defaults to std::chrono::duration<double, 1>, i.e. seconds in double precision |

**Returns**

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

#### 7.19.3.3 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> Timer& qpp::Timer< T, CLOCK_T >::operator= ( const Timer< T, CLOCK_T > & ) [default]

Default copy assignment operator.

#### 7.19.3.4 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> Timer& qpp::Timer< T, CLOCK_T >::operator= ( Timer< T, CLOCK_T > && ) [default]

Default move assignment operator.

#### 7.19.3.5 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> void qpp::Timer< T, CLOCK_T >::tic ( ) [inline], [noexcept]

Resets the chronometer.

Resets the starting/ending point to the current time

#### 7.19.3.6 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> double qpp::Timer< T, CLOCK_T >::tics ( ) const [inline], [noexcept]

Time passed in the duration specified by T.

**Returns**

Number of tics (specified by T) that passed between the instantiation/reset and invocation of qpp::Timer::toc()

#### 7.19.3.7 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> const Timer& qpp::Timer< T, CLOCK_T >::toc ( ) [inline], [noexcept]

Stops the chronometer.

Set the current time as the ending point

**Returns**

Current instance

### 7.19.4 Member Data Documentation

#### 7.19.4.1 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> CLOCK_T::time_point qpp::Timer< T, CLOCK_T >::_end [protected]

#### 7.19.4.2 template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock> CLOCK_T::time_point qpp::Timer< T, CLOCK_T >::_start [protected]

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

- classes/timer.h

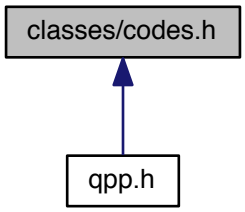
# Chapter 8

# File Documentation

## 8.1 classes/codes.h File Reference

Quantum error correcting codes.

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
  *Quantum++ main namespace.*

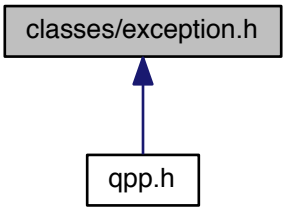
### 8.1.1 Detailed Description

Quantum error correcting codes.

## 8.2 classes/exception.h File Reference

Exceptions.

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

  *Quantum++ main namespace.*

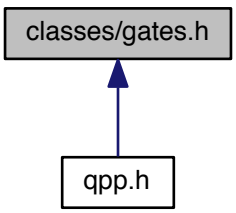
#### 8.2.1 Detailed Description

Exceptions.

## 8.3 classes/gates.h File Reference

Quantum gates.

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

  *Quantum++ main namespace.*

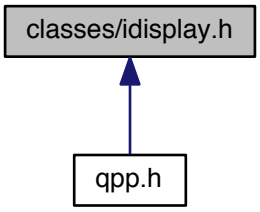
#### 8.3.1 Detailed Description

Quantum gates.

## 8.4 classes/idisplay.h File Reference

Display interface via the non-virtual interface (NVI)

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




### Classes

- class qpp::IDisplay

  *Abstract class (interface) that mandates the definition of virtual std::ostream& display(std::ostream& os) const.*

### Namespaces

- qpp

  *Quantum++ main namespace.*

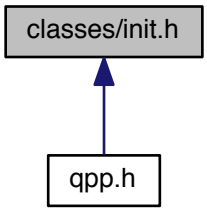
#### 8.4.1 Detailed Description

Display interface via the non-virtual interface (NVI)

## 8.5 classes/init.h File Reference

Initialization.

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
  *Quantum++ main namespace.*

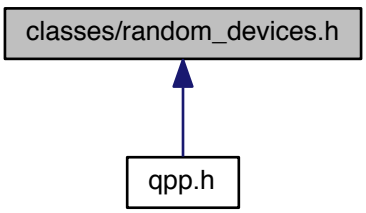
### 8.5.1 Detailed Description

Initialization.

## 8.6 classes/random_devices.h File Reference

Random devices.

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



### Classes

- class qpp::RandomDevices

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

### Namespaces

- qpp

  *Quantum++ main namespace.*

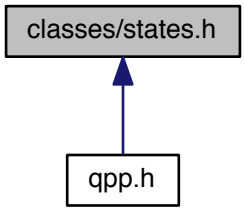
#### 8.6.1 Detailed Description

Random devices.

## 8.7 classes/states.h File Reference

Quantum states.

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

  *Quantum++ main namespace.*

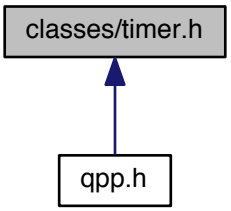
#### 8.7.1 Detailed Description

Quantum states.

## 8.8 classes/timer.h File Reference

Timing.

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




### Classes

- class qpp::Timer< T, CLOCK_T >
  *Chronometer.*

### Namespaces

- qpp
  *Quantum++ main namespace.*

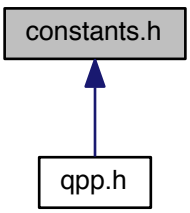
### 8.8.1 Detailed Description

Timing.

## 8.9 constants.h File Reference

Constants.

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



### Namespaces

- qpp

  *Quantum++ main namespace.*

### Functions

- constexpr cplx qpp::operator""_i (unsigned long long int x) noexcept

  *User-defined literal for complex $i = \sqrt{-1}$ (integer overload)*
- constexpr cplx qpp::operator""_i (long double x) noexcept

  *User-defined literal for complex $i = \sqrt{-1}$ (real overload)*
- cplx qpp::omega (idx D)

  *D-th root of unity.*

### Variables

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

  *Used in qpp::disp() for setting to zero numbers that have their absolute value smaller than qpp::chop.*
- constexpr double qpp::eps = 1e-12

  *Used to decide whether a number or expression in double precision is zero or not.*
- constexpr idx qpp::maxn = 64

  *Maximum number of allowed qu(d)its (subsystems)*
- constexpr double qpp::pi = 3.14159265358979323846264338327950288 4

  $\pi$
- constexpr double qpp::ee = 2.71828182845904523536028747135266249 7

  *Base of natural logarithm, $e$.*
- constexpr double qpp::infty = std::numeric_limits<double>::infinity()

  *Used to denote infinity in double precision.*

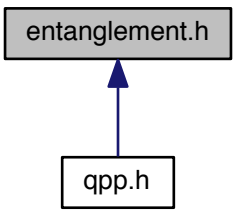
#### 8.9.1 Detailed Description

Constants.

## 8.10 entanglement.h File Reference

Entanglement functions.

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



### Namespaces

- qpp

  *Quantum++ main namespace.*

### Functions

- template<typename Derived >
  dyn_col_vect< double > qpp::schmidtcoeffs (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &dims)

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

- template<typename Derived >
  cmat qpp::schmidtA (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &dims)

  *Schmidt basis on Alice side.*

- template<typename Derived >
  cmat qpp::schmidtB (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &dims)

  *Schmidt basis on Bob side.*

- template<typename Derived >
  std::vector< double > qpp::schmidtprobs (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &dims)

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

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

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

- template<typename Derived >
  double qpp::lognegativity (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &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.*

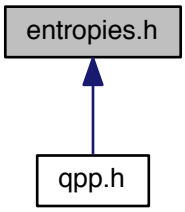
#### 8.10.1 Detailed Description

Entanglement functions.

## 8.11 entropies.h File Reference

Entropy functions.

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




## Namespaces

- qpp

  *Quantum++ main namespace.*

## Functions

- template<typename Derived >
  double qpp::entropy (const Eigen::MatrixBase< Derived > &A)

  *von-Neumann entropy of the density matrix A*
- double qpp::entropy (const std::vector< double > &prob)

  *Shannon entropy of the probability distribution prob.*
- template<typename Derived >
  double qpp::renyi (const Eigen::MatrixBase< Derived > &A, double alpha)

  *Renyi- $\alpha$ entropy of the density matrix A, for $\alpha \geq 0$.*
- double qpp::renyi (const std::vector< double > &prob, double alpha)

  *Renyi- $\alpha$ entropy of the probability distribution prob, for $\alpha \geq 0$.*
- template<typename Derived >
  double qpp::tsallis (const Eigen::MatrixBase< Derived > &A, double q)

  *Tsallis- $q$ entropy of the density matrix A, for $q \geq 0$.*
- double qpp::tsallis (const std::vector< double > &prob, double q)

  *Tsallis- $q$ entropy of the probability distribution prob, for $q \geq 0$.*
- template<typename Derived >
  double qpp::qmutualinfo (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &subsysA, const std::vector< idx > &subsysB, const std::vector< idx > &dims)

  *Quantum mutual information between 2 subsystems of a composite system.*
- template<typename Derived >
  double qpp::qmutualinfo (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &subsysA, const std::vector< idx > &subsysB, idx d=2)

  *Quantum mutual information between 2 subsystems of a composite system.*

### 8.11.1 Detailed Description

Entropy functions.

## 8.12 experimental/experimental.h File Reference

Experimental/test functions/classes.

### Namespaces

- qpp
  *Quantum++ main namespace.*
- qpp::experimental
  *Experimental/test functions/classes, do not use or modify.*

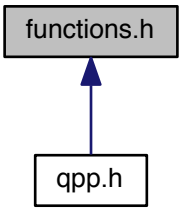
### 8.12.1 Detailed Description

Experimental/test functions/classes.

## 8.13 functions.h File Reference

Generic quantum computing functions.

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




### Namespaces

- qpp
  *Quantum++ main namespace.*

### Functions

- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::transpose (const Eigen::MatrixBase< Derived > &A)
  *Transpose.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::conjugate (const Eigen::MatrixBase< Derived > &A)
  *Complex conjugate.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::adjoint (const Eigen::MatrixBase< Derived > &A)
  *Adjoint.*

- template<typename Derived >
  dyn_mat< 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 >
  std::pair< dyn_col_vect< cplx >, cmat > qpp::eig (const Eigen::MatrixBase< Derived > &A)

  *Full eigen decomposition.*
- template<typename Derived >
  dyn_col_vect< cplx > qpp::evals (const Eigen::MatrixBase< Derived > &A)

  *Eigenvalues.*
- template<typename Derived >
  cmat qpp::evects (const Eigen::MatrixBase< Derived > &A)

  *Eigenvectors.*
- template<typename Derived >
  std::pair< dyn_col_vect< double >, cmat > qpp::heig (const Eigen::MatrixBase< Derived > &A)

  *Full eigen decomposition of Hermitian expression.*
- template<typename Derived >
  dyn_col_vect< double > qpp::hevals (const Eigen::MatrixBase< Derived > &A)

  *Hermitian eigenvalues.*
- template<typename Derived >
  cmat qpp::hevects (const Eigen::MatrixBase< Derived > &A)

  *Hermitian eigenvectors.*
- template<typename Derived >
  std::tuple< cmat, dyn_col_vect< double >, cmat > qpp::svd (const Eigen::MatrixBase< Derived > &A)

  *Full singular value decomposition.*
- template<typename Derived >
  dyn_col_vect< 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 >
  dyn_mat< typename Derived::Scalar > qpp::powm (const Eigen::MatrixBase< Derived > &A, idx n)

  *Matrix power.*
- template<typename Derived >
  double qpp::schatten (const Eigen::MatrixBase< Derived > &A, double p)

  *Schatten matrix norm.*
- template<typename OutputScalar , typename Derived >
  dyn_mat< OutputScalar > qpp::cwise (const Eigen::MatrixBase< Derived > &A, OutputScalar(*f)( const typename Derived::Scalar &))

  *Functor.*
- template<typename T >
  dyn_mat< typename T::Scalar > qpp::kron (const T &head)

  *Kronecker product.*
- template<typename T , typename... Args>
  dyn_mat< typename T::Scalar > qpp::kron (const T &head, const Args &...tail)

  *Kronecker product.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::kron (const std::vector< Derived > &As)

  *Kronecker product.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::kron (const std::initializer_list< Derived > &As)

  *Kronecker product.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::kronpow (const Eigen::MatrixBase< Derived > &A, idx n)

  *Kronecker power.*
- template<typename T >
  dyn_mat< typename T::Scalar > qpp::dirsum (const T &head)

  *Direct sum.*
- template<typename T , typename... Args>
  dyn_mat< typename T::Scalar > qpp::dirsum (const T &head, const Args &...tail)

  *Direct sum.*

- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::dirsum (const std::vector< Derived > &As)

  *Direct sum.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::dirsum (const std::initializer_list< Derived > &As)

  *Direct sum.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::dirsumpow (const Eigen::MatrixBase< Derived > &A, idx n)

  *Direct sum power.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::reshape (const Eigen::MatrixBase< Derived > &A, idx rows, idx cols)

  *Reshape.*
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > qpp::comm (const Eigen::MatrixBase< Derived1 > &A, const Eigen::MatrixBase< Derived2 > &B)

  *Commutator.*
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > qpp::anticomm (const Eigen::MatrixBase< Derived1 > &A, const Eigen::MatrixBase< Derived2 > &B)

  *Anti-commutator.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::prj (const Eigen::MatrixBase< Derived > &V)

  *Projector.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::grams (const std::vector< Derived > &Vs)

  *Gram-Schmidt orthogonalization.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::grams (const std::initializer_list< Derived > &Vs)

  *Gram-Schmidt orthogonalization.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::grams (const Eigen::MatrixBase< Derived > &A)

  *Gram-Schmidt orthogonalization.*
- std::vector< idx > qpp::n2multiidx (idx n, const std::vector< idx > &dims)

  *Non-negative integer index to multi-index.*
- idx qpp::multiidx2n (const std::vector< idx > &midx, const std::vector< idx > &dims)

  *Multi-index to non-negative integer index.*
- ket qpp::mket (const std::vector< idx > &mask, const std::vector< idx > &dims)

  *Multi-partite qudit ket.*
- ket qpp::mket (const std::vector< idx > &mask, idx d=2)

  *Multi-partite qudit ket.*
- cmat qpp::mprj (const std::vector< idx > &mask, const std::vector< idx > &dims)

  *Projector onto multi-partite qudit ket.*
- cmat qpp::mprj (const std::vector< idx > &mask, idx d=2)

  *Projector onto multi-partite qudit ket.*
- template<typename InputIterator >
  std::vector< double > qpp::abssq (InputIterator first, InputIterator last)

  *Computes the absolute values squared of an STL-like range of complex numbers.*
- template<typename Container >
  std::vector< double > qpp::abssq (const Container &c, typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Computes the absolute values squared of an STL-like container.*

- template<typename Derived >
  std::vector< double > qpp::abssq (const Eigen::MatrixBase< Derived > &A)

  *Computes the absolute values squared of an Eigen expression.*
- template<typename InputIterator >
  std::iterator_traits< InputIterator >::value_type qpp::sum (InputIterator first, InputIterator last)

  *Element-wise sum of an STL-like range.*
- template<typename Container >
  Container::value_type qpp::sum (const Container &c, typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Element-wise sum of the elements of an STL-like container.*
- template<typename InputIterator >
  std::iterator_traits< InputIterator >::value_type qpp::prod (InputIterator first, InputIterator last)

  *Element-wise product of an STL-like range.*
- template<typename Container >
  Container::value_type qpp::prod (const Container &c, typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Element-wise product of the elements of an STL-like container.*
- template<typename Derived >
  dyn_col_vect< typename Derived::Scalar > qpp::rho2pure (const Eigen::MatrixBase< Derived > &A)

  *Finds the pure state representation of a matrix proportional to a projector onto a pure state.*
- template<typename T >
  std::vector< T > qpp::complement (std::vector< T > subsys, idx N)

  *Constructs the complement of a subsystem vector.*
- template<typename Derived >
  std::vector< double > qpp::rho2bloch (const Eigen::MatrixBase< Derived > &A)

  *Computes the 3-dimensional real Bloch vector corresponding to the qubit density matrix A.*
- cmat qpp::bloch2rho (const std::vector< double > &r)

  *Computes the density matrix corresponding to the 3-dimensional real Bloch vector r.*

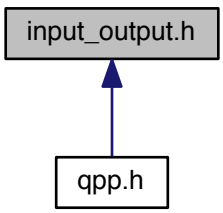
### 8.13.1 Detailed Description

Generic quantum computing functions.

## 8.14 input_output.h File Reference

Input/output functions.

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



### Namespaces

- qpp

  *Quantum++ main namespace.*

### Functions

- template<typename Derived >
  internal::IOManipEigen qpp::disp (const Eigen::MatrixBase< Derived > &A, double chop=qpp::chop)

  *Eigen expression ostream manipulator.*

- internal::IOManipEigen qpp::disp (cplx z, double chop=qpp::chop)

  *Complex number ostream manipulator.*

- template<typename InputIterator >
  internal::IOManipRange< InputIterator > qpp::disp (InputIterator first, InputIterator last, const std::string &separator, const std::string &start="[", const std::string &end="]")

  *Range ostream manipulator.*

- template<typename Container >
  internal::IOManipRange< typename Container::const_iterator > qpp::disp (const Container &c, const std::string &separator, const std::string &start="[", const std::string &end="]", typename std::enable_if< is_iterable< Container >::value >::type *=nullptr)

  *Standard container ostream manipulator. The container must support std::begin(), std::end() and forward iteration.*

- template<typename PointerType >
  internal::IOManipPointer< PointerType > qpp::disp (const PointerType *p, idx n, const std::string &separator, const std::string &start="[", const std::string &end="]")

  *C-style pointer ostream manipulator.*

- 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 >
  dyn_mat< typename Derived::Scalar > qpp::load (const std::string &fname)

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

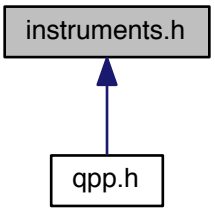
#### 8.14.1 Detailed Description

Input/output functions.

## 8.15 instruments.h File Reference

Measurement functions.

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




## Namespaces

- qpp

  *Quantum++ main namespace.*

## Functions

- template<typename Derived >
  dyn_col_vect< typename Derived::Scalar > qpp::ip (const Eigen::MatrixBase< Derived > &phi, const Eigen::MatrixBase< Derived > &psi, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Generalized inner product.*
- template<typename Derived >
  dyn_col_vect< typename Derived::Scalar > qpp::ip (const Eigen::MatrixBase< Derived > &phi, const Eigen::MatrixBase< Derived > &psi, const std::vector< idx > &subsys, idx d=2)

  *Generalized inner product.*
- template<typename Derived >
  std::tuple< idx, 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::tuple< idx, 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.*
- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > qpp::measure (const Eigen::MatrixBase< Derived > &A, const cmat &U)

  *Measures the state A in the orthonormal basis specified by the unitary matrix U.*
- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > qpp::measure (const Eigen::MatrixBase< Derived > &A, const std::vector< cmat > &Ks, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Measures the part subsys of the multi-partite state vector or density matrix A using the set of Kraus operators Ks.*
- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > qpp::measure (const Eigen::MatrixBase< Derived > &A, const std::initializer_list< cmat > &Ks, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Measures the part subsys of the multi-partite state vector or density matrix A using the set of Kraus operators Ks.*

- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > qpp::measure (const Eigen::MatrixBase< Derived > &A, const std::vector< cmat > &Ks, const std::vector< idx > &subsys, idx d=2)

  *Measures the part subsys of the multi-partite state vector or density matrix A using the set of Kraus operators Ks.*

- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > qpp::measure (const Eigen::MatrixBase< Derived > &A, const std::initializer_list< cmat > &Ks, const std::vector< idx > &subsys, idx d=2)

  *Measures the part subsys of the multi-partite state vector or density matrix A using the set of Kraus operators Ks.*

- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > qpp::measure (const Eigen::MatrixBase< Derived > &A, const cmat &V, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Measures the part subsys of the multi-partite state vector or density matrix A in the orthonormal basis or rank-1 POVM specified by the matrix V.*

- template<typename Derived >
  std::tuple< idx, std::vector< double >, std::vector< cmat > > qpp::measure (const Eigen::MatrixBase< Derived > &A, const cmat &V, const std::vector< idx > &subsys, idx d=2)

  *Measures the part subsys of the multi-partite state vector or density matrix A in the orthonormal basis or rank-1 POVM specified by the matrix V.*

- template<typename Derived >
  std::tuple< std::vector< idx >, double, cmat > qpp::measure_seq (const Eigen::MatrixBase< Derived > &A, std::vector< idx > subsys, std::vector< idx > dims)

  *Sequentially measures the part subsys of the multi-partite state vector or density matrix A in the computational basis.*

- template<typename Derived >
  std::tuple< std::vector< idx >, double, cmat > qpp::measure_seq (const Eigen::MatrixBase< Derived > &A, std::vector< idx > subsys, idx d=2)

  *Sequentially measures the part subsys of the multi-partite state vector or density matrix A in the computational basis.*

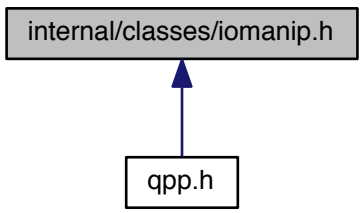
### 8.15.1 Detailed Description

Measurement functions.

## 8.16 internal/classes/iomanip.h File Reference

Input/output manipulators.

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



### Classes

- struct qpp::internal::_details::_Display_Impl
- class qpp::internal::IOManipRange< InputIterator >
- class qpp::internal::IOManipPointer< PointerType >
- class qpp::internal::IOManipEigen

### Namespaces

- qpp

  *Quantum++ main namespace.*
- qpp::internal

  *Internal utility functions, do not use/modify.*
- qpp::internal::_details

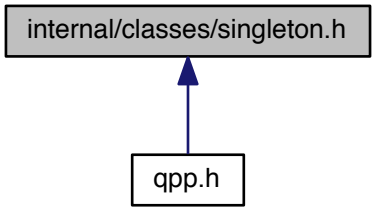
#### 8.16.1 Detailed Description

Input/output manipulators.

## 8.17 internal/classes/singleton.h File Reference

Singleton pattern via CRTP.

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

  *Quantum++ main namespace.*
- qpp::internal

  *Internal utility functions, do not use/modify.*

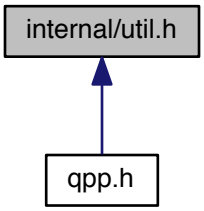
### 8.17.1 Detailed Description

Singleton pattern via CRTP.

## 8.18 internal/util.h File Reference

Internal utility functions.

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




### Namespaces

- qpp
  *Quantum++ main namespace.*
- qpp::internal
  *Internal utility functions, do not use/modify.*

### Functions

- void qpp::internal::_n2multiidx (idx n, idx numdims, const idx ∗dims, idx ∗result) noexcept
- idx qpp::internal::_multiidx2n (const idx ∗midx, idx numdims, const idx ∗dims) noexcept
- 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_rvector (const Eigen::MatrixBase< Derived > &A)
- template<typename Derived >
  bool qpp::internal::_check_cvector (const Eigen::MatrixBase< Derived > &A)
- template<typename T >
  bool qpp::internal::_check_nonzero_size (const T &x) noexcept
- template<typename T1 , typename T2 >
  bool qpp::internal::_check_matching_sizes (const T1 &lhs, const T2 &rhs) noexcept
- bool qpp::internal::_check_dims (const std::vector< idx > &dims)
- template<typename Derived >
  bool qpp::internal::_check_dims_match_mat (const std::vector< idx > &dims, const Eigen::MatrixBase< Derived > &A)
- template<typename Derived >
  bool qpp::internal::_check_dims_match_cvect (const std::vector< idx > &dims, const Eigen::MatrixBase< Derived > &V)

- template<typename Derived >
  bool qpp::internal::_check_dims_match_rvect (const std::vector< idx > &dims, const Eigen::MatrixBase< Derived > &V)
- bool qpp::internal::_check_eq_dims (const std::vector< idx > &dims, idx dim) noexcept
- bool qpp::internal::_check_subsys_match_dims (const std::vector< idx > &subsys, const std::vector< idx > &dims)
- template<typename Derived >
  bool qpp::internal::_check_qubit_matrix (const Eigen::MatrixBase< Derived > &A) noexcept
- template<typename Derived >
  bool qpp::internal::_check_qubit_cvector (const Eigen::MatrixBase< Derived > &V) noexcept
- template<typename Derived >
  bool qpp::internal::_check_qubit_rvector (const Eigen::MatrixBase< Derived > &V) noexcept
- template<typename Derived >
  bool qpp::internal::_check_qubit_vector (const Eigen::MatrixBase< Derived > &V) noexcept
- bool qpp::internal::_check_perm (const std::vector< idx > &perm)
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > qpp::internal::_kron2 (const Eigen::MatrixBase< Derived1 > &A, const Eigen::MatrixBase< Derived2 > &B)
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > qpp::internal::_dirsum2 (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)

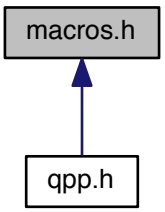
### 8.18.1 Detailed Description

Internal utility functions.

## 8.19 macros.h File Reference

Preprocessor macros.

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




### Macros

- #define PRINT(x)
- #define PRINTLN(x)

- #define ERROR(x)
- #define ERRORLN(x)

### 8.19.1 Detailed Description

Preprocessor macros.

### 8.19.2 Macro Definition Documentation

#### 8.19.2.1 #define ERROR( *x* )

Prints an error message to std::cerr

#### 8.19.2.2 #define ERRORLN( *x* )

Prints an error message to std::cerr and adds a new line

#### 8.19.2.3 #define PRINT( *x* )

Prints a message

#### 8.19.2.4 #define PRINTLN( *x* )

Prints a message and adds a new line

## 8.20 MATLAB/matlab.h File Reference

Input/output interfacing with MATLAB.

```
#include "mat.h"
#include "mex.h"
```

### Namespaces

- qpp

  *Quantum++ main namespace.*

### Functions

- template<typename Derived >
  Derived qpp::loadMATLABmatrix (const std::string &, const std::string &)

  *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 > &, const std::string &, const std::string &, const std::string &)

  *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)*

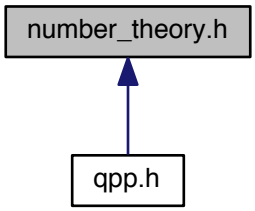
### 8.20.1 Detailed Description

Input/output interfacing with MATLAB.

## 8.21 number_theory.h File Reference

Number theory functions.

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




### Namespaces

- qpp

  *Quantum++ main namespace.*

### Functions

- std::vector< int > qpp::x2contfrac (double x, idx n, idx cut=1e5)

  *Simple continued fraction expansion.*
- double qpp::contfrac2x (const std::vector< int > &cf, idx n)

  *Real representation of a simple continued fraction.*
- double qpp::contfrac2x (const std::vector< int > &cf)

  *Real representation of a simple continued fraction.*
- bigint qpp::gcd (bigint m, bigint n)

*Greatest common divisor of two integers.*

- bigint qpp::gcd (const std::vector< bigint > &ns)

  *Greatest common divisor of a list of integers.*

- bigint qpp::lcm (bigint m, bigint n)

  *Least common multiple of two integers.*

- bigint qpp::lcm (const std::vector< bigint > &ns)

  *Least common multiple of a list of integers.*

- std::vector< idx > qpp::invperm (const std::vector< idx > &perm)

  *Inverse permutation.*

- std::vector< idx > qpp::compperm (const std::vector< idx > &perm, const std::vector< idx > &sigma)

  *Compose permutations.*

- std::vector< bigint > qpp::factors (bigint n)

  *Prime factor decomposition.*

- bool qpp::isprime (bigint n)

  *Primality test.*

- bigint qpp::modpow (bigint a, bigint n, bigint p)

  *Fast integer power modulo p based on the SQUARE-AND-MULTIPLY algorithm.*

- std::tuple< bigint, bigint, bigint > qpp::egcd (bigint m, bigint n)

  *Extended greatest common divisor of two integers.*

- bigint qpp::modinv (bigint a, bigint p)

  *Modular inverse of a mod p.*

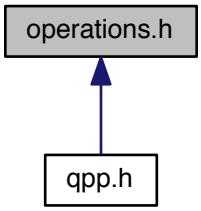
### 8.21.1 Detailed Description

Number theory functions.

## 8.22 operations.h File Reference

Quantum operation functions.

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




### Namespaces

- qpp

  *Quantum++ main namespace.*

## Functions

- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > qpp::applyCTRL (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< idx > &ctrl, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Applies the controlled-gate A to the part subsys of the multi-partite state vector or density matrix state.*
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > qpp::applyCTRL (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< idx > &ctrl, const std::vector< idx > &subsys, idx d=2)

  *Applies the controlled-gate A to the part subsys of the multi-partite state vector or density matrix state.*
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > qpp::apply (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Applies the gate A to the part subsys of the multi-partite state vector or density matrix state.*
- template<typename Derived1 , typename Derived2 >
  dyn_mat< typename Derived1::Scalar > qpp::apply (const Eigen::MatrixBase< Derived1 > &state, const Eigen::MatrixBase< Derived2 > &A, const std::vector< idx > &subsys, idx d=2)

  *Applies the gate A to the part subsys of the multi-partite state vector or density matrix state.*
- template<typename Derived >
  cmat qpp::apply (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::apply (const Eigen::MatrixBase< Derived > &rho, const std::vector< cmat > &Ks, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Applies the channel specified by the set of Kraus operators Ks to the part subsys of the multi-partite density matrix rho.*
- template<typename Derived >
  cmat qpp::apply (const Eigen::MatrixBase< Derived > &rho, const std::vector< cmat > &Ks, const std::vector< idx > &subsys, idx d=2)

  *Applies the channel specified by the set of Kraus operators Ks to the part subsys of the multi-partite density matrix rho.*
- cmat qpp::kraus2super (const std::vector< cmat > &Ks)

  *Superoperator matrix.*
- cmat qpp::kraus2choi (const std::vector< cmat > &Ks)

  *Choi matrix.*
- std::vector< cmat > qpp::choi2kraus (const cmat &A)

  *Orthogonal Kraus operators from Choi matrix.*
- cmat qpp::choi2super (const cmat &A)

  *Converts Choi matrix to superoperator matrix.*
- cmat qpp::super2choi (const cmat &A)

  *Converts superoperator matrix to Choi matrix.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::ptrace1 (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &dims)

  *Partial trace.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::ptrace2 (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &dims)

  *Partial trace.*
- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::ptrace (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &subsys, const std::vector< idx > &dims)

*Partial trace.*

- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::ptrace (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &subsys, idx d=2)

  *Partial trace.*

- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::ptranspose (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &subsys, const std::vector< idx > &dims)

  *Partial transpose.*

- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::ptranspose (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &subsys, idx d=2)

  *Partial transpose.*

- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::syspermute (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &perm, const std::vector< idx > &dims)

  *Subsystem permutation.*

- template<typename Derived >
  dyn_mat< typename Derived::Scalar > qpp::syspermute (const Eigen::MatrixBase< Derived > &A, const std::vector< idx > &perm, idx d=2)

  *Subsystem permutation.*

### 8.22.1 Detailed Description

Quantum operation functions.

## 8.23 qpp.h File Reference

Quantum++ main header file, includes all other necessary headers.

```
#include <algorithm>
#include <cassert>
#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 "macros.h"
#include "types.h"
#include "classes/exception.h"
#include "constants.h"
#include "traits.h"
#include "classes/idisplay.h"
#include "internal/util.h"
#include "internal/classes/iomanip.h"
#include "input_output.h"
#include "internal/classes/singleton.h"
#include "classes/init.h"
#include "functions.h"
#include "classes/codes.h"
#include "classes/gates.h"
#include "classes/states.h"
#include "classes/random_devices.h"
#include "statistics.h"
#include "operations.h"
#include "entropies.h"
#include "entanglement.h"
#include "random.h"
#include "classes/timer.h"
#include "instruments.h"
#include "number_theory.h"
```

## Namespaces

- qpp

*Quantum++ main namespace.*

### Macros

- #define _QPP_UNUSED_

### 8.23.1 Detailed Description

Quantum++ main header file, includes all other necessary headers.

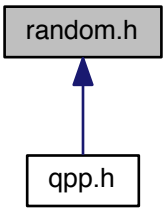
### 8.23.2 Macro Definition Documentation

#### 8.23.2.1 #define _QPP_UNUSED_

## 8.24 random.h File Reference

Randomness-related functions.

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




### Namespaces

- qpp

  *Quantum++ main namespace.*

### Functions

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

  *Generates a random real number uniformly distributed in the interval [a, b)*
- bigint qpp::rand (bigint a=std::numeric_limits< bigint >::min(), bigint b=std::numeric_limits< bigint >::max())

  *Generates a random big integer uniformly distributed in the interval [a, b].*
- ubigint qpp::rand (ubigint a=std::numeric_limits< ubigint >::min(), ubigint b=std::numeric_limits< ubigint >↩ ::max())

  *Generates a random non-negative big integer uniformly distributed in the interval [a, b].*
- idx qpp::randidx (idx a=std::numeric_limits< idx >::min(), idx b=std::numeric_limits< idx >::max())

  *Generates a random index (idx) uniformly distributed in the interval [a, b].*

- template<typename Derived >
  Derived qpp::rand (idx rows, idx cols, double a=0, double b=1)

  *Generates a random matrix with entries uniformly distributed in the interval [a, b)*
- template<>
  dmat qpp::rand (idx rows, idx 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 (idx rows, idx 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)*
- template<typename Derived >
  Derived qpp::randn (idx rows, idx cols, double mean=0, double sigma=1)

  *Generates a random matrix with entries normally distributed in N(mean, sigma)*
- template<>
  dmat qpp::randn (idx rows, idx cols, double mean, double sigma)

  *Generates a random real matrix with entries normally distributed in N(mean, sigma), specialization for double matrices (qpp::dmat)*
- template<>
  cmat qpp::randn (idx rows, idx cols, double mean, double sigma)

  *Generates a random complex matrix with entries (both real and imaginary) normally distributed in N(mean, 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(mean, sigma)*
- cmat qpp::randU (idx D)

  *Generates a random unitary matrix.*
- cmat qpp::randV (idx Din, idx Dout)

  *Generates a random isometry matrix.*
- std::vector< cmat > qpp::randkraus (idx N, idx D)

  *Generates a set of random Kraus operators.*
- cmat qpp::randH (idx D)

  *Generates a random Hermitian matrix.*
- ket qpp::randket (idx D)

  *Generates a random normalized ket (pure state vector)*
- cmat qpp::randrho (idx D)

  *Generates a random density matrix.*
- std::vector< idx > qpp::randperm (idx n)

  *Generates a random uniformly distributed permutation.*

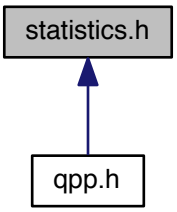
### 8.24.1 Detailed Description

Randomness-related functions.

## 8.25 statistics.h File Reference

Statistics functions.

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




## Namespaces

- qpp

  *Quantum++ main namespace.*

## Functions

- std::vector< double > qpp::uniform (idx N)

  *Uniform probability distribution vector.*
- std::vector< double > qpp::marginalX (const dmat &probXY)

  *Marginal distribution.*
- std::vector< double > qpp::marginalY (const dmat &probXY)

  *Marginal distribution.*
- template<typename Container >
  double qpp::avg (const std::vector< double > &prob, const Container &X, typename std::enable_if< is_↩
  iterable< Container >::value >::type *=nullptr)

  *Average.*
- template<typename Container >
  double qpp::cov (const dmat &probXY, const Container &X, const Container &Y, typename std::enable_if<
  is_iterable< Container >::value >::type *=nullptr)

  *Covariance.*
- template<typename Container >
  double qpp::var (const std::vector< double > &prob, const Container &X, typename std::enable_if< is_↩
  iterable< Container >::value >::type *=nullptr)

  *Variance.*
- template<typename Container >
  double qpp::sigma (const std::vector< double > &prob, const Container &X, typename std::enable_if< is_↩
  iterable< Container >::value >::type *=nullptr)

  *Standard deviation.*
- template<typename Container >
  double qpp::cor (const dmat &probXY, const Container &X, const Container &Y, typename std::enable_if<
  is_iterable< Container >::value >::type *=nullptr)

  *Correlation.*

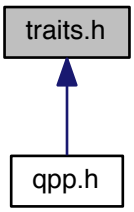
### 8.25.1 Detailed Description

Statistics functions.

## 8.26 traits.h File Reference

Type traits.

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




### Classes

- struct qpp::is_iterable< T, typename >

  *Checks whether T is compatible with an STL-like iterable container.*
- struct qpp::is_iterable< T, to_void< decltype(std::declval< T >().begin()), decltype(std::declval< T >().↩ end()), typename T::value_type > >

  *Checks whether T is compatible with an STL-like iterable container, specialization for STL-like iterable containers.*
- struct qpp::is_matrix_expression< Derived >

  *Checks whether the type is an Eigen matrix expression.*
- struct qpp::is_matrix_expression< typename Eigen::MatrixBase< Derived > >

  *Checks whether the type is an Eigen matrix expression, specialization for Eigen matrix expressions.*
- struct qpp::is_complex< T >

  *Checks whether the type is a complex type.*
- struct qpp::is_complex< std::complex< T > >

  *Checks whether the type is a complex number type, specialization for complex types.*

### Namespaces

- qpp

  *Quantum++ main namespace.*

### Typedefs

- template<typename... >
  using qpp::to_void = void

  *Alias template that implements the proposal for void_t.*

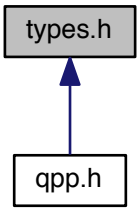
### 8.26.1 Detailed Description

Type traits.

## 8.27 types.h File Reference

Type aliases.

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




### Namespaces

- qpp

  *Quantum++ main namespace.*

### Typedefs

- using qpp::idx = std::size_t

  *Non-negative integer index.*
- using qpp::bigint = long long int

  *Big integer.*
- using qpp::ubigint = unsigned long long int

  *Non-negative big integer.*
- using qpp::cplx = std::complex< double >

  *Complex number in double precision.*
- using qpp::ket = Eigen::VectorXcd

  *Complex (double precision) dynamic Eigen column vector.*
- using qpp::bra = Eigen::RowVectorXcd

  *Complex (double precision) dynamic Eigen row vector.*
- using qpp::cmat = Eigen::MatrixXcd

  *Complex (double precision) dynamic Eigen matrix.*
- using qpp::dmat = Eigen::MatrixXd

  *Real (double precision) dynamic Eigen matrix.*
- template<typename Scalar >
  using qpp::dyn_mat = Eigen::Matrix< Scalar, Eigen::Dynamic, Eigen::Dynamic >

  *Dynamic Eigen matrix over the field specified by Scalar.*
- template<typename Scalar >
  using qpp::dyn_col_vect = Eigen::Matrix< Scalar, Eigen::Dynamic, 1 >

  *Dynamic Eigen column vector over the field specified by Scalar.*
- template<typename Scalar >
  using qpp::dyn_row_vect = Eigen::Matrix< Scalar, 1, Eigen::Dynamic >

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

### 8.27.1 Detailed Description

Type aliases.

# Index