MindQuantum Documentation

Release 0.7.0

a

Jul 11, 2022

INSTALLATION

MindQuantum is a general quantum computing framework developed by MindSpore and HiQ, that can be used to build and train different quantum neural networks. Thanks to the powerful algorithm of quantum software group of Huawei and High-performance automatic differentiation ability of MindSpore, MindQuantum can efficiently handle problems such as quantum machine learning, quantum chemistry simulation, and quantum optimization, which provides an efficient platform for researchers, teachers and students to quickly design and verify quantum machine learning algorithms.
MindQuantum Architecture

CHAPTER

ONE

QUICK START

If you are only looking in using MindQuantum on your system, the easiest way of getting started is installing it directly from Pypi and use one of the pre-compiled binaries: *Install from Pypi*.

If you are looking into doing some development with MindQuantum on your local machine, we highly recommend you using one of the scripts provided to build MindQuantum locally: *Build locally (for developers)*. In that case, you might want to install some of the required programs and libraries. See one of the sub-sections under *Requirements* for your particular system for more information in order to find out how to achieve that. Additionally, if you plan to link some libraries you are developping to MindQuantum, have a look at the *Install locally (as a library)*. This will guide you into adding MindQuantum as a third-party library into your other projects.

If you plan on distributing the version of MindQuantum you have compiled on your system, we would suggest that you have a look at *Build binary Python wheels (for distribution)*.

1.1 Install from Pypi

You can install one of the pre-compiled binary Python packages directly from Pypi using Pip:

```
python3 -m pip install --user mindquantum
```

1.2 Build locally (for developers)

In order to build MindQuantum locally for developping new features or implementing bug fixes for MindQuantum, there are a few scripts that you can use to properly setup a virtual environment and all the required build tools (such as CMake). Currently, there are three local build scripts:

- `build_locally.bat` (MS-DOS BATCH script)
- `build_locally.ps1` (PowerShell script)
- `build_locally.sh` (Bash script)

Except a few minor differences[1], the functionalities of all three scripts are identical. All the scripts accept a flag to display the help message (`-h, --help`, `-H, -Help`, ``/h, /Help` for Bash, Powershell and MS-DOS BATCH). Please invoke the script of your choice in order to view the latest set of functionalities provided by it.

The build scripts mentioned above will perform the following operations in order:

1. Setup a Python virtual environment
2. Update the virtual environment's packages and install some required dependencies (which may include CMake)

[1] PowerShell and Bash scripts typically have identical functionality sets whereas the MS-DOS BATCH script might not. For example, the latter does not support the `/WithOutXXX`-type arguments.

3. Add a PTH-file to the Python virtual environment to make sure that MindQuantum will be detected
4. Create a build directory and run CMake within it
5. Compile MindQuantum in-place

The next time you run the script, unless you specify one of the cleaning options or force a CMake configuration step, the script will only re-compile MindQuantum.

For reference, here is the output of the help message from the Bash script (NB: might differ from the actual help message):

```
Build MindQunantum locally (in-source build)

This is mainly relevant for developers that do not want to always have to reinstall the␣
↪Python
package

This script will create a Python virtualenv in the MindQuantum root directory and then␣
↪build
all the C++ Python modules and place the generated libraries in their right locations␣
↪within
the MindQuantum folder hierarchy so Python knows how to find them.

A pth-file will be created in the virtualenv site-packages directory so that the␣
↪MindQuantum
root folder will be added to the Python PATH without the need to modify PYTHONPATH.

Usage:
  build_locally.sh [options] [-- cmake_options]

Options:
  -h,--help             Show this help message and exit
  -n                    Dry run; only print commands but do not execute them

  -B,--build=[dir]      Specify build directory
                        Defaults to: /home/user/mindquantum/build
  --ccache              If ccache or sccache are found within the PATH, use them with␣
↪CMake
  --clean-3rdparty      Clean 3rd party installation directory
  --clean-all           Clean everything before building.
                        Equivalent to --clean-venv --clean-builddir
  --clean-builddir      Delete build directory before building
  --clean-cache         Re-run CMake with a clean CMake cache
  --clean-venv          Delete Python virtualenv before building
  --config=[dir]        Path to INI configuration file with default values for the␣
↪parameters
                        Defaults to: /home/user/mindquantum/build.conf
                        NB: command line arguments always take precedence over␣
↪configuration
                        file values
  --cxx                 (experimental) Enable MindQuantum C++ support
  --debug               Build in debug mode
  --debug-cmake         Enable debugging mode for CMake configuration step
  --gpu                 Enable GPU support
```

(continues on next page)

(continued from previous page)

```
  -j,--jobs [N]         Number of parallel jobs for building
                        Defaults to: 16
  --local-pkgs          Compile third-party dependencies locally
  --ninja               Build using Ninja instead of make
  --quiet               Disable verbose build rules
  --show-libraries      Show all known third-party libraries
  -v, --verbose         Enable verbose output from the Bash scripts
  --venv=[dir]          Path to Python virtual environment
                        Defaults to: /home/user/mindquantum/venv
  --with-<library>      Build the third-party <library> from source
                        (ignored if --local-pkgs is passed, except for projectq)
  --without-<library>   Do not build the third-party library from source
                        (ignored if --local-pkgs is passed, except for projectq)

Test related options:
  --test                Build C++ tests and install dependencies for Python testing as␣
↪well
  --only-pytest         Only install pytest and its dependencies when creating/building␣
↪the
                        virtualenv

CUDA related options:
  --cuda-arch=[arch]    Comma-separated list of architectures to generate device code for.
                        Only useful if --gpu is passed. See CMAKE_CUDA_ARCHITECTURES for␣
↪more
                        information.

Python related options:
  --update-venv         Update the python virtual environment

Developer options:
  --cmake-no-registry   Disable the use of CMake package registries during configuration

Extra options:
  --clean               Run make clean before building
  -c,--configure        Force running the CMake configure step
  --configure-only      Stop after the CMake configure and generation steps
                        (ie. before building MindQuantum)
  --doc,--docs          Setup the Python virtualenv for building the documentation and ask
                        CMake to build the documentation
  --install             Build the ´install´ target
  --prefix              Specify installation prefix

Any options after "--" will be passed onto CMake during the configuration step

Example calls:
build_locally.sh -B build
build_locally.sh -B build --gpu
build_locally.sh -B build --cxx --with-boost --without-gmp --venv=/tmp/venv
build_locally.sh -B build -- -DCMAKE_CUDA_COMPILER=/opt/cuda/bin/nvcc
build_locally.sh -B build --cxx --gpu -- \
       -DCMAKE_NVCXX_COMPILER=/opt/nvidia/hpc_sdk/Linux_x86_64/22.3/compilers/bin/nvc++
```

1.3 Install locally (as a library)

If you plan on integrating MindQuantum into your own project as a third-party library, you may want to install it locally on your computer. For that you may use the scripts mentioned in Section *Build locally (for developers)*.

There are essentially three ways you can include MindQuantum as a third-party library:

1. As a sub-directory if your project also uses CMake (`add_subdirectory("path/to/mindquantum")`)
2. Installing MindQuantum as a library somewhere on your system
3. Using the build directory as a pseudo-installation location (provided your project also uses CMake)

As option 1. is pretty straightforward, we will not provide more explanation here. However, for the other two options, some more detailed help can be found below.

1.3.1 Installation on your system

If you are using the local build scripts, simply add the `--install` (or `-Install` or `/Install`) and if necessary the `--prefix` (or `-Prefix` or `/Prefix`) arguments to your command line to install MindQuantum in your preferred location.

Given an installation <prefix>, building the `install` target will result in the relevant files being installed into:

<prefix>/include/mindquantum
: All MindQuantum header files

<prefix>/lib/mindquantum
: All MindQuantum libraries (excuding 3rd-party libraries)

<prefix>/lib/mindquantum/third_party
: All 3rd-party libraries (including their header files). This is actually the content of the `build/.mqlibs` folder within the build directory.

<prefix>/share/cmake/mindquantum/
: All CMake installation configuration files. This includes the `mindquantumConfig.cmake` file and other helper files.

Then, in order to use MindQuantum in some other CMake project, you simply need to add the following statement: `find_package(mindquantum CONFIG)`:

```
cmake_minimum_required(VERSION 3.20)
project(XXXX CXX)

# ...

find_package(mindquantum CONFIG)

# ...
```

You may need to also define either of `mindquantum_ROOT` or `mindquantum_DIR` CMake variables in order to help CMake locate the MindQuantum installation. For the former, simply defining it to <prefix> should suffice:

```
cmake ... -Dmindquantum_ROOT=/path/to/mindquantum/install
```

Instead of defining `mindquantum_ROOT` you may alternatively define `mindquantum_DIR`. In this case, the path must be to the directory that contains the `mindquantumConfig.cmake` file.

```
cmake ... -Dmindquantum_DIR=/path/to/mindquantum/install/share/mindquantum/cmake
```

Note: You may defined either of `mindquantum_DEBUG` or `MINDQUANTUM_DEBUG` to a truthful value to have CMake be more verbose when reading the MindQuantum configuration files.

If you have MindQuantum installed as a Python package, you can also use the module itself to locate where the CMake installation config file is located. You may use any of the following commands:

```
> python3 -m mindquantum --cmakedir
/usr/local/lib/python3.10/site-packages/mindquantum/share/mindquantum/cmake

> mindquantum-config --cmakedir
/usr/local/lib/python3.10/site-packages/mindquantum/share/mindquantum/cmake
```

Note: Both of the above commands provide the exact same information. The advantage of the latter over the former is that it does not attempt to load the mindquantum package which may be slower to execute in practice.

1.3.2 In-build "pseudo"-install

If you do not wish to install MindQuantum on your system, you may use the build directory as a *pseudo-installation* location. Simply follow the above instructions and simply set either of `mindquantum_ROOT` or `mindquantum_DIR` CMake variables to point to your build directory:

```
cmake ... -Dmindquantum_DIR=/path/to/mindquantum/build
```

1.4 Build binary Python wheels (for distribution)

If you plan on compiling MindQuantum on your local machine (or some CI) and would like to distribute the code in binary form to other users, we woul dsuggest you take a look at the `build.sh` script.

The build script mentioned above will perform the following operations in order:

1. Setup a Python virtual environment
2. Update the virtual environment's packages and install some required dependencies (which may include CMake)
3. Call `python3 -m build`

It has similar options as the scripts described in *Build locally (for developers)*:

```
Build binary Python wheel for MindQunantum

This is mainly relevant for developers that want to deploy MindQuantum on machines other
than their own.

This script will create a Python virtualenv in the MindQuantum root directory and then␣
↪build a
binary Python wheel of MindQuantum.
```

(continues on next page)

(continued from previous page)

```
Usage:
  build.sh [options] [-- cmake_options]

Options:
  -h,--help             Show this help message and exit
  -n                    Dry run; only print commands but do not execute them

  -B,--build=[dir]      Specify build directory
                        Defaults to: /home/user/mindquantum/build
  --ccache              If ccache or sccache are found within the PATH, use them with␣
↪CMake
  --clean-3rdparty      Clean 3rd party installation directory
  --clean-all           Clean everything before building.
                        Equivalent to --clean-venv --clean-builddir
  --clean-builddir      Delete build directory before building
  --clean-cache         Re-run CMake with a clean CMake cache
  --clean-venv          Delete Python virtualenv before building
  --config=[dir]        Path to INI configuration file with default values for the␣
↪parameters
                        Defaults to: /home/user/mindquantum/build.conf
                        NB: command line arguments always take precedence over␣
↪configuration
                        file values
  --cxx                 (experimental) Enable MindQuantum C++ support
  --debug               Build in debug mode
  --debug-cmake         Enable debugging mode for CMake configuration step
  --gpu                 Enable GPU support
  -j,--jobs [N]         Number of parallel jobs for building
                        Defaults to: 16
  --local-pkgs          Compile third-party dependencies locally
  --ninja               Build using Ninja instead of make
  --quiet               Disable verbose build rules
  --show-libraries      Show all known third-party libraries
  -v, --verbose         Enable verbose output from the Bash scripts
  --venv=[dir]          Path to Python virtual environment
                        Defaults to: /home/user/mindquantum/venv
  --with-<library>      Build the third-party <library> from source
                        (ignored if --local-pkgs is passed, except for projectq)
  --without-<library>   Do not build the third-party library from source
                        (ignored if --local-pkgs is passed, except for projectq)

Test related options:
  --test                Build C++ tests and install dependencies for Python testing as␣
↪well
  --only-pytest         Only install pytest and its dependencies when creating/building␣
↪the
                        virtualenv

CUDA related options:
  --cuda-arch=[arch]    Comma-separated list of architectures to generate device code for.
                        Only useful if --gpu is passed. See CMAKE_CUDA_ARCHITECTURES for␣
↪more
```

(continues on next page)

(continued from previous page)

```
                      information.

Python related options:
  --update-venv       Update the python virtual environment

Developer options:
  --cmake-no-registry  Disable the use of CMake package registries during configuration

Extra options:
  --delocate          Delocate the binary wheels after build is finished
                      (enabled by default; pass --no-delocate to disable)
  --no-delocate       Disable delocating the binary wheels after build is finished
  --no-build-isolation Pass --no-isolation to python3 -m build
  -o,--output=[dir]   Output directory for built wheels
  -p,--plat-name=[dir] Platform name to use for wheel delocation
                      (only effective if --delocate is used)

Example calls:
build.sh
build.sh --gpu
build.sh --cxx --with-boost --without-gmp --venv=/tmp/venv
```

CHAPTER

TWO

REQUIREMENTS

In order to get started with MindQuantum, you will need to have a C++ compiler installed on your system as well as a few libraries and programs:

- Python >= 3.5
- CMake >= 3.20

Below you will find detailed installation instructions for various operating systems.

2.1 Linux

2.1.1 Ubuntu/Debian

After having installed the build tools (for g++):

```
sudo apt-get install build-essential
```

You only need to install Python (and the package manager). For version 3.x, run

```
sudo apt-get install python3-dev python3-pip python3-venv
```

If the CMake version provided by Ubuntu is not recent enough (typically the case for Ubuntu <= 21.10), you may want to install CMake using Pip:

```
python3 -m pip install --user cmake
```

Otherwise, install CMake using APT as normal:

```
sudo apt-get install cmake
```

Note: On Ubuntu, you may use the https://apt.kitware.com/ repository. Follow the instruction there in order to install the latest CMake using APT.

2.1.2 ArchLinux/Manjaro

Make sure that you have a C/C++ compiler installed:

```
sudo pacman -Syu gcc
```

You only need to install Python (and the package manager). For version 3, run

```
sudo pacman -Syu python python-pip
```

Then install CMake using the following command:

```
sudo pacman -Syu cmake
```

2.1.3 CentOS 7

Run the following commands:

```
sudo yum install -y epel-release
sudo yum install -y centos-release-scl
sudo yum install -y devtoolset-8
sudo yum check-update -y

scl enable devtoolset-8 bash
sudo yum install -y gcc-c++ make git
sudo yum install -y python3 python3-devel python3-pip

sudo python3 -m pip install cmake
```

2.1.4 CentOS 8

Run the following commands:

```
# The following two lines might not be required in all situations.
sudo sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-*
sudo sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/
↪yum.repos.d/CentOS-*

sudo dnf config-manager --set-enabled PowerTools
sudo yum install -y epel-release
sudo yum check-update -y

sudo yum install -y gcc-c++ make git
sudo yum install -y python3 python3-devel python3-pip

sudo python3 -m pip install cmake
```

2.2 Mac OS

We require that a C++ compiler is installed on your system. There are two options you can choose from:

1. Using Homebrew
2. Using MacPorts

Before moving on, install the XCode command line tools by opening a terminal window and running the following command:

```
xcode-select --install
```

2.2.1 Homebrew

Install Homebrew with the following command:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/
↪install)"
```

Then proceed to install Python as well as a C++ compiler (note: gcc installed via Homebrew may lead to some issues therefore we choose clang):

```
brew install python llvm
```

Then install the rest of the required libraries/programs using the following command:

```
sudo port install cmake
```

2.2.2 MacPorts

Visit macports.org and install the latest version that corresponds to your operating system's version. Afterwards, open a new terminal window.

Then, use macports to install Python 3.8 with pip by entering the following command

```
sudo port install python38 py38-pip
```

A warning might show up about using Python from the terminal.In this case, you should also install

```
sudo port install py38-gnureadline
```

Then install the rest of the required libraries/programs and a C++ compiler using the following command:

```
sudo port install cmake clang-14
```

2.3 Windows

On Windows, you may compile MindQuantum using any of the following methods:

- *Visual Studio* 2019 or more recent
- *MSYS2* using either MSYS2-MSYS or MSYS2-MINGW64
- *Cygwin*
- *MinGW64*

While we cannot provide an exhaustive guide on how to compile using each of the aforementioned methods, you can use the following as a starting point. Also note that most if not all of the above are testing using GitHub actions. When in doubt, you may have a look at the workflow configuration file to see exactly how MindQunantum is compiled there.

2.3.1 Visual Studio

You may either install Visual Studio 2019 or more recent using the installer provided by Microsoft or use the Chocolatey package manager. Note that in some cases, the automatic build of the Boost libraries during the CMake call might fail. In that case, we would suggest that you compile and install those libraries separately and then attempt building MindQuantum again.

In the following, all the commands are to be run from within a PowerShell window. In some cases, you might need to run PowerShell as administrator.

Pure Windows install

Install Python using the installer provided at https://www.python.org/downloads/.

Chocolatey

First install Chocolatey using the installer following the instructions on their website. Once that is done, you can start by installing some of the required packages. Reboot as needed during the process.

```
choco install -y visualstudio2019-workload-vctools --includeOptional
choco install -y windows-sdk-10-version-2004-all
choco install -y cmake git
```

Installing Python is as simply as running the following commands:

```
choco install -y python3 --version 3.9.11
cmd /c mklink "C:\Python38\python3.exe" "C:\Python38\python.exe"
```

2.3.2 MSYS2

Install MSYS2 using the installer provided at https://www.msys2.org/.

MSYS2-MSYS

From within an MSYS2-MSYS shell, run the following command in order to install the required programs and libraries:

```
pacman -Syu
pacman -S git base-devel gcc cmake python-devel python-pip gmp-devel
```

MSYS2-MINGW64

From within an MSYS2-MINGW64 shell, run the following command in order to install the required programs and libraries:

```
pacman -Syu
pacman -S git patch make mingw-w64-x86_64-toolchain mingw-w64-x86_64-cmake \
            mingw-w64-x86_64-python mingw-w64-x86_64-python-pip
```

Note: When using MSYS2-MINGW64, you will need to use the "MSYS Makefiles" generator for CMake. Simply provide `-G "MSYS Makefiles"` on the command line as argument to CMake.

2.3.3 Cygwin

Install Cygwin using the installer provided at https://www.cygwin.com/install.html.

Then install the following packages:

- autoconf
- automake
- binutils
- m4
- make
- cmake
- patch
- gzip
- bzip2
- tar
- xz
- flex
- file
- findutils
- groff

- gawk
- sed
- libtool
- gettext
- wget
- curl
- grep
- dos2unix
- git
- gcc-core
- gcc-g++
- libgmp-devel
- python3
- python3-devel
- python3-pip
- python3-virtualenv

2.3.4 MinGW64

Install MinGW64 by following the instructions at https://www.mingw-w64.org/downloads/.

You then will want to install Python; e.g. using the installer provided at https://www.python.org/downloads/ and then install CMake using Pip:

```
python -m pip install --user cmake
```

CHAPTER

THREE

CMAKE REFERENCE

3.1 CMake options

Here is an exhaustive list of all CMake options available for customization.

3.1.1 Descriptions

Option name	Description
BUILD_SHARED_LIBS	Build shared libs
BUILD_TESTING	Enable building the test suite
CLEAN_3RDPARTY_INSTALL_DIR	Clean third-party installation directory
CUDA_ALLOW_UNSUPPORTED_COMPILER	Allow the use of an unsupported comipler version for CUDA
CUDA_STATIC	Use static versions of the Nvidia CUDA libraries
DISABLE_FORTRAN_COMPILER	Forcefully disable the Fortran compiler for some 3rd party libraries
ENABLE_CMAKE_DEBUG	Enable verbose output to debug CMake issues
ENABLE_CUDA	Enable the use of CUDA code
ENABLE_CXX_EXPERIMENTAL	Enable the building of the (new) experimental C++ backend
ENABLE_GITEE	Use Gitee instead of GitHub for (some) third-party dependencies
ENABLE_MD	Use /MD, /MDd flags when compiling (MSVC only)
ENABLE_MT	Use /MT, /MTd flags when compiling (MSVC only)
ENABLE_PROFILING	Enable compilation with profiling flags
ENABLE_PROJECTQ	Enable ProjectQ support
ENABLE_RUNPATH	Prefer RUNPATH over RPATH when linking
ENABLE_STACK_PROTECTION	Enable stack protection during compilation
IN_PLACE_BUILD	Build the C++ MindQuantum libraries in-place
IS_PYTHON_BUILD	Whether CMake is called from setup.py
LINKER_DTAGS	Enable –enable-new-dtags (else use –disable-new-dtags)
LINKER_NOEXECSTACK	Use *-z,noexecstack* during linking (if supported)
LINKER_RELRO	Use *-z,relro* during linking (if supported)
LINKER_RPATH	Use RUNPATH/RPATH related flags when compiling
LINKER_STRIP_ALL	Use *–strip-all* during linking (if supported)
USE_OPENMP	Use the OpenMP library for parallelisation
USE_PARALLEL_STL	Use the parallel STL for parallelisation (using TBB or else)
USE_VERBOSE_MAKEFILE	Generate verbose Makefiles (if supported)

3.1.2 Default values

Option name	Default value
BUILD_SHARED_LIBS	OFF
BUILD_TESTING	OFF
CLEAN_3RDPARTY_INSTALL_DIR	OFF
CUDA_ALLOW_UNSUPPORTED_COMPILER	OFF
CUDA_STATIC	OFF
DISABLE_FORTRAN_COMPILER	ON
ENABLE_CMAKE_DEBUG	OFF
ENABLE_CUDA	OFF
ENABLE_CXX_EXPERIMENTAL	OFF
ENABLE_GITEE	OFF
ENABLE_MD	OFF
ENABLE_MT	OFF
ENABLE_PROFILING	OFF
ENABLE_PROJECTQ	ON
ENABLE_RUNPATH	ON
ENABLE_STACK_PROTECTION	ON
IN_PLACE_BUILD	OFF
IS_PYTHON_BUILD	OFF
LINKER_DTAGS	ON
LINKER_NOEXECSTACK	ON
LINKER_RELRO	ON
LINKER_RPATH	ON
LINKER_STRIP_ALL	ON
USE_OPENMP	ON
USE_PARALLEL_STL	OFF
USE_VERBOSE_MAKEFILE	ON

3.1.3 Detailed descriptions

CLEAN_3RDPARTY_INSTALL_DIR
: This will delete any pre-existing installations within the local installation directory (by default `/path/to/build/.mqlibs`) _except_ the ones that are currently needed based on the hashes of the third-party libraries.

DISABLE_FORTRAN_COMPILER
: This currently only has an effect when installing Eigen3.

3.2 CMake variables

In addition to the above CMake options, you may pass certain special CMake variables in order to customize your build. These are described below in more details.

MQ_FORCE_LOCAL_PKGS

This variable value is case-insensitive. It may be either of:

- a single string (`all`)
- a comma-separated list of CMake package names for one or more of MindQuantum's third-party dependencies (e.g. `gmp,eigen3`)

Any or all packages listed will be compiled locally during the CMake configuration process.

`MQ_XXX_FORCE_LOCAL`
: Setting this to a truthful value for one of MindQuantum's third-party dependencies will result in that/these packages to be compiled locally during the CMake configuration process. Note that the package name `XXX` must be all caps.

CHAPTER

FOUR

FIRST EXPERIENCE

4.1 Build parameterized quantum circuit

The below example shows how to build a parameterized quantum circuit.

```
from mindquantum import *
import numpy as np

encoder = Circuit().h(0).rx({'a0': 2}, 0).ry('a1', 1)
print(encoder)
print(encoder.get_qs(pr={'a0': np.pi / 2, 'a1': np.pi / 2}, ket=True))
```

Then you will get,

```
q0: ────H────────RX(2*a0)──

q1: ──RY(a1)─────────────────

-1/2j¦00
-1/2j¦01
-1/2j¦10
-1/2j¦11
```

In jupyter notebook, we can just call `svg()` of any circuit to display the circuit in svg picture (`dark` and `light` mode are also supported).

```
circuit = (qft(range(3)) + BarrierGate(True)).measure_all()
circuit.svg()
```

4.2 Train quantum neural network

```
ansatz = CPN(encoder.hermitian(), {'a0': 'b0', 'a1': 'b1'})
sim = Simulator('projectq', 2)
ham = Hamiltonian(-QubitOperator('Z0 Z1'))
grad_ops = sim.get_expectation_with_grad(
    ham,
    encoder + ansatz,
    encoder_params_name=encoder.params_name,
    ansatz_params_name=ansatz.params_name,
```

(continues on next page)

(continued from previous page)

```
)

import mindspore as ms

ms.context.set_context(mode=ms.context.PYNATIVE_MODE, device_target='CPU')
net = MQLayer(grad_ops)
encoder_data = ms.Tensor(np.array([[np.pi / 2, np.pi / 2]]))
opti = ms.nn.Adam(net.trainable_params(), learning_rate=0.1)
train_net = ms.nn.TrainOneStepCell(net, opti)
for i in range(100):
    train_net(encoder_data)
print(dict(zip(ansatz.params_name, net.trainable_params()[0].asnumpy())))
```

The trained parameters are,

```
{'b1': 1.5720831, 'b0': 0.006396801}
```

CHAPTER

FIVE

TUTORIALS

1. Basic usage
    - Variational Quantum Circuit
2. Variational quantum algorithm
    - Quantum Approximate Optimization Algorithm
    - The Application of Quantum Neural Network in NLP
    - VQE Application in Quantum Chemistry Computing

CHAPTER

SIX

API

API of the Python and C++ code of MindQuantum. To have a Python frontend and C++ in the backend, we make use of Pybind11. The C++ code represents quantum circuits with the help of Tweedledum networks, which can also be turned into directed acyclic graphs (DAG), to make some operations more efficient.

6.1 Python

6.2 C++

This is the documentation of the C++ engine classes and decomposition rule functions of MindQuantum, mainly intended for developers.

6.2.1 C++ Experimental

This document describes the new C++ backend API to MindQuantum.

C++ cengines

namespace **cengines**

Typedefs

using **engine_t** = std::variant<*cpp*::*LocalOptimizer*, *cpp*::*LinearMapper*, *cpp*::*GridMapper*, *CppGraphMapper*, *CppPrinter*, *ResourceCounter*, *cpp*::*TagRemover*, *cpp*::*InstructionFilter*>

class **ComputeCircuit**

#include <*compute.hpp*> Circuit wrapper class that implements the compute-uncompute pattern.

Public Types

using **cbit_t** = tweedledum::Cbit

Public Functions

inline explicit **ComputeCircuit**(circuit_t &original)

Constructor.

Parameters

original – A quantum circuit

inline **~ComputeCircuit**()

Destructor.

Only when the *ComputeCircuit* object is being destroyed are the instructions added to it so far will be transferred to the original quantum circuit object.

ComputeCircuit(const *ComputeCircuit*&) = delete

ComputeCircuit(*ComputeCircuit*&&) = default

ComputeCircuit &**operator=**(const *ComputeCircuit*&) = delete

ComputeCircuit &**operator=**(*ComputeCircuit*&&) = delete

inline void **done_compute**()

For internal-use only.

inline circuit_t &**compute**()

Read-write getter to the circuit storing the *computed* instructions.

inline circuit_t &**non_compute**()

Read-write getter to the circuit storing the instructions between the compute and uncompute regions.

class **ControlledCircuit**

#include <control.hpp>

Public Types

using **cbit_t** = tweedledum::Cbit

Public Functions

inline **ControlledCircuit**(circuit_t &original, const qubits_t &controls)

inline **~ControlledCircuit**()

ControlledCircuit(const *ControlledCircuit*&) = delete

ControlledCircuit(*ControlledCircuit*&&) = default

ControlledCircuit &**operator=**(const *ControlledCircuit*&) = delete

ControlledCircuit &**operator=**(*ControlledCircuit*&&) = delete

inline void **apply**()

inline explicit **operator circuit_t&**()

class **CppGraphMapper**

#include <cpp_graph_mapper.hpp> C++ class to represent an arbitrary graph mapper.

This is intended to be instantiated in Python by users in order to define the hardware architecture they want to use for mapping

Subclassed by mindquantum::python::CppGraphMapper

Public Types

using **device_t** = tweedledum::Device

using **circuit_t** = tweedledum::Circuit

using **placement_t** = tweedledum::Placement

using **mapping_t** = tweedledum::Mapping

using **mapping_ret_t** = std::pair<*circuit_t*, *mapping_t*>

using **mapping_param_t** = std::variant<mapping::sabre_config, mapping::jit_config>

using **edge_list_t** = std::vector<std::tuple<uint32_t, uint32_t>>

Public Functions

explicit **CppGraphMapper**(const *mapping_param_t* &params)

Constructor with empty graph.

The mapping algorithm used by the mapper is defined by the type of the parameters that is passed in argument. The mapper currently supports three mapping algorithms:

- SABRE (SWAP-based Bidirectional heuristic search algorithm)
- JIT (just-in-time algorithm)
- SAT (boolean satisfiability problem solver)

Parameters
params – Parameters for the mapping algorithm

CppGraphMapper(uint32_t num_qubits, const *edge_list_t* &edge_list, const *mapping_param_t* &params)

Constructor with graph defined by number of qubits and edges.

The mapping algorithm used by the mapper is defined by the type of the parameters that is passed in argument. The mapper currently supports three mapping algorithms:

- SABRE (SWAP-based Bidirectional heuristic search algorithm)

- ZDD (zero-suppressed decision diagram)
- SAT (boolean satisfiability problem solver)

Parameters
- **num_qubits** – Number of qubits
- **edge_list** – List of edges in the graph
- **params** – Parameters for the mapping algorithm

inline const auto &**device**() const

Simple getter for the underlying device used by this mapper.

mapping_ret_t **hot_start**(const *device_t* &device, const *circuit_t* &circuit, *placement_t* &placement) const

Apply the mapping algorithm to a network given an initial mapping.

This method will use the architecture defined within the instance of the *CppGraphMapper*, as well as the mapping parameters in order to choose which mapping algorithm to call.

Parameters
state – Current mapping state

mapping_ret_t **cold_start**(const *device_t* &device, const *circuit_t* &circuit) const

Apply the mapping algorithm to a network without an initial mapping.

This method will use the architecture defined within the instance of the *CppGraphMapper*, as well as the mapping parameters in order to choose which mapping algorithm to call.

Parameters
state – Current mapping state

inline const auto &**get_mapping_parameters**() const

Simple getter for mapping parameters.

void **path_device**(uint32_t num_qubits, bool cyclic = false)

Set device graph to be a 1D arrangement of qubits.

void **grid_device**(uint32_t num_columns, uint32_t num_rows)

Set device graph to be a 2D grid of qubits.

class **CppPrinter**

#include <cpp_printer.hpp> C++ class to represent a command printer with seleprojectqctable language.

This is intended to be instantiated in Python by users in order to define the output language they want to use

Subclassed by mindquantum::python::CommandPrinter

Public Types

enum **language_t**

Values:

enumerator **projectq**

enumerator **openqasm**

enumerator **qasm**

enumerator **qiskit**

Public Functions

explicit **CppPrinter**(*language_t* language) noexcept

Constructor.

This is intended to be instantiated in Python by users in order to print all the gates stored in the C++ network.

Parameters

language – The format in which the network should be printed.g

explicit **CppPrinter**(std::string_view language)

Constructor.

This is intended to be instantiated in Python by users in order to print all the gates stored in the C++ network.

Parameters

language – The format in which the network should be printed.

Throws

`std::runtime_error` – if format is invalid

template<typename **circuit_t**>
void **print_output**(const *circuit_t* &circuit, std::ostream &output_stream)

Print circuit to using specified format.

Parameters

- **circuit** – The network to be printed
- **output_stream** – Where to print the network Tweedledum ids

Public Static Functions

static constexpr std::string_view **lang_to_str**(*language_t* language)

Convert language_t to string.

static constexpr std::optional<*language_t*> **str_to_lang**(std::string_view language)

Convert string to language_t.

struct **ResourceCounter**

#include <cpp_resource_counter.hpp> C++ equivalent to projectq.backends.ResourceCounter.

Prints all gate classes and specific gates it encountered (cumulative over several flushes)

Subclassed by mindquantum::python::ResourceCounter

Public Types

using **param_t** = std::optional<double>

using **ctrl_count_t** = std::size_t

using **class_desc_t** = std::pair<std::string_view, *ctrl_count_t*>

using **gate_desc_t** = std::tuple<std::string_view, *param_t*, *ctrl_count_t*>

Public Functions

void **add_gate_count**(std::string_view kind, *param_t* param, std::size_t n_controls, std::size_t count)

void **add_gate_counts**(const tweedledum::Circuit &network)

Add gates in Network to gate (class) counts.

Public Members

std::size_t **max_width_**

std::map<*class_desc_t*, std::size_t> **gate_class_counts_**

std::map<*gate_desc_t*, std::size_t> **gate_counts_**

void ***origin_**

namespace **cpp**

Typedefs

using **mapping_t** = std::map<unsigned int, unsigned int>

class **GridMapper**

#include <*cpp_mapping.hpp*> Subclassed by mindquantum::python::cpp::GridMapper

Public Functions

DECLARE_ATTRIBUTE(*mapping_t*, _current_mapping)

DECLARE_ATTRIBUTE(unsigned int, num_qubits)

DECLARE_ATTRIBUTE(unsigned int, num_mappings)

DECLARE_ATTRIBUTE(unsigned int, storage)

DECLARE_ATTRIBUTE(unsigned int, num_rows)

DECLARE_ATTRIBUTE(unsigned int, num_columns)

class **InstructionFilter**

#include <cpp_engine_list.hpp> Subclassed by mindquantum::python::cpp::InstructionFilter

class **LinearMapper**

#include <cpp_mapping.hpp> Subclassed by mindquantum::python::cpp::LinearMapper

Public Functions

DECLARE_ATTRIBUTE(*mapping_t*, _current_mapping)

DECLARE_ATTRIBUTE(unsigned int, num_qubits)

DECLARE_ATTRIBUTE(unsigned int, num_mappings)

DECLARE_ATTRIBUTE(unsigned int, storage)

DECLARE_ATTRIBUTE(bool, cyclic)

class **LocalOptimizer**

#include <cpp_optimisation.hpp> Subclassed by mindquantum::python::cpp::LocalOptimizer

Public Members

unsigned int **_m_**

class **TagRemover**

#include <cpp_engine_list.hpp> Subclassed by mindquantum::python::cpp::TagRemover

namespace **details**

class **ComputeCircuitProxy**

#include <compute.hpp> Helper class to implement WITH_COMPUTE statements in C++.

Public Functions

inline explicit **ComputeCircuitProxy**(*ComputeCircuit* &compute)

Constructor.

Parameters

compute – ComputCircuit object to wrap.

inline **~ComputeCircuitProxy**()

Destructor.

ComputeCircuitProxy(const *ComputeCircuitProxy*&) = delete

ComputeCircuitProxy(*ComputeCircuitProxy*&&) = delete

ComputeCircuitProxy &**operator=**(const *ComputeCircuitProxy*&) = delete

ComputeCircuitProxy &**operator=**(*ComputeCircuitProxy*&&) = delete

inline **operator circuit_t&**() &

Read-write getter to the circuit storing the *computed* instructions.

C++ core

namespace **core**

class **CppCore**

#include <cpp_core.hpp> Subclassed by mindquantum::python::CppCore

Public Types

using **gate_t** = *ops*::*Command*::*gate_t*

Provide ProjectQ's functionality in C++.

using **engine_list_t** = std::vector<*cengines*::*engine_t*>

using **circuit_t** = td::Circuit

using **instruction_t** = CircuitManager::instruction_t

using **Complex** = std::complex<double>

using **c_type** = std::complex<double>

using **ArrayType** = std::vector<*c_type*, ::projectq::aligned_allocator<*c_type*, 64>>

using **MatrixType** = std::vector<*ArrayType*>

Public Functions

CppCore()

CppCore(*CppCore* const&) = delete

CppCore **operator=**(*CppCore* const&) = delete

inline bool **sim_backend**() const

void **set_engine_list**(const *engine_list_t* &engine_list)

void **set_simulator_backend**(::projectq::Simulator &sim)

void **allocate_qubit**(unsigned id)

Allocate a single qubit.

void **apply_command**(const *ops*::*Command* &cmd)

Apply a ProjectQ command.

void **flush**()

Execute stored gates.

void **traverse_engine_list**()

Go through all engines in engine_list_.

Gets called in beginning of *flush()*

std::map<unsigned, bool> **get_measure_info**()

Return measurement outcomes.

Returns

A map where the key values are the measured qubits' IDs and the mapped values are the measurement outcomes

void **set_output_stream**(std::string_view file_name)

Set output file name (stdout for printing to standard output)

void **write**(std::string_view format)

Output Quantum circuit to standard output.

Parameters

format – in which to output circuit

inline auto **cheat**()

Return state vector.

Returns

A map where the key values are the allocated qubits' ids and the mapped values are their positions in the ket: |q_n, q_(n-1), ..., q_1, q_0>

Returns

The current state vector of the allocated qubits

C++ decompositions

namespace **decompositions**

Typedefs

using **num_target_t** = uint32_t

using **num_control_t** = int32_t

using **num_param_t** = uint32_t

using **GateDecompositionRule** = *GateDecompositionRuleCXX17*<derived_t, kinds_t, num_targets, num_controls, num_params, atoms_t...>

using **ParametricSimpleAtom** = *ParametricAtom*<op_t, traits::num_targets<op_t>, num_controls_>

using **TrivialSimpleAtom** = *TrivialAtom*<op_t, traits::num_targets<op_t>, num_controls_>

using **circuit_t** = tweedledum::Circuit

using **instruction_t** = tweedledum::Instruction

Functions

struct mindquantum::decompositions::DecompositionRuleParam MQ_ALIGN (16)

void **decompose_cnot2cz**(*circuit_t* &result, const *instruction_t* &inst)

void **decompose_cnot2rxx_M**(*circuit_t* &result, const *instruction_t* &inst)
Decompose CNOT gate, M for 'Minus' because ends with Ry(-pi/2)

void **decompose_cnot2rxx_P**(*circuit_t* &result, const *instruction_t* &inst)
Decompose CNOT gate, M for 'Minus' because ends with Ry(+pi/2)

void **decompose_cnu2toffoliandcu**(*circuit_t* &result, const *instruction_t* &inst)
Decompose multi-controlled gates.

void **decompose_entangle**(*circuit_t* &result, const *instruction_t* &inst)
Decompose entangle into h and cnot/cx gates.

void **decompose_ph2r**(*circuit_t* &result, const *instruction_t* &inst)
Decompose controlled global phase gates into (controlled) R/r1.

Shaves off one control qubit when applied

void **decompose_PhNoCtrl**(*circuit_t* &result, const *instruction_t* &inst)
Delete all phase/ph gates without controls.

void **decompose_h2rx_M**(*circuit_t* &result, const *instruction_t* &inst)
Decompose H gate, M for 'Minus' because ends with Ry(-pi/2)

void **decompose_h2rx_N**(*circuit_t* &result, const *instruction_t* &inst)
Decompose H gate, N for 'Neutral'.

void **decompose_qft2crandhadamard**(*circuit_t* &result, const *instruction_t* &inst)
Decompose QFT (Quantum Fourier Transform) into h and controlled R (cr1)

void **decompose_qubitop2onequbit**(*circuit_t* &result, const *instruction_t* &inst)
Decompose (controlled) Qubit Operator into (controlled) Paulis.

void **decompose_rx2rz**(*circuit_t* &result, const *instruction_t* &inst)

void **decompose_ry2rz**(*circuit_t* &result, const *instruction_t* &inst)

void **decompose_rz2rx_P**(*circuit_t* &result, const *instruction_t* &inst)

void **decompose_rz2rx_M**(*circuit_t* &result, const *instruction_t* &inst)

void **decompose_r2rzandph**(*circuit_t* &result, const *instruction_t* &inst)
Decompose (controlled) phase-shift gate using z-rotation and global phase.

void **decompose_sqrtswap2cnot**(*circuit_t* &result, const *instruction_t* &inst)
Decompose sqrtswap into controlled x and sqrtx.

void **decompose_swap2cnot**(*circuit_t* &result, const *instruction_t* &inst)
Decompose swap into controlled x gates.

bool **recognize_time_evolution_commuting**(const *instruction_t* &inst)

void **decompose_time_evolution_commuting**(*circuit_t* &result, const *instruction_t* &inst)

bool **recognize_time_evolution_individual_terms**(const *instruction_t* &inst)

void **decompose_time_evolution_individual_terms**(*circuit_t* &result, const *instruction_t* &inst)

void **decompose_toffoli2cnotandtgate**(*circuit_t* &result, const *instruction_t* &inst)
Decompose toffoli gate (ccx) into cx, t, tdg and h gates.

Variables

static constexpr auto **any_target** = *num_target_t*(0)
Constant representing no constraints on the number of target qubits.

static constexpr auto **any_control** = *num_control_t*(-1)
Constant representing no constraints on the number of control qubits.

class mindquantum::*decompositions*::*DecompositionRule* **MQ_ALIGN**

class **AtomStorage**
#include <atom_storage.hpp>

Public Types

using **atom_t** = *DecompositionAtom*

using **map_t** = std::map<std::pair<std::string, *num_control_t*>, *atom_t*, *details*::*atom_less*>

Public Functions

inline MQ_NODISCARD auto **size**() const noexcept

Return the number of decomposition atoms in the storage.

Returns

Pointer to inserted element, pointer to compatible element or nullptr.

template<typename o_atom_t> MQ_NODISCARD bool has_atom () const noexcept

Check whether a matching (gate) atom can be found in the storage.

The comparison is performed based on the value of `o_atom_t::num_controls()`, `o_atom_t::name()`, as well as taking `o_atom_t::kinds_t` into account. The matching for the kind is performed by calling `atom->is_kind(type::kind())` for each operator contained in `o_atom_t::kinds_t`.

See also:

has_atom(num_control_t num_controls, std::string_view name) const

Template Parameters

o_atom_t – Type of atom to look for

Returns

True/false depending on whether the atom can be found or not

template<typename o_atom_t, typename... op_t> MQ_NODISCARD bool has_atom (num_control_t num_controls, std::string_view name) const noexcept

Check whether a matching (gate) atom can be found in the storage.

Note: This method does not take general decompositions into account.

Parameters

- **kind** – Kind of atom to look for
- **num_controls** – Number of control the atom must be constrained by
- **name** – Name of atom to look for

Returns

True/false depending on whether the atom can be found or not

MQ_NODISCARD atom_t * get_atom_for (const instruction_t &inst) noexcept

Look for a suitable decomposition atom within the storage.

Parameters

inst – An instruction

Returns

Pointer to atom if any, `nullptr` otherwise

`template<typename o_atom_t, std::size_t kind_idx = 0, typename... args_t> MQ_NODISCARD atom_t * add_or_compatible_atom (args_t &&... args)`

Inserts a new element, constructed in-place with the given args or returns an existing compatible atom.

This is different from add_or_return_atom in the sense that it does not enforce an exact match for the atom.

Template Parameters

- **o_atom_t** – Type of atom to insert/return
- **kind_idx** – If the atom has multiple element in its kinds_t tuple, this is the index of the type in that tuple to use to register the atom inside the storage.

Returns

Pointer to inserted element, pointer to compatible element.

`template<typename o_atom_t, std::size_t kind_idx = 0, typename... args_t> MQ_NODISCARD atom_t * add_or_return_atom (args_t &&... args)`

Inserts a new element, constructed in-place with the given args or returns an existing one.

This is different from add_or_compatible_atom in that it looks for an exact match.

Template Parameters

- **o_atom_t** – Type of atom to insert/return
- **kind_idx** – If the atom has multiple element in its kinds_t tuple, this is the index of the type in that tuple to use to register the atom inside the storage.

Returns

Pointer to inserted element, pointer to compatible element.

`template<typename o_atom_t, std::size_t kind_idx = 0, typename... args_t> MQ_NODISCARD atom_t * add_or_replace_atom (args_t &&... args)`

Inserts or replaces a new element, constructed in-place with the given args.

Template Parameters

- **o_atom_t** – Type of atom to insert/replace
- **kind_idx** – If the atom has multiple element in its kinds_t tuple, this is the index of the type in that tuple to use to register the atom inside the storage.

Returns

Pointer to inserted element, pointer to compatible element or nullptr.

class **DecompositionAtom**

#include <decomposition_atom.hpp>

Public Types

using **gate_param_t** = mindquantum::*ops*::*parametric*::*gate_param_t*

Public Functions

template<typename **atom_t**, typename = std::enable_if_t<!std::is_same_v<std::remove_cvref_t<*atom_t*>, *DecompositionAtom*>>>
inline **DecompositionAtom**(*atom_t* &&atom) noexcept

inline **DecompositionAtom**(const *DecompositionAtom* &other) noexcept

inline *DecompositionAtom* &**operator=**(const *DecompositionAtom* &other) noexcept

inline **DecompositionAtom**(*DecompositionAtom* &&other) noexcept

inline *DecompositionAtom* &**operator=**(*DecompositionAtom* &&other) noexcept

inline **~DecompositionAtom**() noexcept

inline MQ_NODISCARD auto **name**() const noexcept

Return the name of this decomposition atom.

inline MQ_NODISCARD auto **is_kind**(std::string_view kind) const noexcept

Test whether an atom has (supports) a particular kind of operator.

inline MQ_NODISCARD bool is_applicable (const instruction_t &inst) const noexcept

Test whether this atom is applicable to a particular instruction.

Child classes may implement a method named `is_applicable_impl` in order to customize the default behaviour for this method.

Parameters

inst – An instruction

Returns

True if the atom can be applied, false otherwise

inline void **apply**(*circuit_t* &circuit, const *instruction_t* &inst) noexcept

Apply a decomposition atom to decompose an instruction.

Parameters

- **circuit** – A quantum circuit to apply the decomposition atom to
- **inst** – A quantum instruction to decompose

Pre

is_applicable() returns true

inline void **apply**(*circuit_t* &circuit, const operator_t &op, const qubits_t &qubits, const cbits_t &cbits = {}) noexcept

Apply the atom (ie. the decomposition it represents) to a quantum circuit.

This overload assumes the decomposition atom is not parametric

Note: Currently the `cbits` parameter is not used at all! It is here to make the API futureproof.

Parameters

- **circuit** – A quantum circuit to apply the decomposition atom to

- **op** – A quantum operation
- **qubits** – A list of qubits to apply the decomposition atom
- **cbits** – A list of classical bit the decomposition applies to

template<class **atom_t**>

struct **Model**<*atom_t*, false>

#include <decomposition_atom.hpp>

Public Functions

inline explicit **Model**(*atom_t* &&op) noexcept

inline explicit **Model**(*atom_t* const &op) noexcept

Public Members

std::unique_ptr<*atom_t*> **operator_**

Public Static Functions

static inline auto ***self_cast**(void *self) noexcept

static inline auto ***self_cast**(const void *self) noexcept

static inline void **dtor**(void *self) noexcept

static inline void **clone**(void const *self, void *other) noexcept

static inline constexpr std::string_view **name**() noexcept

static inline constexpr bool **is_kind**(std::string_view kind) noexcept

static inline bool **is_applicable**(void const *self, const *instruction_t* &inst) noexcept

static inline void **apply**(void *self, *circuit_t* &circuit, const *instruction_t* &inst) noexcept

static inline void **apply_operator**(void *self, *circuit_t* &circuit, const operator_t &op, const qubits_t &qubits, const cbits_t &cbits) noexcept

Public Static Attributes

static constexpr Concept **vtable_** = {*dtor*, *clone*, *name*, *is_kind*, *is_applicable*, *apply*, *apply_operator*}

template<class **atom_t**>

struct **Model**<*atom_t*, true>

#include <decomposition_atom.hpp>

Public Types

using **type** = *atom_t*

Public Functions

inline explicit **Model**(*atom_t* &&op) noexcept

inline explicit **Model**(*atom_t* const &op) noexcept

Public Members

atom_t **operator_**

Public Static Functions

static inline auto ***self_cast**(void *self) noexcept

static inline auto ***self_cast**(const void *self) noexcept

static inline void **dtor**(void *self) noexcept

static inline void **clone**(void const *self, void *other) noexcept

static inline constexpr std::string_view **name**() noexcept

static inline constexpr bool **is_kind**(std::string_view kind) noexcept

static inline bool **is_applicable**(void const *self, const *instruction_t* &inst) noexcept

static inline void **apply**(void *self, *circuit_t* &circuit, const *instruction_t* &inst) noexcept

static inline void **apply_operator**(void *self, *circuit_t* &circuit, const operator_t &op, const qubits_t &qubits, const cbits_t &cbits) noexcept

Public Static Attributes

static constexpr Concept **vtable_** = {*dtor*, *clone*, *name*, *is_kind*, *is_applicable*, *apply*, *apply_operator*}

template<typename **derived_t**, typename ...**atoms_t**>

class **DecompositionRule**

#include <decomposition_rule.hpp>

Public Types

using **base_t** = *DecompositionRule*

using **self_t** = *DecompositionRule*<*derived_t*, *atoms_t*...>

using **gate_param_t** = *ops*::*parametric*::*gate_param_t*

Public Functions

explicit **DecompositionRule**(*AtomStorage* &storage)

Constructor.

Note: The atoms of this decomposition rule will be insertd into the storage if they are not already present. However, existing (exactly matching) atoms will not be replaced.

Parameters
storage – Atom storage within which this decomposition will live in

template<std::size_t idx> constexpr auto * atom () noexcept MQ_REQUIRES((idx< sizeof. ..(atoms_t)))

Getter function for the individual atoms.

Overload using a non-type template parameter corresponding to the index of the atom in the atom list.

Template Parameters
idx – Index of atom in atom list

template<typename atom_t> constexpr auto *atom() noexcept MQ_REQUIRES((concepts atom (circuit_t const operator_t&op, const qubits_t &qubits, const cbits_t &cbits) noexceptvoid

Getter function for the individual atoms.

Overload using a type template type parameter. This works since the list of atoms contains only unique values.

Note: Currently the `cbits` parameter is not used at all! It is here to make the API futureproof.

Template Parameters
atom_t – Type of atom to look for.Apply a decomposition
Parameters
- **circuit** – A quantum circuit to apply the decomposition atom to
- **op** – A quantum operation to decompose
- **qubits** – A list of qubits to apply the decomposition atom
- **cbits** – A list of classical bit the decomposition applies to

Public Static Functions

static inline constexpr auto **name**() noexcept

Return the name of this *DecompositionRule*.

template<typename ...**args_t**>
static inline MQ_NODISCARD auto **create**(*AtomStorage* &storage, *args_t*&&... args) noexcept

Helper function to create a *DecompositionRule* instance.

Parameters

storage – Atom storage within which this decomposition will live in

struct **DecompositionRuleParam**

#include <decomposition_param.hpp> Aggregate type to store *DecompositionRule* template parameters.

Public Members

num_target_t **num_targets**

Number of target qubits the decomposition is constrained on.

num_control_t **num_controls**

Number of control qubits the decomposition is constrained on.

num_param_t **num_params**

Number of parameters the decomposition rule possesses.

class **GateDecomposer**

#include <gate_decomposer.hpp>

Public Types

using **atom_storage_t** = *AtomStorage*

using **atom_t** = *atom_storage_t*::atom_t

using **general_rule_storage_t** = std::set<*DecompositionAtom*, *details*::*rules_less*>

Public Functions

inline MQ_NODISCARD auto **num_atoms**() const noexcept

Return the number of atoms in the internal storage.

inline MQ_NODISCARD auto **num_rules**() const noexcept

Return the number of general decomposition rules in the internal storage.

inline MQ_NODISCARD const auto & storage () const noexcept

Simple getter to the internal storage.

`template<typename o_atom_t> MQ_NODISCARD bool has_atom () const noexcept`

Check whether a matching (gate) atom can be found in the storage.

The comparison is performed based on the value of `o_atom_t::num_controls()`, `o_atom_t::name()`, as well as taking `o_atom_t::kinds_t` into account. The matching for the kind is performed by calling `atom->is_kind(type::kind())` for each operator contained in `o_atom_t::kinds_t`.

Note: This method does not take general decompositions into account.

Template Parameters
`o_atom_t` – Type of atom to look for

Returns
True/false depending on whether the atom can be found or not

`MQ_NODISCARD atom_t * get_atom_for (const instruction_t &inst) noexcept`

Look for a suitable decomposition within the storages.

This method will favour gate decompositions over general decompositions when looking for a match.

Parameters
`inst` – An instruction

Returns
Pointer to atom if any, `nullptr` otherwise

`template<typename o_atom_t, std::size_t kind_idx = 0, typename... args_t> MQ_NODISCARD atom_t * add_or_replace_atom (args_t &&... args)`

Inserts a new element, constructed in-place with the given args.

The type `o_atom_t` will be used to determine where the atom will be stored; ie. whether it is a gate decomposition or a general decomposition.

Template Parameters
- `o_atom_t` – Type of atom to insert/replace
- `kind_idx` – If the atom has multiple element in its kinds_t tuple, this is the index of the type in that tuple to use to register the atom inside the storage.

Returns
Pointer to inserted element, pointer to compatible element or nullptr.

template<typename **derived_t**, typename **kinds_t_**, *DecompositionRuleParam* **param_** = *tparam*::*default_t*, typename ...**atoms_t**>
class **GateDecompositionRule** : public mindquantum::*decompositions*::*DecompositionRule*<*derived_t*, *atoms_t*...>, public mindquantum::traits::controls<*param_*.num_controls>

#include <gate_decomposition_rule_cxx20.hpp>

Public Types

using **base_t** = *GateDecompositionRule*

using **kinds_t** = *kinds_t_*

using **self_t** = *GateDecompositionRule*<*derived_t*, *kinds_t*, *param_*, *atoms_t*...>

using **gate_param_t** = *ops*::*parametric*::*gate_param_t*

using **double_list_t** = *ops*::*parametric*::*double_list_t*

using **param_list_t** = *ops*::*parametric*::*param_list_t*

Public Functions

template<typename rule_t> MQ_NODISCARD constexpr bool is_compatible () const noexcept

Check whether a decomposition is compatible with another one.

Another *GateDecompositionRule* instance is deemed compatible iff:

- the number of target qubit are identical
- the number of controls are compatible:
  - the number of control qubits in the decomposition rule is left undefined
  - or they have the same number of controls

Parameters

- **num_targets** – Number of target qubits of the operation to decompose
- **num_controls** – Number of control qubits of the operation to decompose

MQ_NODISCARD bool is_applicable (const instruction_t &inst) const noexcept

Check whether a decomposition is applicable with a given instruction.

Parameters

inst – A quantum instruction

Public Static Functions

static inline constexpr auto **num_targets**() noexcept

Return the number of target qubits this *DecompositionRule* is constrained on.

static inline constexpr auto **num_controls**() noexcept

Return the number of control qubits this *DecompositionRule* is constrained on.

static inline constexpr auto **num_params**() noexcept

Return the number of parameters of this *GateDecompositionRule*.

Public Static Attributes

static constexpr auto **is_parametric** = *param_*.num_params != 0U

Constant boolean indicating whether the *GateDecompositionRule* is parametric or not.

template<typename **derived_t**, typename **kinds_t_**, uint32_t **num_targets_**, *num_control_t* **num_controls_**, uint32_t **num_params_**, typename ...**atoms_t**>
class **GateDecompositionRuleCXX17** : public mindquantum::*decompositions*::*DecompositionRule*<*derived_t*, *atoms_t*...>, public mindquantum::traits::controls<*num_controls_*>

#include <gate_decomposition_rule_cxx17.hpp>

Public Types

using **base_t** = *GateDecompositionRuleCXX17*

using **kinds_t** = *kinds_t_*

using **parent_t** = *DecompositionRule*<*derived_t*, *atoms_t*...>

using **self_t** = *GateDecompositionRuleCXX17*<*derived_t*, *kinds_t*, *num_targets_*, *num_controls_*, *num_params_*, *atoms_t*...>

using **gate_param_t** = *ops*::*parametric*::*gate_param_t*

using **double_list_t** = *ops*::*parametric*::*double_list_t*

using **param_list_t** = *ops*::*parametric*::*param_list_t*

Public Functions

template<typename rule_t> MQ_NODISCARD constexpr bool is_compatible () const noexcept

Check whether a decomposition is compatible with another one.

Another *GateDecompositionRuleCXX17* instance is deemed compatible iff:

- the number of target qubit are identical
- the number of controls are compatible:
  - the number of control qubits in the decomposition rule is left undefined
  - or they have the same number of controls

Parameters

- **num_targets** – Number of target qubits of the operation to decompose
- **num_controls** – Number of control qubits of the operation to decompose

MQ_NODISCARD bool is_applicable (const instruction_t &inst) const noexcept

Check whether a decomposition is applicable with a given instruction.

Parameters

inst – A quantum instruction

Public Static Functions

static inline constexpr auto **num_targets**() noexcept

Return the number of target qubits this *DecompositionRule* is constrained on.

static inline constexpr auto **num_controls**() noexcept

Return the number of control qubits this *DecompositionRule* is constrained on.

static inline constexpr auto **num_params**() noexcept

Return the number of parameters of this *GateDecompositionRuleCXX17*.

Public Static Attributes

static constexpr auto **is_parametric** = *num_params_* != 0U

Constant boolean indicating whether the *GateDecompositionRuleCXX17* is parametric or not.

template<typename **derived_t**, typename ...**atoms_t**>

class **NonGateDecompositionRule** : public mindquantum::*decompositions*::*DecompositionRule*<*derived_t*, *atoms_t*...>

#include <non_gate_decomposition_rule.hpp>

Public Types

using **base_t** = *NonGateDecompositionRule*

using **self_t** = *NonGateDecompositionRule*<*derived_t*, *atoms_t*...>

using **parent_t** = *DecompositionRule*<*derived_t*, *atoms_t*...>

using **non_gate_decomposition** = void

Public Functions

inline explicit **NonGateDecompositionRule**(*AtomStorage* &storage)

template<typename **atom_t**, typename ...**args_t**>
constexpr auto ***atom**(*args_t*&&... args) noexcept

Getter function for the individual atoms.

Overload using a type template type parameter. This works since the list of atoms contains only unique values.

Template Parameters

atom_t – Type of atom to look for.

inline auto &**storage**() noexcept

void **apply**(*circuit_t* &circuit, const *instruction_t* &inst) noexcept

Apply a decomposition rule.

Parameters

- **circuit** – Quantum circuit
- **inst** – Quantum instructio to decompose

template<typename **op_t**, *num_target_t* **num_targets_** = *any_target*, *num_control_t* **num_controls_** = *any_control*>
class **ParametricAtom** : public mindquantum::traits::controls<*num_controls_*>

#include <parametric_atom.hpp> A decomposition atom representing a gate with some free-parameter(s)

Note: This must be a parametric gate with at least one free parameter

Template Parameters

- **operator_t** – Type of the gate the atom is representing
- **num_targets_** – Number of target qubits the gate this atom is representing has
- **num_controls_** – Number of control qubits the gate this atom is representing has. Possible values: -1 for any number of control qubits, >= 0 for a specified number of control qubits.

Public Types

using **subs_map_t** = mindquantum::*ops*::*parametric*::*subs_map_t*

using **double_list_t** = mindquantum::*ops*::*parametric*::*double_list_t*

using **param_list_t** = mindquantum::*ops*::*parametric*::*param_list_t*

using **self_t** = *ParametricAtom*<*op_t*, *num_targets_*, *num_controls_*>

using **kinds_t** = std::tuple<*op_t*>

Public Functions

inline explicit **ParametricAtom**(*AtomStorage*&)

MQ_NODISCARD bool is_applicable (const instruction_t &inst) const noexcept

Test whether this atom is applicable to a particular instruction.

Parameters

inst – An instruction

Returns

True if the atom can be applied, false otherwise

void **apply**(*circuit_t* &circuit, const operator_t &op, const qubits_t &qubits, const cbits_t &cbits) noexcept

Apply the atom (ie. the decomposition it represents) to a quantum circuit.

Note: Currently the `cbits` parameter is not used at all! It is here to make the API futureproof.

Parameters

- **circuit** – A quantum circuit to apply the decomposition atom to
- **op** – A quantum operation to decompose
- **qubits** – A list of qubits to apply the decomposition atom
- **param** – Some parameters to apply the decomposition atom with
- **cbits** – A list of classical bit the decomposition applies to

Public Static Functions

static inline MQ_NODISCARD constexpr std::string_view name () noexcept

Return the name of this decomposition atom.

static inline MQ_NODISCARD constexpr auto num_targets () noexcept

Return the number of target qubits this decomposition atom is constrained on.

static inline MQ_NODISCARD constexpr auto num_controls () noexcept

Return the number of control qubits this decomposition atom is constrained on.

static inline MQ_NODISCARD constexpr auto num_params () noexcept

Return the number of parameters of this decomposition atom.

static inline MQ_NODISCARD auto **create**(*AtomStorage* &storage) noexcept

Helper function to create an instance of this atom.

Parameters

storage – Atom storage within which this decomposition will live in

template<typename **op_t**, *num_target_t* **num_targets_** = *any_target*, *num_control_t* **num_controls_** = *any_control*>

class **TrivialAtom** : public mindquantum::traits::controls<*num_controls_*>

#include <trivial_atom.hpp> A decomposition atom representing a gate with no free-parameter.

Note: This can be a parametric gate with all of its parameters fully specified

Template Parameters

- **operator_t** – Type of the gate the atom is representing
- **num_controls_** – Number of control qubits the gate this atom is representing has. Possible values: -1 for any number of control qubits, >= 0 for a specified number of control qubits.

Public Types

using **self_t** = *TrivialAtom*<*op_t*, *num_targets_*, *num_controls_*>

using **kinds_t** = std::tuple<*op_t*>

Public Functions

inline explicit **TrivialAtom**(*AtomStorage*&)

MQ_NODISCARD bool is_applicable (const instruction_t &inst) const noexcept

Test whether this atom is applicable to a particular instruction.

Parameters

inst – An instruction

Returns

True if the atom can be applied, false otherwise

void **apply**(*circuit_t* &circuit, const operator_t &op, const qubits_t &qubits, const cbits_t &cbits) noexcept

Apply the atom (ie. the decomposition it represents) to a quantum circuit.

Note: Currently the `cbits` parameter is not used at all! It is here to make the API futureproof.

Parameters

- **circuit** – A quantum circuit to apply the decomposition atom to
- **op** – A quantum operation to decompose
- **qubits** – A list of qubits to apply the decomposition atom
- **cbits** – A list of classical bit the decomposition applies to

Public Static Functions

static inline MQ_NODISCARD constexpr std::string_view name () noexcept

Return the name of this decomposition atom.

static inline MQ_NODISCARD constexpr auto num_targets () noexcept

Return the number of target qubits this decomposition atom is constrained on.

static inline MQ_NODISCARD constexpr auto num_controls () noexcept

Return the number of control qubits this decomposition atom is constrained on.

static inline MQ_NODISCARD constexpr auto num_params () noexcept

Return the number of parameters of this decomposition atom.

static inline MQ_NODISCARD auto **create**(*AtomStorage* &storage) noexcept

Helper function to create an instance of this atom.

Parameters

storage – Atom storage within which this decomposition will live in

namespace **atoms**

Typedefs

using **Control** = typename traits::atom_control_type<op_t, num_controls>::control_type

using **C** = *Control*<op_t, num_controls>

namespace **details**

Functions

template<typename **T**, typename **U**, typename **V**>
constexpr auto **kind_lookup**(const *T* &lhs, const std::pair<*U*, *V*> &rhs)

template<typename **atom_t**>
void **apply_gate**(*atom_t* &atom, *circuit_t* &circuit, const *instruction_t* &inst)

template<std::size_t **idx**, typename **elem_t**, typename **tuple_t**>
constexpr auto **index_in_tuple_fn**()

Variables

template<typename **elem_t**, typename **tuple_t**>

constexpr auto **index_in_tuple** = *index_in_tuple_fn*<0, *elem_t*, *tuple_t*>()

struct **atom_less**

#include <atom_storage.hpp>

Public Types

using **is_transparent** = void

Public Functions

template<typename **T**, typename **U**, typename =
std::enable_if_t<!*is_pair*<std::remove_cvref_t<*T*>>::value &&
!*is_pair*<std::remove_cvref_t<*U*>>::value>>
inline constexpr auto **operator()**(*T* &&lhs, *U* &&rhs) const

template<typename **T**, typename **U**, typename **V**>
inline constexpr auto **operator()**(const std::pair<*T*, *V*> &lhs, const std::pair<*U*, *V*> &rhs) const

template<typename **T**>

struct **is_pair** : public false_type

#include <atom_storage.hpp>

template<typename T, typename U> pair< T, U > > : public true_type

#include <atom_storage.hpp>

struct **rules_less**

#include <gate_decomposer.hpp>

Public Types

using **is_transparent** = void

Public Functions

template<typename **T**, typename **U**>
inline constexpr auto **operator()**(*T* &&lhs, *U* &&rhs) const

namespace **impl**

Functions

template<typename **CircuitType**>
void **decompose_time_evolution_individual_terms**(*CircuitType* &result, const *instruction_t* &inst)

namespace **rules**

Variables

static constexpr auto **PI_VAL** = 3.141592653589793

static constexpr auto **PI_VAL_2** = *PI_VAL* / 2.

static constexpr auto **PI_VAL_4** = *PI_VAL* / 4.

class **CNOT2CZ** : public mindquantum::*decompositions*::*GateDecompositionRule*<*CNOT2CZ*, std::tuple<*ops*::X>, SINGLE_TGT_SINGLE_CTRL, *ops*::H, *atoms*::*C*<*ops*::Z>>

#include <cnot2cz.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const operator_t&, const qubits_t &qubits, const cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

class **CNOT2Rxx** : public mindquantum::*decompositions*::*GateDecompositionRule*<*CNOT2Rxx*, std::tuple<*ops*::X>, SINGLE_TGT_SINGLE_CTRL, *ops*::*parametric*::*Rx*, *ops*::*parametric*::*Ry*, *ops*::*parametric*::*Ph*, *atoms*::*C*<*ops*::*parametric*::*Rxx*>>

#include <cnot2rxx.hpp>

Public Functions

inline explicit **CNOT2Rxx**(*AtomStorage* &storage)

inline void **apply_positive_decomp**(*circuit_t* &circuit, const qubits_t &qubits)

inline void **apply_negative_decomp**(*circuit_t* &circuit, const qubits_t &qubits)

inline void **apply_impl**(*circuit_t* &circuit, const operator_t&, const qubits_t &qubits, const cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

class **CNu2ToffoliAndCu** : public mindquantum::*decompositions*::*NonGateDecompositionRule*<*CNu2ToffoliAndCu*, *atoms*::*C*<*ops*::X, 2>>

#include <cnu2toffoliandcu.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const *instruction_t* &inst, const qubits_t &qubits)

Public Static Functions

static inline constexpr auto **name**() noexcept

static inline MQ_NODISCARD bool is_applicable (const decompositions::instruction_t &inst)

class **CRZ2CXAndRz** : public mindquantum::*decompositions*::*NonGateDecompositionRule*<*CRZ2CXAndRz*, *ops*::*parametric*::*Rz*, *ops*::X>

#include <crz2cxandrz.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const *instruction_t* &inst)

Public Static Functions

static inline constexpr auto **name**() noexcept

static inline MQ_NODISCARD bool is_applicable (const instruction_t &inst)

class **Entangle2HAndCNOT** : public mindquantum::*decompositions*::*GateDecompositionRule*<*Entangle2HAndCNOT*, std::tuple<*ops*::*Entangle*>, ANY_TGT_NO_CTRL, *ops*::H, *atoms*::*C*<*ops*::X>>

#include <entangle.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const *decompositions*::operator_t&, const *decompositions*::qubits_t &qubits, const *decompositions*::cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

class **H2Rx** : public mindquantum::*decompositions*::*GateDecompositionRule*<*H2Rx*, std::tuple<*ops*::H>, SINGLE_TGT_NO_CTRL, *ops*::*parametric*::*Rx*, *ops*::*parametric*::*Ph*, *ops*::*parametric*::*Ry*>

#include <h2rx.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const operator_t&, const qubits_t &qubits, const cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

class **Ph2R** : public mindquantum::*decompositions*::*GateDecompositionRule*<*Ph2R*, std::tuple<*ops*::*Ph*, *ops*::*parametric*::*Ph*>, SINGLE_TGT_PARAM_SINGLE_CTRL, *ops*::*parametric*::*P*>

#include <ph2r.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const operator_t &op, const qubits_t &qubits, const cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

class **QFT2CrAndHadamard** : public mindquantum::*decompositions*::*GateDecompositionRule*<*QFT2CrAndHadamard*, std::tuple<*ops*::X>, ANY_TGT_NO_CTRL, *ops*::H, *atoms*::*C*<*ops*::P>>

#include <qft2crandhadamard.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const operator_t&, const qubits_t &qubits, const cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

class **R2RzAndPh** : public mindquantum::*decompositions*::*GateDecompositionRule*<*R2RzAndPh*, std::tuple<*ops*::P, *ops*::*parametric*::*P*>, SINGLE_TGT_PARAM_ANY_CTRL, *ops*::*parametric*::*Rz*, *ops*::*parametric*::*Ph*>

#include <r2rzandph.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const operator_t &op, const qubits_t &qubits, const cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

class **RemovePhNoCtrl** : public mindquantum::*decompositions*::*NonGateDecompositionRule*<*RemovePhNoCtrl*>

#include <no_control_ph.hpp>

Public Functions

inline void **apply_impl**(*circuit_t*&, const *instruction_t*&)

Public Static Functions

static inline constexpr auto **name**() noexcept

static inline MQ_NODISCARD bool is_applicable (const instruction_t &inst)

class **Rx2Rz** : public mindquantum::*decompositions*::*GateDecompositionRule*<*Rx2Rz*, std::tuple<*ops*::Rx, *ops*::*parametric*::*Rx*>, SINGLE_TGT_PARAM_ANY_CTRL, *ops*::H, *ops*::*parametric*::*Rz*>

#include <rx2rz.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const operator_t &op, const qubits_t &qubits, const cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

class **Ry2Rz** : public mindquantum::*decompositions*::*GateDecompositionRule*<*Ry2Rz*, std::tuple<*ops*::Ry, *ops*::*parametric*::*Ry*>, SINGLE_TGT_PARAM_ANY_CTRL, *ops*::*parametric*::*Rx*, *ops*::*parametric*::*Rz*>

#include <ry2rz.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const operator_t &op, const qubits_t &qubits, const cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

class **Rz2RxAndRy** : public mindquantum::*decompositions*::*GateDecompositionRule*<*Rz2RxAndRy*, std::tuple<*ops*::Rz, *ops*::*parametric*::*Rz*>, SINGLE_TGT_PARAM_ANY_CTRL, *ops*::*parametric*::*Rx*, *ops*::*parametric*::*Ry*>

#include <rz2rxandry.hpp>

Public Functions

inline explicit **Rz2RxAndRy**(*AtomStorage* &storage)

inline void **apply_positive_decomp**(*circuit_t* &circuit, const qubits_t &qubits, const gate_param_t &param)

inline void **apply_negative_decomp**(*circuit_t* &circuit, const qubits_t &qubits, const gate_param_t &param)

inline void **apply_impl**(*circuit_t* &circuit, const operator_t &op, const qubits_t &qubits, const cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

class **SqrtSwap2CNOTAndSqrtX** : public mindquantum::*decompositions*::*GateDecompositionRule*<*SqrtSwap2CNOTAndSqrtX*, std::tuple<*ops*::*SqrtSwap*>, DUAL_TGT_ANY_CTRL, *ops*::Sx, *atoms*::*C*<*ops*::X>>

#include <sqrtswap2cnotandsqrtx.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const operator_t&, const qubits_t &qubits, const cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

class **Swap2CNOT** : public mindquantum::*decompositions*::*GateDecompositionRule*<*Swap2CNOT*, std::tuple<*ops*::Swap>, DUAL_TGT_ANY_CTRL, *atoms*::*C*<*ops*::X>>

#include <swap2cnot.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const operator_t&, const qubits_t &qubits, const cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

class **Toffoli2CNOTAndT** : public mindquantum::*decompositions*::*GateDecompositionRule*<*Toffoli2CNOTAndT*, std::tuple<*ops*::X>, SINGLE_TGT_DOUBLE_CTRL, *atoms*::*C*<*ops*::X>, *ops*::H, *ops*::T, *ops*::Tdg>

#include <toffoli2cnotandtgate.hpp>

Public Functions

inline void **apply_impl**(*circuit_t* &circuit, const operator_t&, const qubits_t &qubits, const cbits_t&)

Public Static Functions

static inline constexpr auto **name**() noexcept

namespace **tparam**

Namespace containing some helper constants to use when defining *DecompositionRule* classes.

Variables

static constexpr auto **default_t** = *any_tgt*::*any_ctrl*

namespace **any_tgt**

Constants for decomposition rules without constraints on the number of target qubits.

Variables

static constexpr auto **any_ctrl** = *DecompositionRuleParam*{0, *any_control*, 0}

static constexpr auto **no_ctrl** = *DecompositionRuleParam*{0, 0, 0}

static constexpr auto **single_ctrl** = *DecompositionRuleParam*{0, 1, 0}

static constexpr auto **double_ctrl** = *DecompositionRuleParam*{0, 2, 0}

namespace **dual_tgt**

Constants for decomposition rules for single target qubit gates.

Variables

static constexpr auto **any_ctrl** = *DecompositionRuleParam*{2, *any_control*, 0}

static constexpr auto **no_ctrl** = *DecompositionRuleParam*{2, 0, 0}

static constexpr auto **single_ctrl** = *DecompositionRuleParam*{2, 1, 0}

static constexpr auto **double_ctrl** = *DecompositionRuleParam*{2, 2, 0}

namespace **single_tgt**

Constants for decomposition rules for single target qubit gates.

Variables

static constexpr auto **any_ctrl** = *DecompositionRuleParam*{1, *any_control*, 0}

static constexpr auto **no_ctrl** = *DecompositionRuleParam*{1, 0, 0}

static constexpr auto **single_ctrl** = *DecompositionRuleParam*{1, 1, 0}

static constexpr auto **double_ctrl** = *DecompositionRuleParam*{1, 2, 0}

namespace **single_tgt_param**

Constants for decomposition rules for single target qubit with parameteric gates with 1 parameter.

Variables

static constexpr auto **any_ctrl** = *DecompositionRuleParam*{1, *any_control*, 1}

static constexpr auto **no_ctrl** = *DecompositionRuleParam*{1, 0, 1}

static constexpr auto **single_ctrl** = *DecompositionRuleParam*{1, 1, 1}

static constexpr auto **double_ctrl** = *DecompositionRuleParam*{1, 2, 1}

C++ operators

namespace **ops**

class **Allocate**

#include <allocate.hpp>

Public Static Functions

static inline constexpr std::string_view **kind**()

class **Command**

#include <cpp_command.hpp> Subclassed by mindquantum::python::Command

Public Types

using **gate_t** = td::Operator

Public Functions

inline const auto &**get_qubits**() const

inline const auto &**get_control_qubits**() const

inline const *gate_t* &**get_gate**() const

class **DaggerOperation**

#include <dagger.hpp>

Public Functions

template<typename **OpT**>
inline explicit **DaggerOperation**(*OpT* &&op)

inline uint32_t **num_targets**() const

inline td::Operator **adjoint**() const

inline std::optional<td::UMatrix> **matrix**() const

inline bool **operator==**(const *DaggerOperation* &other) const

Public Static Functions

static inline constexpr std::string_view **kind**()

class **Deallocate**

#include <deallocate.hpp>

Public Static Functions

static inline constexpr std::string_view **kind**()

class **Entangle**

#include <entangle.hpp>

Public Types

using **non_const_num_targets** = void

Public Functions

inline explicit **Entangle**(uint32_t num_targets)

inline td::Operator **adjoint**() const

inline uint32_t **num_targets**() const

inline bool **operator==**(const *Entangle* &other) const

Public Static Functions

static inline constexpr std::string_view **kind**()

class **Invalid**

#include <invalid.hpp>

Public Static Functions

static inline constexpr std::string_view **kind**()

class **Measure**

#include <measure.hpp>

Public Static Functions

static inline constexpr std::string_view **kind**()

class **Ph**

#include <ph.hpp>

Public Functions

inline explicit **Ph**(double angle)

inline *Ph* **adjoint**() const

inline constexpr bool **is_symmetric**() const

inline UMatrix2 const **matrix**() const

inline bool **operator==**(*Ph* const &other) const

inline const auto &**angle**() const

Public Static Functions

static inline constexpr std::string_view **kind**()

Public Static Attributes

static constexpr auto **num_params** = 1UL

class **QFT**

#include <qft.hpp>

Public Types

using **non_const_num_targets** = void

Public Functions

inline explicit **QFT**(uint32_t num_targets)

inline td::Operator **adjoint**() const

inline uint32_t **num_targets**() const

inline bool **operator==**(const *QFT* &other) const

Public Static Functions

static inline constexpr std::string_view **kind**()

class **QubitOperator**

#include <qubit_operator.hpp>

Public Types

using **non_const_num_targets** = void

using **ComplexTerm** = std::pair<std::vector<std::pair<uint32_t, char>>, std::complex<double>>

using **ComplexTermsDict** = std::map<std::vector<std::pair<uint32_t, char>>, std::complex<double>>

Public Functions

inline explicit **QubitOperator**(uint32_t num_targets, *ComplexTermsDict* terms)

inline td::Operator **adjoint**() const

inline uint32_t **num_targets**() const

inline bool **operator==**(const *QubitOperator* &other) const

inline const *ComplexTermsDict* &**get_terms**() const

inline *QubitOperator* **Subtract**(*QubitOperator* const &other) const

inline *QubitOperator* **operator-**(*QubitOperator* const &other) const

inline *QubitOperator* **Multiply**(*QubitOperator* const &other) const

inline *QubitOperator* **operator***(*QubitOperator* const &other) const

inline bool **is_close**(const *QubitOperator* &other, double rel_tol = 1e-12, double abs_tol = 1e-12) const

inline bool **is_identity**(double abs_tol = 1e-12) const

Public Static Functions

static inline constexpr std::string_view **kind**()

class **SqrtSwap**

#include <sqrtswap.hpp>

Public Functions

inline uint32_t **num_targets**() const

inline td::Operator **adjoint**() const

Public Static Functions

static inline constexpr std::string_view **kind**()

static inline td::UMatrix4 const **matrix**()

class **TimeEvolution**

#include <time_evolution.hpp>

Public Types

using **non_const_num_targets** = void

Public Functions

inline **TimeEvolution**(uint32_t num_targets, *QubitOperator* hamiltonian, double time)

Constructor.

Overload required in some cases for metaprogramming with operators.

inline **TimeEvolution**(*QubitOperator* hamiltonian, double time)

Constructor.

inline MQ_NODISCARD TimeEvolution adjoint () const

inline MQ_NODISCARD uint32_t num_targets () const

inline bool **operator==**(const *TimeEvolution* &other) const

inline MQ_NODISCARD const QubitOperator & get_hamiltonian () const

inline MQ_NODISCARD auto **get_time**() const

inline MQ_NODISCARD auto **param**() const

Public Static Functions

static inline constexpr std::string_view **kind**()

namespace **parametric**

Typedefs

using **subs_map_t** = SymEngine::map_basic_basic

using **basic_t** = SymEngine::RCP<const SymEngine::Basic>

using **double_list_t** = std::vector<double>

using **param_list_t** = SymEngine::vec_basic

using **gate_param_t** = std::variant<std::monostate, double, *double_list_t*, *param_list_t*>

Functions

template<typename op_t> void register_gate_type() MQ_REQUIRES((concepts[[nodiscard]] gate_para

Register a new gate class.

Note: If the gate is neither a parametric gate or a gate with an angle() method, this method is no-op.Get the parameters of an operation

---

Template Parameters
operator_t – Type of the gate to register

Parameters
optor – A quantum operation

template<typename **op_t**, typename ...**args_t**>
MQ_NODISCARD auto **generate_subs**(*args_t*&&... args)

Generate a substitution dictionary from a single double.

Pre
`sizeof(args_t)` == operator_t::num_params()

template<typename **op_t**>
MQ_NODISCARD auto **generate_subs**(const *double_list_t* &params)

Generate a substitution dictionary from an array of double.

See also:

generate_subs_(const std::vector<T>& param) const;

template<typename **op_t**>

MQ_NODISCARD auto **generate_subs**(const *param_list_t* &params)

Generate a substitution dictionary from an array of expressions.

See also:

generate_subs_(const std::vector<T>& param) const;

template<typename **T**>
auto **to_symengine**(*T* &&t)

template<typename **derived_t**, typename **non_param_t**, std::size_t **mod_pi**>

class **AngleParametricBase** : public mindquantum::*ops*::*parametric*::*ParametricBase*<*derived_t*, *non_param_t*, *real*::*theta*>

#include <angle_base.hpp>

Public Types

using **operator_t** = tweedledum::Operator

using **parent_t** = *ParametricBase*<*derived_t*, *non_param_t*, *real*::*theta*>

using **base_t** = *AngleParametricBase*

using **self_t** = *AngleParametricBase*<*derived_t*, *non_param_t*, *mod_pi*>

using **non_param_type** = *non_param_t*

using **subs_map_t** = SymEngine::map_basic_basic

Public Functions

inline MQ_NODISCARD bool operator== (const AngleParametricBase &other) const noexcept

Test whether another operation is the same as this instance.

inline MQ_NODISCARD auto **adjoint**() const noexcept

Get the adjoint of an *AngleParametricBase* gate instance.

inline MQ_NODISCARD non_param_type eval_full () const

Fully evaluate this parametric gate.

Attempt to fully evaluate this parametric gate (ie. evaluate all parameter numerically). The constructor of the non-parametric gate type is called by passing each of the numerically evaluated parameter in the order that is defined when passing the type as the template parameters to this base class.

Throws

SymEngine::SymEngineException – if the expression cannot be fully evaluated numerically

Returns
An instance of a non-parametric gate (non_param_type)

`inline MQ_NODISCARD non_param_type eval_full (const subs_map_t &subs_map) const`

Fully evaluate this parametric gate.

Attempt to fully evaluate this parametric gate (ie. evaluate all parameter numerically). The constructor of the non-parametric gate type is called by passing each of the numerically evaluated parameter in the order that is defined when passing the type as the template parameters to this base class.

This overload accepts a dictionary of substitutions to perform on the parameters.

Parameters
subs_map – Dictionary containing all the substitution to perform

Throws
SymEngine::SymEngineException – if the expression cannot be fully evaluated numerically

Returns
An instance of a non-parametric gate (non_param_type)

Public Static Functions

template<typename **evaluated_param_t**>
static inline MQ_NODISCARD auto **to_param_type**(const *self_t*&, *evaluated_param_t* &&evaluated_param)

Evaluation helper function.

template<typename **evaluated_param_t**>
static inline MQ_NODISCARD auto **to_non_param_type**(const *self_t*&, *evaluated_param_t* &&evaluated_param)

Evaluation helper function.

class **P** : public mindquantum::*ops*::*parametric*::*AngleParametricBase*<*P*, tweedledum::Op::P, 2>

#include <angle_gates.hpp>

Public Static Functions

static inline constexpr std::string_view **kind**()

Public Static Attributes

static constexpr auto **num_targets** = traits::num_targets<tweedledum::Op::P>

template<typename **derived_t**, typename **non_param_t**, typename ...**params_t**>

class **ParametricBase**

#include <gate_base.hpp>

Public Types

using **base_t** = *ParametricBase*

using **self_t** = *ParametricBase*

using **operator_t** = tweedledum::Operator

using **is_parametric** = void

using **non_param_type** = *non_param_t*

using **params_type** = std::tuple<*params_t*...>

using **param_array_t** = std::array<*basic_t*, *num_params*>

using **subs_map_t** = SymEngine::map_basic_basic

Public Functions

template<typename ...**Ts**, typename **T** = *self_t*, typename = std::enable_if_t<*T*::has_const_num_targets && !traits::is_param_base<*Ts*...>::value>>
inline constexpr **ParametricBase**(*Ts*&&... args)

Constructor from a list of either C++ numeric types or symbolic expressions.

Note: This constructor expects exactly `num_params` arguments

Parameters
expr – List of SymEngine expressions

template<typename ...**Ts**, typename **T** = *self_t*, typename = std::enable_if_t<!*T*::has_const_num_targets>>
inline constexpr **ParametricBase**(uint32_t num_targets, *Ts*&&... args)

Constructor from a list of either C++ numeric types or symbolic expressions.

Note: This constructor expects exactly `num_params` arguments

Parameters
- **num_targets** – Number of target qubits
- **expr** – List of SymEngine expressions

template<typename **T** = *self_t*, typename = std::enable_if_t<*T*::has_const_num_targets>>
inline constexpr **ParametricBase**(*param_array_t* &&params)

Constructor from an array of parameters.

template<typename **T** = *self_t*, typename = std::enable_if_t<!*T*::has_const_num_targets>>

inline constexpr **ParametricBase**(uint32_t num_targets, *param_array_t* &&params)

Constructor from an array of parameters.

ParametricBase() = delete

ParametricBase(const *ParametricBase*&) = default

ParametricBase(*ParametricBase*&&) noexcept = default

ParametricBase &**operator=**(const *ParametricBase*&) = default

ParametricBase &**operator=**(*ParametricBase*&&) noexcept = default

template<typename T = non_param_type, typename = std::enable_if_t<!
traits::has_const_num_targets_v<T>,
uint32_t>> inline MQ_NODISCARD constexpr auto num_targets () const noexcept

inline MQ_NODISCARD bool operator== (const ParametricBase &other) const noexcept

Test whether another operation is the same as this instance.

inline MQ_NODISCARD bool operator!
= (const ParametricBase &other) const noexcept

Test whether another operation is the same as this instance.

inline MQ_NODISCARD bool is_symmetric () const noexcept

True if this operation has no particular ordering of qubits.

inline MQ_NODISCARD constexpr const auto & param (std::size_t idx) const

Parameter getter method.

Parameters

idx – Index of parameter

Returns

Parameter at index `idx`

inline MQ_NODISCARD auto **params**() const noexcept

Get all the parameters in an array.

Returns

SymEngine::vec_basic (`std::vector<...>`) containing all parameters

inline MQ_NODISCARD derived_t eval (const subs_map_t &subs_map) const

Evaluate the parameters of this parametric gate using some substitutions.

This function does not attempt to fully evaluate this parametric gate (ie. evaluate all parameter numerically)

See also:

non_param_type eval_full(const SymEngine::map_basic_basic& subs_map) const

Parameters

subs_map – Dictionary containing all the substitution to perform

Returns

An new instance of the parametric gate with evaluated parameters

inline MQ_NODISCARD auto **eval_full**() const

Fully evaluate this parametric gate.

Attempt to fully evaluate this parametric gate (ie. evaluate all parameter numerically). The constructor of the non-parametric gate type is called by passing each of the numerically evaluated parameter in the order that is defined when passing the type as the template parameters to this base class.

Throws

SymEngine::SymEngineException – if the expression cannot be fully evaluated numerically

Returns

An instance of a non-parametric gate (non_param_type)

inline MQ_NODISCARD auto **eval_full**(const *subs_map_t* &subs_map) const

Fully evaluate this parametric gate.

Attempt to fully evaluate this parametric gate (ie. evaluate all parameter numerically). The constructor of the non-parametric gate type is called by passing each of the numerically evaluated parameter in the order that is defined when passing the type as the template parameters to this base class.

This overload accepts a dictionary of substitutions to perform on the parameters.

Parameters

subs_map – Dictionary containing all the substitution to perform

Throws

SymEngine::SymEngineException – if the expression cannot be fully evaluated numerically

Returns

An instance of a non-parametric gate (non_param_type)

inline MQ_NODISCARD operator_t eval_smart () const

Evaluate the parameters of this parametric gate using some substitutions.

This function will attempt to fully evaluate this parametric gate if possible. This will happen only if, after substitutions, all the parameters have a numeric representation.

See also:

non_param_type eval_full(const SymEngine::map_basic_basic& subs_map) const

Parameters

subs_map – Dictionary containing all the substitution to perform

Returns

An new instance of the parametric gate with evaluated parameters

inline MQ_NODISCARD operator_t eval_smart (const subs_map_t &subs_map) const

Evaluate the parameters of this parametric gate using some substitutions.

This function will attempt to fully evaluate this parametric gate if possible. This will happen only if, after substitutions, all the parameters have a numeric representation.

See also:

non_param_type eval_full(const SymEngine::map_basic_basic& subs_map) const

Parameters
subs_map – Dictionary containing all the substitution to perform
Returns
An new instance of the parametric gate with evaluated parameters

Public Static Functions

```
static inline MQ_NODISCARD constexpr derived_t create_op ()
```

Create a default instance of `derived_t` type.

Overload only available if `non_param_type` has compile-time constant number of qubits

```
static inline MQ_NODISCARD constexpr derived_t create_op (uint32_t num_targets)
```

Create a default instance of `derived_t` type.

Overload only available if `non_param_type` does not have compile-time constant number of qubits

```
template<typename...
funcs_t> static inline MQ_NODISCARD constexpr derived_t create_op (funcs_t &&.
.. transforms)
```

Create an instance of `derived_t` type with some transformation on the parameters.

Overload only available if `non_param_type` has compile-time constant number of qubits

```
template<typename...
funcs_t> static inline MQ_NODISCARD constexpr derived_t create_op (uint32_t num_targets,
funcs_t &&... transforms)
```

Create an instance of `derived_t` type with some transformation on the parameters.

Overload only available if `non_param_type` does not have compile-time constant number of qubits

```
template<typename T = non_param_type,
typename = std::enable_if_t<traits::has_const_num_targets_v<T>,
uint32_t>> static inline MQ_NODISCARD constexpr auto num_targets_static () noexcept
```

```
static inline MQ_NODISCARD constexpr const auto & param_name (std::size_t idx)
```

Get the name of the parameter at some index.
Parameters
idx – Index of parameter

Public Static Attributes

static constexpr auto **num_params** = sizeof...(*params_t*)

static constexpr auto **has_const_num_targets** =
traits::has_const_num_targets_v<*non_param_type*>

class **Ph** : public mindquantum::*ops*::*parametric*::*AngleParametricBase*<*Ph*, *ops*::*Ph*, 2>

#include <angle_gates.hpp>

Public Static Functions

static inline constexpr std::string_view **kind**()

Public Static Attributes

static constexpr auto **num_targets** = traits::num_targets<*ops*::*Ph*>

class **Rx** : public mindquantum::*ops*::*parametric*::*AngleParametricBase*<*Rx*, tweedledum::Op::Rx, 4>

#include <angle_gates.hpp>

Public Static Functions

static inline constexpr std::string_view **kind**()

Public Static Attributes

static constexpr auto **num_targets** = traits::num_targets<tweedledum::Op::Rx>

class **Rxx** : public mindquantum::*ops*::*parametric*::*AngleParametricBase*<*Rxx*, tweedledum::Op::Rxx, 4>

#include <angle_gates.hpp>

Public Static Functions

static inline constexpr std::string_view **kind**()

Public Static Attributes

static constexpr auto **num_targets** = traits::num_targets<tweedledum::Op::Rxx>

class **Ry** : public mindquantum::*ops*::*parametric*::*AngleParametricBase*<*Ry*, tweedledum::Op::Ry, 4>
#include <angle_gates.hpp>

Public Static Functions

static inline constexpr std::string_view **kind**()

Public Static Attributes

static constexpr auto **num_targets** = traits::num_targets<tweedledum::Op::Ry>

class **Ryy** : public mindquantum::*ops*::*parametric*::*AngleParametricBase*<*Ryy*, tweedledum::Op::Ryy, 4>
#include <angle_gates.hpp>

Public Static Functions

static inline constexpr std::string_view **kind**()

Public Static Attributes

static constexpr auto **num_targets** = traits::num_targets<tweedledum::Op::Ryy>

class **Rz** : public mindquantum::*ops*::*parametric*::*AngleParametricBase*<*Rz*, tweedledum::Op::Rz, 4>
#include <angle_gates.hpp>

Public Static Functions

static inline constexpr std::string_view **kind**()

Public Static Attributes

static constexpr auto **num_targets** = traits::num_targets<tweedledum::Op::Rz>

class **Rzz** : public mindquantum::*ops*::*parametric*::*AngleParametricBase*<*Rzz*, tweedledum::Op::Rzz, 4>
#include <angle_gates.hpp>

Public Static Functions

static inline constexpr std::string_view **kind**()

Public Static Attributes

static constexpr auto **num_targets** = traits::num_targets<tweedledum::Op::Rzz>

class **TimeEvolution** : public mindquantum::*ops*::*parametric*::*ParametricBase*<*TimeEvolution*, *ops*::*TimeEvolution*, *real*::*alpha*>

#include <time_evolution.hpp>

Public Types

using **operator_t** = tweedledum::Operator

using **parent_t** = *ParametricBase*<*TimeEvolution*, *ops*::*TimeEvolution*, *real*::*alpha*>

using **self_t** = *TimeEvolution*

using **non_const_num_targets** = void

using **non_param_type** = non_param_t

using **subs_map_t** = SymEngine::map_basic_basic

Public Functions

template<typename **param_t**>
inline **TimeEvolution**(uint32_t num_targets, *QubitOperator* hamiltonian, *param_t* &&param)

Constructor.

Overload required in some cases for metaprogramming with operators.

template<typename **param_t**>
inline **TimeEvolution**(*QubitOperator* hamiltonian, *param_t* &&param)

Constructor.

inline MQ_NODISCARD auto **adjoint**() const noexcept

Get the adjoint of an *TimeEvolution* gate instance.

inline bool **operator==**(const *self_t* &other) const

inline MQ_NODISCARD const QubitOperator & get_hamiltonian () const

inline MQ_NODISCARD const auto & get_time () const

Public Static Functions

static inline constexpr std::string_view **kind**()

template<typename **evaluated_param_t**>
static inline MQ_NODISCARD auto **to_param_type**(const *self_t* &self, *evaluated_param_t* &&evaluated_param)

template<typename **evaluated_param_t**>
static inline MQ_NODISCARD auto **to_non_param_type**(const *self_t* &self, *evaluated_param_t* &&evaluated_param)

namespace **complex**

struct **alpha**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "alpha"

struct **beta**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "beta"

struct **chi**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "chi"

struct **delta**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "delta"

struct **epsilon**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "epsilon"

struct **eta**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "eta"

struct **gamma**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "gamma"

struct **iota**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "iota"

struct **kappa**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "kappa"

struct **lambda**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "lambda"

struct **mu**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "mu"

struct **nu**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "nu"

struct **omega**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "omega"

struct **omicron**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "omicron"

struct **phi**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "phi"

struct **pi**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "pi"

struct **rho**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "rho"

struct **sigma**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "sigma"

struct **tau**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "tau"

struct **theta**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "theta"

struct **upsilon**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "upsilon"

struct **xi**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "xi"

struct **zeta**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*complex_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "zeta"

namespace **details**

Functions

template<typename **op_t**, typename **T**>
MQ_NODISCARD auto **generate_subs**(const std::vector<*T*> &params)

Generate a substitution dictionary from an array of elements.

Pre

`size(param)` == op_t::num_params()

template<typename **op_t**, std::size_t... **indices**, typename ...**args_t**>
MQ_NODISCARD auto **create_subs_from_params**(std::index_sequence<*indices*...>, *args_t*&&... args)

void **register_gate**(std::string_view kind, double_func_t angle_func)

void **register_gate**(std::string_view kind, vec_double_func_t vec_double_func)

void **register_gate**(std::string_view kind, params_func_t params_func)

struct **complex_tag_t**

#include <param_names.hpp> Defines a complex parameter.

Public Types

using **type** = std::complex<double>

Public Static Functions

static inline auto **eval**(const *basic_t* &expr)

struct **real_tag_t**

#include <param_names.hpp> Defines a real parameter.

Public Types

using **type** = double

Public Static Functions

static inline auto **eval**(const *basic_t* &expr)

namespace **real**

struct **alpha**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "alpha"

struct **beta**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "beta"

struct **chi**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "chi"

struct **delta**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "delta"

struct **epsilon**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "epsilon"

struct **eta**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "eta"

struct **gamma**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "gamma"

struct **iota**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "iota"

struct **kappa**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "kappa"

struct **lambda**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "lambda"

struct **mu**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "mu"

struct **nu**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "nu"

struct **omega**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "omega"

struct **omicron**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "omicron"

struct **phi**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "phi"

struct **pi**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "pi"

struct **rho**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "rho"

struct **sigma**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "sigma"

struct **tau**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "tau"

struct **theta**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "theta"

struct **upsilon**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "upsilon"

struct **xi**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "xi"

struct **zeta**

#include <param_names.hpp>

Public Types

using **param_type** = *details*::*real_tag_t*

Public Static Attributes

static constexpr std::string_view **name** = "zeta"

CHAPTER

SEVEN

HOW TO CITE

When using MindQuantum for research, please cite:

```
@misc{mq_2021,
    author      = {MindQuantum Developer},
    title       = {MindQuantum, version 0.5.0},
    month       = {March},
    year        = {2021},
    url         = {https://gitee.com/mindspore/mindquantum}
}
```

CHAPTER

EIGHT

MINDQUANTUM LICENSE

Apache License 2.0

INDEX

M