qoqo Documentation

Release 0.0

HQS Quantum Simulations GmbH

Apr 28, 2023

CONTENTS:

qoqo the HQS Quantum Simulation software package to represent quantum circuits.

Introduction to qoqo with a jupyter notebook.

CHAPTER

ONE

QOQO DOCUMENTATION

qoqo	Qoqo

1.1 qoqo

Qoqo

Quantum Operation Quantum Operation
Yes we use reduplication.

qoqo is the HQS python package to represent quantum circuits.

Circuit	Circuit of Operations.
operations	Operations are the atomic instructions in any quantum program that can be represented by qoqo.
measurements	Measurements
QuantumProgram	Represents a quantum program evaluating measurements based on a one or more free float parameters.

1.1.1 qoqo.Circuit

class qoqo.**Circuit**

Circuit of Operations.

A quantum program is represented as a linear sequence of Operations.

__init__()

Methods

__init__()	

add(op)	Add an Operation to Circuit.
count_occurences(operations)	Count the number of occurences of a set of operation tags in the circuit.
definitions()	Return a list of definitions in the Circuit.
filter_by_tag(tag)	Return a list of operations with given tag.
from_bincode(input)	Convert the bincode representation of the Circuit to a Circuit using the [bincode] crate.
from_json(json_string)	Convert the json representation of a Circuit to a Circuit.
get(index)	Return a copy of the Operation at a certain index of the Circuit.
get_operation_types()	Return a list of the hqslang names of all operations occuring in the circuit.
get_slice()	Return the copy of a slice of the Circuit.
operations()	Return a list of all operations in the Circuit.
overrotate()	Return clone of the circuit with all overrotation Pragmas applied.
remap_qubits(mapping)	Remap qubits in operations in clone of Circuit.
substitute_parameters(substitution_parameters)	Substitute the symbolic parameters in a clone of the Circuit according to the substitution_parameters input.
to_bincode()	Return the bincode representation of the Circuit using the [bincode] crate.
to_json()	Return the json representation of the Circuit.

add(*op*)

Add an Operation to Circuit.

Parameters
op (*Operation*) – The Operation to add to the Circuit.

count_occurences(*operations*)

Count the number of occurences of a set of operation tags in the circuit.

Parameters
operations (*list[str]*) – List of operation tags that should be counted.

Returns
The number of occurences of these operation tags.

Return type
int

definitions()

Return a list of definitions in the Circuit.

Definitions need to be unique.

Returns
A vector of the definitions in the Circuit.

Return type
list[Operation]

`filter_by_tag`(*tag*)

Return a list of operations with given tag.

Parameters
tag (*str*) – tag by which to filter operations.

Returns
A vector of the operations with the specified tag in the Circuit.

Return type
list[Operation]

`static from_bincode`(*input*)

Convert the bincode representation of the Circuit to a Circuit using the [bincode] crate.

Parameters
input (*ByteArray*) – The serialized Circuit (in [bincode] form).

Returns
The deserialized Circuit.

Return type
Circuit

Raises

- **TypeError** – Input cannot be converted to byte array.
- **ValueError** – Input cannot be deserialized to Circuit.

`static from_json`(*json_string*)

Convert the json representation of a Circuit to a Circuit.

Parameters
input (*str*) – The serialized Circuit in json form.

Returns
The deserialized Circuit.

Return type
Circuit

Raises
ValueError – Input cannot be deserialized to Circuit.

`get`(*index*)

Return a copy of the Operation at a certain index of the Circuit.

Parameters
index (*int*) – The index of the Operation to get in the Circuit.

Returns
The operation at the given index (if it exists).

Return type
Operation

Raises
IndexError – Index out of range.

`get_operation_types()`

Return a list of the hqslang names of all operations occuring in the circuit.

Returns
The operation types in the Circuit.

Return type
set[str]

`get_slice()`

Return the copy of a slice of the Circuit.

Parameters

- **start** (*Optional[int]*) – The starting index of the slice (inclusive).
- **stop** (*Optional[int]*) – The stopping index of the slice (exclusive).

Returns
The slice of the operations in the Circuit with the specified indices.

Return type
Circuit

Raises

- **IndexError** – Stop index smaller than start index.
- **IndexError** – Stop index out of range.
- **IndexError** – Start index out of range.

`operations()`

Return a list of all operations in the Circuit.

Returns
A vector of the operations in the Circuit.

Return type
list[Operation]

`overrotate()`

Return clone of the circuit with all overrotation Pragmas applied.

Returns
Circuit with the overrotation applied

Return type
Circuit

Raises
RuntimeError – Error applying PragmaOverrotation in circuit.

Example:

```
>>> circuit = Circuit()
>>> circuit += PragmaOverrotation("RotateY", [1,], 20.0, 30.0)
>>> circuit += RotateX(0, 0.0)
>>> circuit += RotateY(0, 1.0)
>>> circuit += RotateY(1, 2.0)
>>> circuit += RotateY(1, 3.0)
>>> circuit_overrotated = circuit.overrotate()
```

(continues on next page)

(continued from previous page)

```
print(circuit)
print(circuit_overrotated)
```

remap_qubits(*mapping*)

Remap qubits in operations in clone of Circuit.

Parameters
mapping (*dict[int, int]*) – The dictionary containing the {qubit: qubit} mapping to use in the Circuit.

Returns
The Circuit with the qubits remapped.

Return type
self

Raises
RuntimeError – The qubit remapping failed.

substitute_parameters(*substitution_parameters*)

Substitute the symbolic parameters in a clone of the Circuit according to the substitution_parameters input.

Parameters
substitution_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the Circuit.

Returns
The Circuit with the parameters substituted.

Return type
self

Raises
RuntimeError – The parameter substitution failed.

to_bincode()

Return the bincode representation of the Circuit using the [bincode] crate.

Returns
The serialized Circuit (in [bincode] form).

Return type
ByteArray

Raises
ValueError – Cannot serialize Circuit to bytes.

to_json()

Return the json representation of the Circuit.

Returns
The serialized form of Circuit.

Return type
str

Raises
ValueError – Cannot serialize Circuit to json.

1.1.2 qoqo.operations

Operations are the atomic instructions in any quantum program that can be represented by qoqo.

Operations can be of various kinds: Definitions, GateOperations, PRAGMAs or measurement Operations.

Operations:

Definition operations define the classical registers and variables in the Circuit. GateOperations are single-, two- or multi-qubit gate operations that act on a set of qubits and can be executed on a quantum computing device. PRAGMAs are operations that can be used when running a simulation of a quantum computing program. Measurement Operations are operations that perform a measurement either on a quantum computing device (MeasuareQubit) or on a simulation of a quantum computing program (PRAGMA measurement operations).

SingleQubitGate	The general single qubit unitary gate.
RotateZ	The ZPower gate $e^{-i\frac{\theta}{2}\sigma^z}$.
RotateY	The YPower gate $e^{-i\frac{\theta}{2}\sigma^y}$.
RotateX	The XPower gate $e^{-i\frac{\theta}{2}\sigma^x}$.
RotateAroundSphericalAxis	Implements a rotation around an axis in the x-y plane in spherical coordinates.
PauliZ	The Pauli Z gate.
PauliY	The Pauli Y gate.
PauliX	The Pauli X gate.
SqrtPauliX	The square root of the XPower gate $e^{-i\frac{\pi}{4}\sigma^x}$.
InvSqrtPauliX	The inverse square root XPower gate $e^{i\frac{\pi}{2}\sigma^x}$.
Hadamard	The Hadamard gate.
TGate	The T gate.
SGate	The S gate.
DefinitionUsize	DefinitionUsize is the Definition for an Integer type register.
DefinitionBit	DefinitionBit is the Definition for a Bit type register.
DefinitionFloat	DefinitionFloat is the Definition for a Float type register.
DefinitionComplex	DefinitionComplex is the Definition for a Complex type register.
InputSymbolic	InputSymbolic is the Definition for a Float which will replace a certain symbolic parameter.
InputSymbolic	InputSymbolic is the Definition for a Float which will replace a certain symbolic parameter.
MeasureQubit	Measurement gate operation.
PragmaGetStateVector	This PRAGMA measurement operation returns the statevector of a quantum register.
PragmaGetDensityMatrix	This PRAGMA measurement operation returns the density matrix of a quantum register.
PragmaGetOccupationProbability	This PRAGMA measurement operation returns the vector of the occupation probabilities.
PragmaGetPauliProduct	This PRAGMA measurement operation returns a Pauli product expectation value.
PragmaRepeatedMeasurement	This PRAGMA measurement operation returns a measurement record for N repeated measurements.
PragmaSetNumberOfMeasurements	Wrap function automatically generates functions in these traits.
PragmaSetStateVector	This PRAGMA operation sets the statevector of a quantum register.

continues on next page

Table 1 – continued from previous page

PragmaSetDensityMatrix	This PRAGMA operation sets the density matrix of a quantum register.
PragmaRepeatGate	The repeated gate PRAGMA operation.
PragmaOverrotation	The statistical overrotation PRAGMA operation.
PragmaBoostNoise	This PRAGMA operation boosts noise and overrotations in the circuit.
PragmaStopParallelBlock	This PRAGMA operation signals the STOP of a parallel execution block.
PragmaGlobalPhase	The global phase PRAGMA operation.
PragmaSleep	This PRAGMA operation makes the quantum hardware wait a given amount of time.
PragmaActiveReset	This PRAGMA operation resets the chosen qubit to the zero state.
PragmaStopDecompositionBlock	This PRAGMA operation signals the STOP of a decomposition block.
PragmaDamping	The damping PRAGMA noise operation.
PragmaDepolarising	The depolarising PRAGMA noise operation.
PragmaDephasing	The dephasing PRAGMA noise operation.
PragmaRandomNoise	The random noise PRAGMA operation.
PragmaGeneralNoise	The general noise PRAGMA operation.
PragmaConditional	The conditional PRAGMA operation.
PragmaLoop	This PRAGMA measurement operation returns the statevector of a quantum register.
CNOT	The controlled NOT quantum operation.
SWAP	The controlled SWAP quantum operation.
FSwap	The controlled fermionic SWAP gate.
ISwap	The controlled ISwap quantum operation.
SqrtISwap	The controlled square root ISwap quantum operation.
InvSqrtISwap	The controlled inverse square root ISwap quantum operation.
XY	The controlled XY quantum operation
ControlledPhaseShift	The controlled-PhaseShift quantum operation.
ControlledPauliY	The controlled PauliY quantum operation
ControlledPauliZ	The controlled PauliZ quantum operation
ControlledRotateX	Implements the controlled RotateX operation.
ControlledRotateXY	Implements the controlled RotateXY operation.
ControlledControlledPauliZ	Implements the double-controlled PauliZ gate.
ControlledControlledPhaseShift	Implements the double-controlled PhaseShift gate.
Toffoli	Implements Toffoli gate.
MolmerSorensenXX	The fixed phase MolmerSorensen XX gate.
VariableMSXX	The variable-angle MolmerSorensen XX gate.
GivensRotation	The Givens rotation interaction gate in big endian notation: $e^{-\mathrm{i}\theta(X_cY_t-Y_cX_t)}$.
GivensRotationLittleEndian	The Givens rotation interaction gate in little endian notation: $e^{-\mathrm{i}\theta(X_cY_t-Y_cX_t)}$.
Qsim	The qubit simulation (Qsim) gate.
Fsim	The fermionic qubit simulation (Fsim) gate.
SpinInteraction	The generalized, anisotropic XYZ Heisenberg interaction between spins.
Bogoliubov	The Bogoliubov DeGennes interaction gate.
PMInteraction	The transversal interaction gate.

continues on next page

Table 1 – continued from previous page

ComplexPMInteraction	The complex hopping gate.
MultiQubitMS	The Molmer-Sorensen gate between multiple qubits.

qoqo.operations.SingleQubitGate

class qoqo.operations.**SingleQubitGate**

The general single qubit unitary gate.

$$U = \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Parameters

- **qubit** – The qubit that the unitary gate is applied to.
- **alpha_r** – The real part of the on-diagonal elements of the single-qubit unitary.
- **alpha_i** – The imaginary part of the on-diagonal elements of the single-qubit unitary.
- **beta_r** – The real part of the off-diagonal elements of the single-qubit unitary.
- **beta_i** – The imaginary part of the off-diagonal elements of the single-qubit unitary.
- **global_phase** – The global phase of the single-qubit unitary.

__init__()

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.RotateZ

class qoqo.operations.**RotateZ**

The ZPower gate $e^{-i\frac{\theta}{2}\sigma^z}$.

$$U = \begin{pmatrix} \cos(\frac{\theta}{2}) & 0 \\ 0 & \cos(\frac{\theta}{2}) \end{pmatrix} + \begin{pmatrix} -i\sin(\frac{\theta}{2}) & 0 \\ 0 & i\sin(\frac{\theta}{2}) \end{pmatrix}$$

Parameters

- **qubit** (*int*) – The qubit the unitary gate is applied to.
- **theta** (*CalculatorFloat*) – The angle θ of the rotation.

__init__()

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
powercf(power)	Returns Rotated gate raised to power
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`mul(other)`

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.RotateY

class qoqo.operations.**RotateY**

The YPower gate $e^{-i\frac{\theta}{2}\sigma^y}$.

$$U = \begin{pmatrix} \cos(\frac{\theta}{2}) & 0 \\ 0 & \cos(\frac{\theta}{2}) \end{pmatrix} + \begin{pmatrix} 0 & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & 0 \end{pmatrix}$$

Parameters

- **qubit** (*int*) – The qubit the unitary gate is applied to.
- **theta** (*CalculatorFloat*) – The angle θ of the rotation.

__init__()

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
powercf(power)	Returns Rotated gate raised to power
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`mul(other)`

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.RotateX

class qoqo.operations.**RotateX**

The XPower gate $e^{-i\frac{\theta}{2}\sigma^x}$.

$$U = \begin{pmatrix} \cos(\frac{\theta}{2}) & 0 \\ 0 & \cos(\frac{\theta}{2}) \end{pmatrix} + \begin{pmatrix} 0 & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & 0 \end{pmatrix}$$

Parameters

- **qubit** (*int*) – The qubit the unitary gate is applied to.
- **theta** (*CalculatorFloat*) – The angle θ of the rotation.

__init__()

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
powercf(power)	Returns Rotated gate raised to power
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`mul(other)`

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.RotateAroundSphericalAxis

class qoqo.operations.**RotateAroundSphericalAxis**

Implements a rotation around an axis in the x-y plane in spherical coordinates.

$$U = \begin{pmatrix} \cos(\frac{\theta}{2}) & 0 \\ 0 & \cos(\frac{\theta}{2}) \end{pmatrix} + \begin{pmatrix} -i\sin(\frac{\theta}{2})v_z & \sin(\frac{\theta}{2})\left(-iv_x - v_y\right) \\ \sin(\frac{\theta}{2})\left(-iv_x + v_y\right) & i\sin(\frac{\theta}{2})v_z \end{pmatrix}$$

with

$$\begin{aligned} v_x &= \sin(\theta_{sph})\cos(\phi_{sph})\ , \\ v_y &= \sin(\theta_{sph})\sin(\phi_{sph})\ , \\ v_z &= \cos(\theta_{sph})\ . \end{aligned}$$

Parameters

- **qubit** (*int*) – The qubit the unitary gate is applied to.
- **theta** (*CalculatorFloat*) – The angle θ of the rotation.
- **spherical_theta** (*CalculatorFloat*) – The rotation axis, unit-vector spherical coordinates θ_{sph}.
- **spherical_phi** (*CalculatorFloat*) – The rotation axis, unit-vector spherical coordinates ϕ_{sph} gives the angle in the x-y plane.

__init__()

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
powercf(power)	Returns Rotated gate raised to power
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
spherical_phi()	Returns value of attribute spherical_phi
spherical_theta()	Returns value of attribute spherical_theta
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

spherical_phi()

Returns value of attribute spherical_phi

spherical_theta()

Returns value of attribute spherical_theta

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.PauliZ

class qoqo.operations.**PauliZ**

The Pauli Z gate.

$$U = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Parameters
qubit (*int*) – The qubit the unitary gate is applied to.

__init__()

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns

CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns

CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`mul(other)`

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate].(*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

`qubit()`

Return the qubit the operation acts on

Returns
int

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.PauliY

class qoqo.operations.**PauliY**

The Pauli Y gate.

$$U = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

Parameters

qubit (*int*) – The qubit the unitary gate is applied to.

__init__()

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns

CalculatorFloat

`alpha_r()`

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_i()`

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.PauliX

`class qoqo.operations.PauliX`

The Pauli X gate.

$$U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Parameters
qubit (*int*) – The qubit the unitary gate is applied to.

`__init__()`

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate].(*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

`qubit()`

Return the qubit the operation acts on

Returns
int

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.SqrtPauliX

class qoqo.operations.**SqrtPauliX**

The square root of the XPower gate $e^{-i\frac{\pi}{4}\sigma^x}$.

$$U = \frac{1}{\sqrt{(2)}} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$$

Parameters

qubit (*int*) – The qubit the unitary gate is applied to.

__init__()

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns

CalculatorFloat

`alpha_r()`

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_i()`

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`mul(other)`

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

`qubit()`

Return the qubit the operation acts on

Returns
int

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.InvSqrtPauliX

`class qoqo.operations.InvSqrtPauliX`

The inverse square root XPower gate $e^{i\frac{\pi}{2}\sigma^x}$.

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}$$

Parameters
qubit (*int*) – The qubit the unitary gate is applied to.

`__init__()`

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns

CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns

CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`mul(other)`

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate].(*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

`qubit()`

Return the qubit the operation acts on

Returns
int

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.Hadamard

class qoqo.operations.**Hadamard**

The Hadamard gate.

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Parameters

qubit (*int*) – The qubit the unitary gate is applied to.

__init__()

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns

CalculatorFloat

`alpha_r()`

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_i()`

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.TGate

`class qoqo.operations.TGate`

The T gate.

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$$

Parameters
qubit (*int*) – The qubit the unitary gate is applied to.

`__init__()`

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns

CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns

CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`mul(other)`

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate].(*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.SGate

class qoqo.operations.**SGate**

The S gate.

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

Parameters

qubit (*int*) – The qubit the unitary gate is applied to.

__init__()

Methods

__init__()	
alpha_i()	Return the property alpha_i α_i of a unitary gate acting on one qubit
alpha_r()	Return the property alpha_r α_r of a unitary gate acting on one qubit
beta_i()	Returns the property beta_i β_i of a unitary gate acting on one qubit
beta_r()	Return the property beta_r β_r of a unitary gate acting on one qubit
global_phase()	Return the global phase g of a unitary gate acting on one qubit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
mul(other)	Multiplies two compatible operations implementing OperateSingleQubitGate.
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
unitary_matrix()	Return unitary matrix of gate.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns

CalculatorFloat

`alpha_r()`

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_i()`

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`mul(other)`

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

`qubit()`

Return the qubit the operation acts on

Returns
int

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`
Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`
Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.DefinitionUsize

`class qoqo.operations.DefinitionUsize`
DefinitionUsize is the Definition for an Integer type register.

Parameters

- **name** (*string*) – The name of the register that is defined.
- **length** (*int*) – The length of the register that is defined, usually the number of qubits to be measured.
- **is_output** (*bool*) – True/False if the variable is an output to the program.

`__init__()`

Methods

__init__()	
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_output()	Get value of struct field is_output
is_parametrized()	Returns true if operation contains symbolic parameters
length()	Get value of struct field length
name()	Return name of definition operation.
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_output()`

Get value of struct field is_output

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`length()`

Get value of struct field length

`name()`

Return name of definition operation.

Returns
str

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.DefinitionBit

`class qoqo.operations.DefinitionBit`

DefinitionBit is the Definition for a Bit type register.

Parameters

- **name** (*string*) – The name of the register that is defined.
- **length** (*int*) – The length of the register that is defined, usually the number of qubits to be measured.
- **is_output** (*bool*) – True/False if the variable is an output to the program.

`__init__()`

Methods

__init__()

hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_output()	Get value of struct field is_output
is_parametrized()	Returns true if operation contains symbolic parameters
length()	Get value of struct field length
name()	Return name of definition operation.
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_output()`

Get value of struct field is_output

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`length()`

Get value of struct field length

`name()`

Return name of definition operation.

Returns
str

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.DefinitionFloat

class qoqo.operations.**DefinitionFloat**

DefinitionFloat is the Definition for a Float type register.

Parameters

- **name** (*string*) – The name of the register that is defined.
- **length** (*int*) – The length of the register that is defined, usually the number of qubits to be measured.
- **is_output** (*bool*) – True/False if the variable is an output to the program.

__init__()

Methods

__init__()

hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_output()	Get value of struct field is_output
is_parametrized()	Returns true if operation contains symbolic parameters
length()	Get value of struct field length
name()	Return name of definition operation.
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_output()`

Get value of struct field is_output

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`length()`

Get value of struct field length

`name()`

Return name of definition operation.

Returns
str

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.DefinitionComplex

class qoqo.operations.**DefinitionComplex**

DefinitionComplex is the Definition for a Complex type register.

Parameters

- **name** (*string*) – The name of the register that is defined.
- **length** (*int*) – The length of the register that is defined, usually the number of qubits to be measured.
- **is_output** (*bool*) – True/False if the variable is an output to the program.

__init__()

Methods

__init__()

hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_output()	Get value of struct field is_output
is_parametrized()	Returns true if operation contains symbolic parameters
length()	Get value of struct field length
name()	Return name of definition operation.
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_output()

Get value of struct field is_output

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

`length`()

Get value of struct field length

`name`()

Return name of definition operation.

Returns
str

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.InputSymbolic

class qoqo.operations.**InputSymbolic**

InputSymbolic is the Definition for a Float which will replace a certain symbolic parameter.

Parameters

- **name** (*string*) – The name of the register that is defined.
- **input** (*float*) – The float by which to replace the quantities marked as "name".

__init__()

Methods

__init__()	

hqslang()	Returns hqslang name of Operation
input()	Get value of struct field input
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
name()	Return name of definition operation.
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

input()

Get value of struct field input

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

name()

Return name of definition operation.

Returns
str

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.MeasureQubit

class qoqo.operations.**MeasureQubit**

Measurement gate operation.

This Operation acts on one qubit writing the result of the measurement into a readout. The classical register for the readout needs to be defined in advance by using a Definition operation.

Parameters

- **qubit** (*int*) – The measured qubit.
- **readout** (*string*) – The classical register for the readout.
- **readout_index** (*int*) – The index in the readout the result is saved to.

__init__()

Methods

__init__()

hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
qubit()	Return the qubit the operation acts on
readout()	Get value of struct field readout
readout_index()	Get value of struct field readout_index
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

qubit()

Return the qubit the operation acts on

Returns
int

readout()

Get value of struct field readout

readout_index()

Get value of struct field readout_index

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaGetStateVector

class qoqo.operations.**PragmaGetStateVector**

This PRAGMA measurement operation returns the statevector of a quantum register.

Parameters

- **readout** (*string*) – The name of the classical readout register.
- **circuit** (*Optional[Circuit]*) – The measurement preparation Circuit, applied on a copy of the register before measurement.

__init__()

Methods

__init__()	

circuit()	Get value of struct field circuit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
readout()	Get value of struct field readout
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

circuit()

Get value of struct field circuit

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

readout()

Get value of struct field readout

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaGetDensityMatrix

class qoqo.operations.**PragmaGetDensityMatrix**

This PRAGMA measurement operation returns the density matrix of a quantum register.

Parameters

- **readout** (*string*) – The name of the classical readout register.
- **circuit** (*Optional[Circuit]*) – The measurement preparation Circuit, applied on a copy of the register before measurement.

__init__()

Methods

__init__()	
circuit()	Get value of struct field circuit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
readout()	Get value of struct field readout
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

`circuit()`

Get value of struct field circuit

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`readout()`

Get value of struct field readout

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaGetOccupationProbability

class qoqo.operations.**PragmaGetOccupationProbability**

This PRAGMA measurement operation returns the vector of the occupation probabilities.

Occupation probabilities in the context of this PRAGMA operation are probabilities of finding the quantum register in each σ_z basis state. The quantum register remains unchanged by this PRAGMA measurement operation.

Parameters

- **readout** (*string*) – The name of the classical readout register.
- **circuit** (*Optional[Circuit]*) – The Circuit used to rotate the qureg.

__init__()

Methods

__init__()	
circuit()	Get value of struct field circuit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
readout()	Get value of struct field readout
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

circuit()

Get value of struct field circuit

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`readout()`

Get value of struct field readout

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaGetPauliProduct

class qoqo.operations.**PragmaGetPauliProduct**

This PRAGMA measurement operation returns a Pauli product expectation value.

This PRAGMA operation returns a Pauli product expectation value after applying a Rotate to another basis. It performs all of the operation on a clone of the quantum register, sothat the actual quantum register remains unchanged.

Parameters

- **qubit_paulis** (*dict[int, int]*) – The dictionary of the pauli matrix to apply to each qubit in the form {qubit: pauli}. Allowed values to be provided for 'pauli' are: 0 = identity, 1 = PauliX, 2 = PauliY, 3 = PauliZ.
- **readout** (*string*) – The name of the classical readout register.
- **circuit** (Circuit) – The measurement preparation Circuit, applied on a copy of the register before measurement.

__init__()

Methods

__init__()	
circuit()	Get value of struct field circuit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
qubit_paulis()	Get value of struct field qubit_paulis
readout()	Get value of struct field readout
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

circuit()

Get value of struct field circuit

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`qubit_paulis()`

Get value of struct field qubit_paulis

`readout()`

Get value of struct field readout

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaRepeatedMeasurement

class qoqo.operations.**PragmaRepeatedMeasurement**

This PRAGMA measurement operation returns a measurement record for N repeated measurements.

Parameters

- **readout** (*string*) – The name of the classical readout register.
- **qubit_mapping** (*dict[int, int]*) – The mapping of qubits to indices in readout register.
- **number_measurements** (*int*) – The number of times to repeat the measurement.

__init__()

Methods

__init__()

hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
number_measurements()	Get value of struct field number_measurements
qubit_mapping()	Get value of struct field qubit_mapping
readout()	Get value of struct field readout
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

number_measurements()

Get value of struct field number_measurements

qubit_mapping()

Get value of struct field qubit_mapping

readout()

Get value of struct field readout

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaSetNumberOfMeasurements

class qoqo.operations.**PragmaSetNumberOfMeasurements**

Wrap function automatically generates functions in these traits. This PRAGMA operation sets the number of measurements of the circuit.

This is used for backends that allow setting the number of tries. However, setting the number of measurements does not allow access to the underlying wavefunction or density matrix.

Parameters

- **number_measurements** (*uint*) – The number of measurements.
- **readout** (*string*) – The register for the readout.

__init__()

Methods

__init__()	

hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
number_measurements()	Get value of struct field number_measurements
readout()	Get value of struct field readout
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

number_measurements()

Get value of struct field number_measurements

readout()

Get value of struct field readout

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaSetStateVector

class qoqo.operations.**PragmaSetStateVector**

This PRAGMA operation sets the statevector of a quantum register.

The Circuit() module automatically initializes the qubits in the |0> state, so this PRAGMA operation allows you to set the state of the qubits to a state of your choosing. For instance, to initialize the psi-minus Bell state, we pass the following vector to the PRAGMA:

vector = np.array([0, 1 / np.sqrt(2), -1 / np.sqrt(2), 0])

Parameters
internal (PragmaSetStateVector) – The statevector that is initialized.

__init__()

Methods

__init__()	

hqslang()	Return hqslang name of the operation.
involved_qubits()	List all involved qubits (here, all).
is_parametrized()	Return true when the operation has symbolic parameters.
remap_qubits(mapping)	Remap qubits in a clone of the PRAGMA operation.
statevector()	Return the statevector.
substitute_parameters(substitution_parameters)	Substitute the symbolic parameters in a clone of the PRAGMA operation according to the substitution_parameters input.
tags()	Return tags classifying the type of the operation.

hqslang()

Return hqslang name of the operation.

Returns
The hqslang name of the operation.

Return type
str

involved_qubits()

List all involved qubits (here, all).

Returns
The involved qubits of the PRAGMA operation.

Return type
set[int]

is_parametrized()

Return true when the operation has symbolic parameters.

Returns
True if the operation contains symbolic parameters, False if it does not.

Return type
bool

remap_qubits(*mapping*)

Remap qubits in a clone of the PRAGMA operation.

Parameters
mapping (*dict[int, int]*) – The dictionary containing the {qubit: qubit} mapping to use in the PRAGMA operation.

Returns
The PRAGMA operation with the qubits remapped.

Return type
self

Raises
RuntimeError – The qubit remapping failed.

statevector()

Return the statevector.

Returns
The statevector representing the qubit register.

Return type
np.ndarray

substitute_parameters(*substitution_parameters*)

Substitute the symbolic parameters in a clone of the PRAGMA operation according to the substitution_parameters input.

Parameters
substitution_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the PRAGMA operation.

Returns
The PRAGMA operation operation with the parameters substituted.

Return type
self

Raises
RuntimeError – The parameter substitution failed.

tags()

Return tags classifying the type of the operation.

Used for the type based dispatch in ffi interfaces.

Returns
The tags of the operation.

Return type
list[str]

qoqo.operations.PragmaSetDensityMatrix

class qoqo.operations.**PragmaSetDensityMatrix**

This PRAGMA operation sets the density matrix of a quantum register.

The Circuit() module automatically initializes the qubits in the |0> state, so this PRAGMA operation allows you to set the state of the qubits by setting a density matrix of your choosing.

Parameters
density_matrix (*a 2d array of complex numbers*) – The density matrix that is initialized.

__init__()

Methods

__init__()	

density_matrix()	Return the set density matrix.
hqslang()	Return hqslang name of the operation.
involved_qubits()	List all involved qubits (here, all).
is_parametrized()	Return true when the operation has symbolic parameters.
remap_qubits(mapping)	Remap qubits in a clone of the PRAGMA operation.
substitute_parameters(substitution_parameters)	Substitute the symbolic parameters in a clone of the PRAGMA operation according to the input.
tags()	Return tags classifying the type of the operation.

density_matrix()

Return the set density matrix.

Returns
The density matrix (2d array) representing the qubit register.

Return type
np.ndarray

hqslang()

Return hqslang name of the operation.

Returns
The hqslang name of the operation.

Return type
str

involved_qubits()

List all involved qubits (here, all).

Returns
The involved qubits of the PRAGMA operation.

Return type
set[int]

is_parametrized()

Return true when the operation has symbolic parameters.

Returns
True if the operation contains symbolic parameters, False if it does not.

Return type
bool

remap_qubits(*mapping*)

Remap qubits in a clone of the PRAGMA operation.

Parameters
mapping (*dict[int, int]*) – The dictionary containing the {qubit: qubit} mapping to use in the PRAGMA operation.

Returns
The PRAGMA operation with the qubits remapped.

Return type
self

Raises
RuntimeError – The qubit remapping failed.

substitute_parameters(*substitution_parameters*)

Substitute the symbolic parameters in a clone of the PRAGMA operation according to the input.

Parameters
substitution_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the PRAGMA operation.

Returns
The PRAGMA operation with the parameters substituted.

Return type
self

Raises
RuntimeError – The parameter substitution failed.

tags()

Return tags classifying the type of the operation.

Used for type based dispatch in ffi interfaces.

Returns
The tags of the Operation.

Return type
list[str]

qoqo.operations.PragmaRepeatGate

class qoqo.operations.**PragmaRepeatGate**

The repeated gate PRAGMA operation.

This PRAGMA operation repeats the next gate in the circuit the given number of times to increase the rate for error mitigation.

Parameters
repetition_coefficient (*int*) – The number of times the following gate is repeated.

__init__()

Methods

__init__()	

hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
repetition_coefficient()	Get value of struct field repetition_coefficient
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

repetition_coefficient()

Get value of struct field repetition_coefficient

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaOverrotation

class qoqo.operations.**PragmaOverrotation**

The statistical overrotation PRAGMA operation.

This PRAGMA applies a statistical overrotation to the next rotation gate in the circuit, which matches the hqslang name in the *gate* parameter of PragmaOverrotation and the involved qubits in *qubits*.

The applied overrotation corresponds to adding a random number to the rotation angle. The random number is drawn from a normal distribution with mean *0* and standard deviation *variance* and is multiplied by the *amplitude*.

Parameters

- **gate** (*str*) – The unique hqslang name of the gate to overrotate.
- **qubits** (*list[int]*) – The qubits of the gate to overrotate.
- **amplitude** (*float*) – The amplitude the random number is multiplied by.
- **variance** (*float*) – The standard deviation of the normal distribution the random number is drawn from.

__init__()

Methods

__init__()	

amplitude()	Get value of struct field amplitude
gate_hqslang()	Get value of struct field gate_hqslang
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
qubits()	Return list of qubits of the multi qubit operation in order of descending significance
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
variance()	Get value of struct field variance

amplitude()

Get value of struct field amplitude

gate_hqslang()

Get value of struct field gate_hqslang

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

qubits()

Return list of qubits of the multi qubit operation in order of descending significance

Returns
list[int]

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

variance()

Get value of struct field variance

qoqo.operations.PragmaBoostNoise

class qoqo.operations.**PragmaBoostNoise**

This PRAGMA operation boosts noise and overrotations in the circuit.

Parameters
noise_coefficient (*CalculatorFloat*) – The coefficient by which the noise is boosted.

__init__()

Methods

__init__()	

hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
noise_coefficient()	Returns value of attribute noise_coefficient
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

noise_coefficient()

Returns value of attribute noise_coefficient

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaStopParallelBlock

class qoqo.operations.**PragmaStopParallelBlock**

This PRAGMA operation signals the STOP of a parallel execution block.

Parameters

- **qubits** (*list[int]*) – The qubits involved in parallel execution block.
- **execution_time** (*CalculatorFloat*) – The time for the execution of the block in seconds.

__init__()

Methods

__init__()	

execution_time()	Returns value of attribute execution_time
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
qubits()	Return list of qubits of the multi qubit operation in order of descending significance
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

execution_time()

Returns value of attribute execution_time

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

qubits()

Return list of qubits of the multi qubit operation in order of descending significance

Returns
list[int]

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaGlobalPhase

`class qoqo.operations.PragmaGlobalPhase`

The global phase PRAGMA operation.

This PRAGMA operation signals that the quantum register picks up a global phase, i.e. it provides information that there is a global phase to be considered.

Parameters
phase (*CalculatorFloat*) – The picked up global phase.

`__init__()`

Methods

__init__()	
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
phase()	Returns value of attribute phase
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized`()

Returns true if operation contains symbolic parameters

Returns
bool

`phase`()

Returns value of attribute phase

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaSleep

`class qoqo.operations.PragmaSleep`

This PRAGMA operation makes the quantum hardware wait a given amount of time.

This PRAGMA operation is used for error mitigation reasons, for instance. It can be used to boost the noise on the qubits since it gets worse with time.

Parameters

- **qubits** (*list[int]*) – The qubits involved in the sleep block.

- **sleep_time** (*CalculatorFloat*) – The time for the execution of the block in seconds.

__init__()

Methods

__init__()	

hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
qubits()	Return list of qubits of the multi qubit operation in order of descending significance
remap_qubits(mapping)	Remap qubits
sleep_time()	Returns value of attribute sleep_time
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

qubits()

Return list of qubits of the multi qubit operation in order of descending significance

Returns
list[int]

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`sleep_time()`

Returns value of attribute sleep_time

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaActiveReset

`class qoqo.operations.PragmaActiveReset`

This PRAGMA operation resets the chosen qubit to the zero state.

Parameters
qubit (*int*) – The qubit to be reset.

`__init__()`

Methods

__init__()	

hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaStopDecompositionBlock

class qoqo.operations.**PragmaStopDecompositionBlock**

This PRAGMA operation signals the STOP of a decomposition block.

Parameters
qubits (*list[int]*) – The qubits involved in the decomposition block.

__init__()

Methods

__init__()	

hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
qubits()	Return list of qubits of the multi qubit operation in order of descending significance
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`qubits()`

Return list of qubits of the multi qubit operation in order of descending significance

Returns
list[int]

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaDamping

class qoqo.operations.**PragmaDamping**

The damping PRAGMA noise operation.

This PRAGMA operation applies a pure damping error corresponding to zero temperature environments.

Note

Damping means going from state |*1*> to |*0*> and corresponds to zero-temperature in a physical device where |*0*> is the ground state. With respect to the definition of the Pauli operator *Z*, |*0*> is the excited state and damping leads to an increase in energy.

Parameters

- **qubit** (*int*) – The qubit on which to apply the damping.
- **gate_time** (*CalculatorFloat*) – The time (in seconds) the gate takes to be applied to the qubit on the (simulated) hardware
- **rate** (*CalculatorFloat*) – The error rate of the damping (in 1/second).

__init__()

Methods

__init__()	
gate_time()	Returns value of attribute gate_time
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
powercf(power)	Return the power of the noise gate
probability()	Returns the probability associated with the noise operation
qubit()	Return the qubit the operation acts on
rate()	Returns value of attribute rate
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
superoperator()	Return the superoperator defining the evolution of the density matrix under the noise gate
tags()	Returns tags identifying the Operation

gate_time()

Returns value of attribute gate_time

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`powercf(power)`

Return the power of the noise gate

Parameters
power (*CalculatorFloat*) – exponent in the power operation of the noise gate

Returns
Self

`probability()`

Returns the probability associated with the noise operation

Returns
CalculatorFloat

`qubit()`

Return the qubit the operation acts on

Returns
int

`rate()`

Returns value of attribute rate

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

superoperator()

Return the superoperator defining the evolution of the density matrix under the noise gate

Returns
np.ndarray

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaDepolarising

class qoqo.operations.**PragmaDepolarising**

The depolarising PRAGMA noise operation.

This PRAGMA operation applies a depolarising error corresponding to infinite temperature environments.

Parameters

- **qubit** (*int*) – The qubit on which to apply the depolarising.
- **gate_time** (*CalculatorFloat*) – The time (in seconds) the gate takes to be applied to the qubit on the (simulated) hardware
- **rate** (*CalculatorFloat*) – The error rate of the depolarisation (in 1/second).

__init__()

Methods

__init__()	

gate_time()	Returns value of attribute gate_time
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
powercf(power)	Return the power of the noise gate
probability()	Returns the probability associated with the noise operation
qubit()	Return the qubit the operation acts on
rate()	Returns value of attribute rate
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
superoperator()	Return the superoperator defining the evolution of the density matrix under the noise gate
tags()	Returns tags identifying the Operation

gate_time()

Returns value of attribute gate_time

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Return the power of the noise gate

Parameters
power (*CalculatorFloat*) – exponent in the power operation of the noise gate

Returns
Self

`probability`()

Returns the probability associated with the noise operation

Returns
CalculatorFloat

`qubit`()

Return the qubit the operation acts on

Returns
int

`rate`()

Returns value of attribute rate

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`superoperator`()

Return the superoperator defining the evolution of the density matrix under the noise gate

Returns
np.ndarray

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaDephasing

class qoqo.operations.**PragmaDephasing**

The dephasing PRAGMA noise operation.

This PRAGMA operation applies a pure dephasing error.

Parameters

- **qubit** (*int*) – The qubit on which to apply the dephasing.
- **gate_time** (*CalculatorFloat*) – The time (in seconds) the gate takes to be applied to the qubit on the (simulated) hardware
- **rate** (*CalculatorFloat*) – The error rate of the dephasing (in 1/second).

__init__()

Methods

__init__()	
gate_time()	Returns value of attribute gate_time
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
powercf(power)	Return the power of the noise gate
probability()	Returns the probability associated with the noise operation
qubit()	Return the qubit the operation acts on
rate()	Returns value of attribute rate
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
superoperator()	Return the superoperator defining the evolution of the density matrix under the noise gate
tags()	Returns tags identifying the Operation

gate_time()

Returns value of attribute gate_time

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`powercf(power)`

Return the power of the noise gate

Parameters
power (*CalculatorFloat*) – exponent in the power operation of the noise gate

Returns
Self

`probability()`

Returns the probability associated with the noise operation

Returns
CalculatorFloat

`qubit()`

Return the qubit the operation acts on

Returns
int

`rate()`

Returns value of attribute rate

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`superoperator()`

Return the superoperator defining the evolution of the density matrix under the noise gate

Returns
np.ndarray

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaRandomNoise

`class qoqo.operations.PragmaRandomNoise`

The random noise PRAGMA operation.

This PRAGMA operation applies a pure damping error corresponding to zero temperature environments.

Parameters

- **qubit** (*int*) – The qubit on which to apply the damping.
- **gate_time** (*CalculatorFloat*) – The time (in seconds) the gate takes to be applied to the qubit on the (simulated) hardware
- **depolarising_rate** (*CalculatorFloat*) – The error rate of the depolarisation (in 1/second).
- **dephasing_rate** (*CalculatorFloat*) – The error rate of the dephasing (in 1/second).

`__init__()`

Methods

__init__()	

dephasing_rate()	Returns value of attribute dephasing_rate
depolarising_rate()	Returns value of attribute depolarising_rate
gate_time()	Returns value of attribute gate_time
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
powercf(power)	Return the power of the noise gate
probability()	Returns the probability associated with the noise operation
qubit()	Return the qubit the operation acts on
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
superoperator()	Return the superoperator defining the evolution of the density matrix under the noise gate
tags()	Returns tags identifying the Operation

dephasing_rate()

Returns value of attribute dephasing_rate

depolarising_rate()

Returns value of attribute depolarising_rate

gate_time()

Returns value of attribute gate_time

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Return the power of the noise gate

Parameters
power (*CalculatorFloat*) – exponent in the power operation of the noise gate

Returns
Self

probability()

Returns the probability associated with the noise operation

Returns
CalculatorFloat

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

superoperator()

Return the superoperator defining the evolution of the density matrix under the noise gate

Returns
np.ndarray

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaGeneralNoise

class qoqo.operations.**PragmaGeneralNoise**

The general noise PRAGMA operation.

This PRAGMA operation applies a noise term according to the given operators.

Parameters

- **qubit** (*int*) – The qubit the PRAGMA operation is applied to.
- **gate_time** (*CalculatorFloat*) – The time (in seconds) the gate takes to be applied to the qubit on the (simulated) hardware
- **Rates** – The rates representing the general noise matrix M (a 3x3 matrix as 2d array).

__init__()

Methods

__init__()	
gate_time()	Return the *gate_time* of the PRAGMA operation.
hqslang()	Return hqslang name of the operation.
involved_qubits()	List all involved qubits.
is_parametrized()	Return true when the operation has symbolic parameters.
qubit()	Return the qubit on which the PRAGMA operation is applied.
rates()	Return the rates of the PRAGMA operation.
remap_qubits(mapping)	Remap qubits in a clone of the PRAGMA operation.
substitute_parameters(substitution_parameters)	Substitute the symbolic parameters in a clone of the PRAGMA operation according to the input.
superoperator()	Return the superoperator of the PRAGMA operation.
tags()	Return tags classifying the type of the operation.

gate_time()

Return the *gate_time* of the PRAGMA operation.

Returns
The gate time of the PRAGMA operation.

Return type
CalculatorFloat

hqslang()

Return hqslang name of the operation.

Returns
The hqslang name of the operation.

Return type
str

`involved_qubits`()

List all involved qubits.

Returns
The involved qubits of the PRAGMA operation.

Return type
set[int]

`is_parametrized`()

Return true when the operation has symbolic parameters.

Returns
True if the operation contains symbolic parameters, False if it does not.

Return type
is_parametrized (bool)

`qubit`()

Return the qubit on which the PRAGMA operation is applied.

Returns
The qubit of the PRAGMA operation.

Return type
int

`rates`()

Return the rates of the PRAGMA operation.

Returns
The rates of the PRAGMA operation.

Return type
np.ndarray

`remap_qubits`(*mapping*)

Remap qubits in a clone of the PRAGMA operation.

Parameters
mapping (*dict[int, int]*) – The dictionary containing the {qubit: qubit} mapping to use in the PRAGMA operation.

Returns
The PRAGMA operation with the qubits remapped.

Return type
self

Raises
RuntimeError – The qubit remapping failed.

`substitute_parameters`(*substitution_parameters*)

Substitute the symbolic parameters in a clone of the PRAGMA operation according to the input.

Parameters
substitution_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the PRAGMA operation.

Returns
The PRAGMA operation with the parameters substituted.

Return type
self

Raises
RuntimeError – The parameter substitution failed.

`superoperator()`

Return the superoperator of the PRAGMA operation.

Returns
The matrix form of the superoperator of the PRAGMA operation.

Return type
np.ndarray

`tags()`

Return tags classifying the type of the operation.

Used for the type based dispatch in ffi interfaces.

Returns
The tags of the Operation.

Return type
list[str]

qoqo.operations.PragmaConditional

`class qoqo.operations.PragmaConditional`

The conditional PRAGMA operation.

This PRAGMA executes a circuit when the condition bit/bool stored in a classical bit register is true.

Parameters

- **condition_register** (*str*) – The name of the bit register containting the condition bool value.
- **condition_index** (*int*) –
  - The index in the bit register containting the condition bool value.
- **circuit** (Circuit) –
  - The circuit executed if the condition is met.

`__init__()`

Methods

__init__()	

circuit()	Get value of struct field circuit
condition_index()	Get value of struct field condition_index
condition_register()	Get value of struct field condition_register
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

circuit()

Get value of struct field circuit

condition_index()

Get value of struct field condition_index

condition_register()

Get value of struct field condition_register

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.PragmaLoop

class qoqo.operations.**PragmaLoop**

This PRAGMA measurement operation returns the statevector of a quantum register.

Parameters

- **repetitions** (*CalculatorFloat*) – The number of repetitions as a symbolic float. At evaluation the floor of any float value is taken
- **circuit** (Circuit) – The Circuit that is looped.

__init__()

Methods

__init__()	

circuit()	Get value of struct field circuit
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
repetitions()	Returns value of attribute repetitions
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation

circuit()

Get value of struct field circuit

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

repetitions()

Returns value of attribute repetitions

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

qoqo.operations.CNOT

class qoqo.operations.**CNOT**

The controlled NOT quantum operation.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of NOT on the target qubit.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit NOT is applied to.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.SWAP

class qoqo.operations.**SWAP**

The controlled SWAP quantum operation.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.FSwap

class qoqo.operations.**FSwap**

The controlled fermionic SWAP gate.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.ISwap

class qoqo.operations.**ISwap**

The controlled ISwap quantum operation.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.SqrtISwap

class qoqo.operations.**SqrtISwap**

The controlled square root ISwap quantum operation.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} & 0 \\ 0 & \frac{i}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.InvSqrtISwap

class qoqo.operations.**InvSqrtISwap**

The controlled inverse square root ISwap quantum operation.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-i}{\sqrt{2}} & 0 \\ 0 & \frac{-i}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.XY

class qoqo.operations.**XY**

The controlled XY quantum operation

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta/2) & isin(\theta/2) & 0 \\ 0 & isin(\theta/2) & cos(\theta/2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.
- **theta** (*CalculatorFloat*) – The rotation angle θ.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
powercf(power)	Returns Rotated gate raised to power
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.ControlledPhaseShift

class qoqo.operations.**ControlledPhaseShift**

The controlled-PhaseShift quantum operation.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of the phase-shift on the target qubit.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit phase-shift is applied to.
- **theta** (*CalculatorFloat*) – The rotation angle θ.

__init__()

Methods

__init__()	
control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
powercf(power)	Returns Rotated gate raised to power
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`powercf(power)`

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target()`
Returns target qubit of the two-qubit operation

`theta()`
Returns angle of rotation

`unitary_matrix()`
Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.ControlledPauliY

`class qoqo.operations.ControlledPauliY`
The controlled PauliY quantum operation

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of PauliY gate on the target qubit.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit PauliY is applied to.

`__init__()`

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.ControlledPauliZ

class qoqo.operations.**ControlledPauliZ**

The controlled PauliZ quantum operation

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of PauliZ gate on the target qubit.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit PauliZ is applied to.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.ControlledRotateX

class qoqo.operations.**ControlledRotateX**

Implements the controlled RotateX operation.

The unitary matrix representation is:

$$U = /beginpmatrix 1 0 \\ 0 0 \\ \\ 0 1 \\ 0 0 \\ \\ 0 0 \\ \cos(\frac{\theta}{2}) - i\sin(\frac{\theta}{2}) \\ \\ 0 0 \\ -i\sin(\frac{\theta}{2})\cos(\frac{\theta}{2})$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of the Rotatex Operation on the target qubit.
- **target** (*int*) –
- **theta** (*CalculatorFloat*) – The angle $theta$ of the rotation.

__init__()

Methods

__init__()	
control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
powercf(power)	Returns Rotated gate raised to power
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

`control`()

Returns control qubit of the two-qubit operation

`hqslang`()

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits`()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized`()

Returns true if operation contains symbolic parameters

Returns
bool

`powercf`(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target()`

Returns target qubit of the two-qubit operation

`theta()`

Returns angle of rotation

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.ControlledRotateXY

`class qoqo.operations.ControlledRotateXY`

Implements the controlled RotateXY operation.

The unitary matrix representation is:

$$U = /beginpmatrix 1 0$$
$$0 0$$

$$0 1$$
$$0 0$$

$$0 0$$
$$\cos(\frac{\theta}{2}) - ie^{-i\phi}\sin(\frac{\theta}{2})$$

$$0 0$$
$$-ie^{-i\phi}\sin(\frac{\theta}{2})\cos(\frac{\theta}{2})$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of the Rotatex Operation on the target qubit.
- **target** (*int*) –
- **theta** (*CalculatorFloat*) – The angle $theta$ of the rotation.
- **phi** (*CalculatorFloat*) – The rotation axis, in spherical coordinates ϕ_{sph} gives the angle in the x-y plane.

__init__()

Methods

__init__()

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
phi()	Returns value of attribute phi
powercf(power)	Returns Rotated gate raised to power
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

`phi()`

Returns value of attribute phi

`powercf(power)`

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target()`

Returns target qubit of the two-qubit operation

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.ControlledControlledPauliZ

class qoqo.operations.**ControlledControlledPauliZ**

Implements the double-controlled PauliZ gate.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

Parameters

- **control_0** (*int*) – The index of the most significant qubit in the unitary representation. Here, the first controlling qubit of the operation.
- **control_1** (*int*) – The index of the second most significant qubit in the unitary representation. Here, the second controlling qubit of the operation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit PauliZ is applied to.

__init__()

Methods

__init__()	

circuit()	Returns circuit implementing the ThreeQubitGateOperation
control_0()	Returns control_0 qubit of the three-qubit operation
control_1()	Returns control_1 qubit of the three-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the three-qubit operation
unitary_matrix()	Return unitary matrix of gate.

circuit()

Returns circuit implementing the ThreeQubitGateOperation

Returns
Circuit

control_0()

Returns control_0 qubit of the three-qubit operation

control_1()

Returns control_1 qubit of the three-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the three-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.ControlledControlledPhaseShift

class qoqo.operations.**ControlledControlledPhaseShift**

Implements the double-controlled PhaseShift gate.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\theta} \end{pmatrix}$$

Parameters

- **control_0** (*int*) – The index of the most significant qubit in the unitary representation. Here, the first controlling qubit of the operation.
- **control_1** (*int*) – The index of the second most significant qubit in the unitary representation. Here, the second controlling qubit of the operation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit the phase-shift is applied to.
- **theta** (*float*) – The rotation angle .

__init__()

Methods

__init__()	
circuit()	Returns circuit implementing the ThreeQubitGateOperation
control_0()	Returns control_0 qubit of the three-qubit operation
control_1()	Returns control_1 qubit of the three-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
powercf(power)	Returns Rotated gate raised to power
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the three-qubit operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

`circuit`()

Returns circuit implementing the ThreeQubitGateOperation

Returns
Circuit

`control_0`()

Returns control_0 qubit of the three-qubit operation

`control_1`()

Returns control_1 qubit of the three-qubit operation

`hqslang`()

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits`()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized`()

Returns true if operation contains symbolic parameters

Returns
bool

`powercf`(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the three-qubit operation

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.Toffoli

class qoqo.operations.**Toffoli**

Implements Toffoli gate.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Parameters

- **control_0** (*int*) – The index of the most significant qubit in the unitary representation. Here, the first controlling qubit of the operation.
- **control_1** (*int*) – The index of the second most significant qubit in the unitary representation. Here, the second controlling qubit of the operation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit the PauliX gate is applied to.

__init__()

Methods

__init__()	
circuit()	Returns circuit implementing the ThreeQubitGateOperation
control_0()	Returns control_0 qubit of the three-qubit operation
control_1()	Returns control_1 qubit of the three-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the three-qubit operation
unitary_matrix()	Return unitary matrix of gate.

circuit()

Returns circuit implementing the ThreeQubitGateOperation

Returns
Circuit

`control_0`()

Returns control_0 qubit of the three-qubit operation

`control_1`()

Returns control_1 qubit of the three-qubit operation

`hqslang`()

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits`()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized`()

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the three-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.MolmerSorensenXX

class qoqo.operations.**MolmerSorensenXX**

The fixed phase MolmerSorensen XX gate. <http://arxiv.org/abs/1705.02771>

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & -i \\ 0 & 1 & -i & 0 \\ 0 & -i & 1 & 0 \\ -i & 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. The gate is symmetric under the exchange of qubits.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. The gate is symmetric under the exchange of qubits.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.VariableMSXX

class qoqo.operations.**VariableMSXX**

The variable-angle MolmerSorensen XX gate.

$$U = \begin{pmatrix} \cos(\theta/2) & 0 & 0 & -i\sin(\theta/2) \\ 0 & \cos(\theta/2) & -i\sin(\theta/2) & 0 \\ 0 & -i\sin(\theta/2) & \cos(\theta/2) & 0 \\ -i\sin(\theta/2) & 0 & 0 & \cos(\theta/2) \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. The gate is symmetric under the exchange of qubits.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. The gate is symmetric under the exchange of qubits.
- **theta** (*CalculatorFloat*) – The rotation angle θ.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
powercf(power)	Returns Rotated gate raised to power
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target`()

Returns target qubit of the two-qubit operation

`theta`()

Returns angle of rotation

`unitary_matrix`()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.GivensRotation

class qoqo.operations.**GivensRotation**

The Givens rotation interaction gate in big endian notation: $e^{-\mathrm{i}\theta(X_c Y_t - Y_c X_t)}$.

Where X_c is the Pauli matrix σ^x acting on the control qubit and Y_t is the Pauli matrix σ^y acting on the target qubit.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) \cdot e^{i\phi} & \sin(\theta) & 0 \\ 0 & -\sin(\theta) \cdot e^{i\phi} & \cos(\theta) & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.
- **theta** (*CalculatorFloat*) – The rotation angle θ.
- **phase** (*CalculatorFloat*) – The phase ϕ of the rotation.

__init__()

Methods

__init__()	
control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
phi()	Returns value of attribute phi
powercf(power)	Returns Rotated gate raised to power
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns

The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`phi()`

Returns value of attribute phi

`powercf(power)`

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.GivensRotationLittleEndian

class qoqo.operations.**GivensRotationLittleEndian**

The Givens rotation interaction gate in little endian notation: $e^{-\mathrm{i}\theta(X_c Y_t - Y_c X_t)}$.

Where X_c is the Pauli matrix σ^x acting on the control qubit and Y_t is the Pauli matrix σ^y acting on the target qubit.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) \cdot e^{i\phi} & \cos(\theta) \cdot e^{i\phi} & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.
- **theta** (*CalculatorFloat*) – The rotation angle θ.
- **phase** (*CalculatorFloat*) – The phase ϕ of the rotation.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
phi()	Returns value of attribute phi
powercf(power)	Returns Rotated gate raised to power
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

phi()

Returns value of attribute phi

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target`()

Returns target qubit of the two-qubit operation

`theta`()

Returns angle of rotation

`unitary_matrix`()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.Qsim

class qoqo.operations.**Qsim**

The qubit simulation (Qsim) gate.

$$U = \begin{pmatrix} \cos(x-y)\cdot e^{-iz} & 0 & 0 & -i\sin(x-y)\cdot e^{-iz} \\ 0 & -i\sin(x+y)\cdot e^{iz} & \cos(x+y)\cdot e^{iz} & 0 \\ 0 & \cos(x+y)\cdot e^{iz} & -i\sin(x+y)\cdot e^{iz} & 0 \\ -\sin(x-y)\cdot e^{-iz} & 0 & 0 & \cos(x-y)\cdot e^{-iz} \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) –
- **x** (*CalculatorFloat*) – The prefactor of the XX interaction.
- **y** (*CalculatorFloat*) – The prefactor of the YY interaction.
- **z** (*CalculatorFloat*) – The prefactor of the ZZ interaction.

__init__()

Methods

__init__()	
control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.
x()	Returns value of attribute x
y()	Returns value of attribute y
z()	Returns value of attribute z

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns

The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target()`

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

x()

Returns value of attribute x

y()

Returns value of attribute y

z()

Returns value of attribute z

qoqo.operations.Fsim

class qoqo.operations.**Fsim**

The fermionic qubit simulation (Fsim) gate.

$$U = \begin{pmatrix} \cos(\Delta) & 0 & 0 & i\sin(\Delta) \\ 0 & -i\sin(t) & \cos(t) & 0 \\ 0 & \cos(t) & -i\sin(t) & 0 \\ -\sin(\Delta)\cdot e^{-iU} & 0 & 0 & -\cos(\Delta)\cdot e^{-iU} \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) –
- **t** (*CalculatorFloat*) – The hopping strength.
- **u** (*CalculatorFloat*) – The interaction strength.
- **delta** (*CalculatorFloat*) – The Bogoliubov interaction strength Δ.

Note: The qubits have to be adjacent, i.e., $|i - j| = 1$ has to hold. This is the only case in which the gate is valid as a two-qubit gate (due to the Jordan-Wigner transformation).

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
delta()	Returns value of attribute delta
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
t()	Returns value of attribute t
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
u()	Returns value of attribute u
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

delta()

Returns value of attribute delta

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

t()

Returns value of attribute t

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

u()

Returns value of attribute u

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.SpinInteraction

class qoqo.operations.**SpinInteraction**

The generalized, anisotropic XYZ Heisenberg interaction between spins.

$e^{-\mathrm{i}(x \cdot X_c X_t + y \cdot Y_c Y_t + z \cdot Z_c Z_t)}$

Where x, y, z are prefactors of the X_cX_t, Y_cY_t, Z_cZ_t Pauliproducts acting on control and target qubit, with $XX \equiv \sigma_x\sigma_x$, $YY \equiv \sigma_y\sigma_y$ and $ZZ \equiv \sigma_z\sigma_z$.

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) –
- **x** (*CalculatorFloat*) – The prefactor of the XX interaction.
- **y** (*CalculatorFloat*) – The prefactor of the YY interaction.
- **z** (*CalculatorFloat*) – The prefactor of the ZZ interaction.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.
x()	Returns value of attribute x
y()	Returns value of attribute y
z()	Returns value of attribute z

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

x()

Returns value of attribute x

y()

Returns value of attribute y

z()

Returns value of attribute z

qoqo.operations.Bogoliubov

class qoqo.operations.**Bogoliubov**

The Bogoliubov DeGennes interaction gate.

$e^{-\mathrm{i}Re(\Delta)(X_cX_t-Y_cY_t)/2+Im(\Delta)(X_cY_t+Y_cX_t)/2}$

Where X_c is the Pauli matrix σ^x acting on the control qubit and Y_t is the Pauli matrix σ^y acting on the target qubit.

The unitary matrix representation is:

$$U = \begin{pmatrix} \cos(|\Delta|) & 0 & 0 & \mathrm{i}\sin(|\Delta|)e^{\mathrm{i}\cdot\mathrm{angle}(\Delta)} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \mathrm{i}\sin(|\Delta|)e^{-\mathrm{i}\cdot\mathrm{angle}(\Delta)} & 0 & 0 & \cos(|\Delta|) \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) –
- **delta_real** (*CalculatorFloat*) – The real part of the complex Bogoliubov interaction strength $Re(\Delta)$.
- **delta_imag** (*CalculatorFloat*) – The imaginary part of the complex Bogoliubov interaction strength $Im(\Delta)$.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
delta_imag()	Returns value of attribute delta_imag
delta_real()	Returns value of attribute delta_real
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

`delta_imag()`

Returns value of attribute delta_imag

`delta_real()`

Returns value of attribute delta_real

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.PMInteraction

class qoqo.operations.**PMInteraction**

The transversal interaction gate.

$e^{-i\theta(X_cX_t+Y_cY_t)} = e^{-i\theta(\sigma_c^+\sigma_t^- + \sigma_c^-\sigma_t^+)}$

Where X_c is the Pauli matrix σ^x acting on the control qubit and Y_t is the Pauli matrix σ^y acting on the target qubit.

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) –
- **t** (*CalculatorFloat*) – The strength of the rotation θ.

__init__()

Methods

__init__()	
control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
t()	Returns value of attribute t
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.

`control()`

Returns control qubit of the two-qubit operation

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`t()`

Returns value of attribute t

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.ComplexPMInteraction

class qoqo.operations.**ComplexPMInteraction**

The complex hopping gate.

$e^{-\mathrm{i}[Re(\theta)\cdot(X_cX_t+Y_cY_t)-Im(\theta)\cdot(X_cY_t-Y_cX_t)]}$

Where X_c is the Pauli matrix σ^x acting on the control qubit and Y_t is the Pauli matrix σ^y acting on the target qubit.

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) –
- **t_real** (*CalculatorFloat*) – The real part of the strength of the rotation $Re(\theta)$.
- **t_imag** (*CalculatorFloat*) – The imaginary part of the strength of the rotation $Im(\theta)$.

__init__()

Methods

__init__()	

control()	Returns control qubit of the two-qubit operation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
t_imag()	Returns value of attribute t_imag
t_real()	Returns value of attribute t_real
tags()	Returns tags identifying the Operation
target()	Returns target qubit of the two-qubit operation
unitary_matrix()	Return unitary matrix of gate.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

t_imag()

Returns value of attribute t_imag

t_real()

Returns value of attribute t_real

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

qoqo.operations.MultiQubitMS

class qoqo.operations.**MultiQubitMS**

The Molmer-Sorensen gate between multiple qubits.

The gate applies the rotation under the product of Pauli X operators on multiple qubits. In mathematical terms the gate applies exp(-i * theta/2 * X_i0 * X_i1 * … * X_in).

__init__()

Methods

__init__()	

circuit()	Return circuit implementing MultiQubitGateOperation
hqslang()	Returns hqslang name of Operation
involved_qubits()	List all involved Qubits
is_parametrized()	Returns true if operation contains symbolic parameters
powercf(power)	Returns Rotated gate raised to power
qubits()	Return list of qubits of the multi qubit operation in order of descending significance
remap_qubits(mapping)	Remap qubits
substitute_parameters(substitution_parameters)	Substitutes internal symbolic parameters with float values
tags()	Returns tags identifying the Operation
theta()	Returns angle of rotation
unitary_matrix()	Return unitary matrix of gate.

circuit()

Return circuit implementing MultiQubitGateOperation

Returns
Circuit

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`qubits`()

Return list of qubits of the multi qubit operation in order of descending significance

Returns
list[int]

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`theta`()

Returns angle of rotation

`unitary_matrix`()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

Classes

Bogoliubov	The Bogoliubov DeGennes interaction gate.
CNOT	The controlled NOT quantum operation.
ComplexPMInteraction	The complex hopping gate.
ControlledControlledPauliZ	Implements the double-controlled PauliZ gate.
ControlledControlledPhaseShift	Implements the double-controlled PhaseShift gate.
ControlledPauliY	The controlled PauliY quantum operation
ControlledPauliZ	The controlled PauliZ quantum operation
ControlledPhaseShift	The controlled-PhaseShift quantum operation.
ControlledRotateX	Implements the controlled RotateX operation.
ControlledRotateXY	Implements the controlled RotateXY operation.
DefinitionBit	DefinitionBit is the Definition for a Bit type register.
DefinitionComplex	DefinitionComplex is the Definition for a Complex type register.
DefinitionFloat	DefinitionFloat is the Definition for a Float type register.
DefinitionUsize	DefinitionUsize is the Definition for an Integer type register.
FSwap	The controlled fermionic SWAP gate.
Fsim	The fermionic qubit simulation (Fsim) gate.
GivensRotation	The Givens rotation interaction gate in big endian notation: $e^{-\mathrm{i}\theta(X_cY_t-Y_cX_t)}$.
GivensRotationLittleEndian	The Givens rotation interaction gate in little endian notation: $e^{-\mathrm{i}\theta(X_cY_t-Y_cX_t)}$.
Hadamard	The Hadamard gate.
ISwap	The controlled ISwap quantum operation.
InputBit	InputBit sets a certain bit in an existing BitRegister of the circuit.
InputSymbolic	InputSymbolic is the Definition for a Float which will replace a certain symbolic parameter.
InvSqrtISwap	The controlled inverse square root ISwap quantum operation.
InvSqrtPauliX	The inverse square root XPower gate $e^{i\frac{\pi}{2}\sigma^x}$.
MeasureQubit	Measurement gate operation.
MolmerSorensenXX	The fixed phase MolmerSorensen XX gate.
MultiQubitMS	The Molmer-Sorensen gate between multiple qubits.
MultiQubitZZ	The multi qubit Pauli-Z-Product gate.
PMInteraction	The transversal interaction gate.
PauliX	The Pauli X gate.
PauliY	The Pauli Y gate.
PauliZ	The Pauli Z gate.
PhaseShiftState0	The phase shift gate applied on state |0>.
PhaseShiftState1	The phase shift gate applied on state |1>.
PhaseShiftedControlledPhase	Implements the phase-shifted controlled PhaseShift gate.
PhaseShiftedControlledZ	The phased-shifted controlled-Z gate.
PragmaActiveReset	This PRAGMA operation resets the chosen qubit to the zero state.
PragmaBoostNoise	This PRAGMA operation boosts noise and overrotations in the circuit.

continues on next page

Table 2 – continued from previous page

PragmaChangeDevice	A wrapper around backend specific PRAGMA operations capable of changing a device.
PragmaConditional	The conditional PRAGMA operation.
PragmaDamping	The damping PRAGMA noise operation.
PragmaDephasing	The dephasing PRAGMA noise operation.
PragmaDepolarising	The depolarising PRAGMA noise operation.
PragmaGeneralNoise	The general noise PRAGMA operation.
PragmaGetDensityMatrix	This PRAGMA measurement operation returns the density matrix of a quantum register.
PragmaGetOccupationProbability	This PRAGMA measurement operation returns the vector of the occupation probabilities.
PragmaGetPauliProduct	This PRAGMA measurement operation returns a Pauli product expectation value.
PragmaGetStateVector	This PRAGMA measurement operation returns the statevector of a quantum register.
PragmaGlobalPhase	The global phase PRAGMA operation.
PragmaLoop	This PRAGMA measurement operation returns the statevector of a quantum register.
PragmaOverrotation	The statistical overrotation PRAGMA operation.
PragmaRandomNoise	The random noise PRAGMA operation.
PragmaRepeatGate	The repeated gate PRAGMA operation.
PragmaRepeatedMeasurement	This PRAGMA measurement operation returns a measurement record for N repeated measurements.
PragmaSetDensityMatrix	This PRAGMA operation sets the density matrix of a quantum register.
PragmaSetNumberOfMeasurements	Wrap function automatically generates functions in these traits.
PragmaSetStateVector	This PRAGMA operation sets the statevector of a quantum register.
PragmaSleep	This PRAGMA operation makes the quantum hardware wait a given amount of time.
PragmaStartDecompositionBlock	This PRAGMA operation signals the START of a decomposition block.
PragmaStopDecompositionBlock	This PRAGMA operation signals the STOP of a decomposition block.
PragmaStopParallelBlock	This PRAGMA operation signals the STOP of a parallel execution block.
Qsim	The qubit simulation (Qsim) gate.
RotateAroundSphericalAxis	Implements a rotation around an axis in the x-y plane in spherical coordinates.
RotateX	The XPower gate $e^{-i\frac{\theta}{2}\sigma^x}$.
RotateXY	Implements a rotation around an axis in the x-y plane in spherical coordinates.
RotateY	The YPower gate $e^{-i\frac{\theta}{2}\sigma^y}$.
RotateZ	The ZPower gate $e^{-i\frac{\theta}{2}\sigma^z}$.
SGate	The S gate.
SWAP	The controlled SWAP quantum operation.
SingleQubitGate	The general single qubit unitary gate.
SpinInteraction	The generalized, anisotropic XYZ Heisenberg interaction between spins.

continues on next page

Table 2 – continued from previous page

SqrtISwap	The controlled square root ISwap quantum operation.
SqrtPauliX	The square root of the XPower gate $e^{-i\frac{\pi}{4}\sigma^x}$.
TGate	The T gate.
Toffoli	Implements Toffoli gate.
VariableMSXX	The variable-angle MolmerSorensen XX gate.
XY	The controlled XY quantum operation

class qoqo.operations.**Bogoliubov**

The Bogoliubov DeGennes interaction gate.

$e^{-\mathrm{i}Re(\Delta)(X_cX_t-Y_cY_t)/2+Im(\Delta)(X_cY_t+Y_cX_t)/2}$

Where X_c is the Pauli matrix σ^x acting on the control qubit and Y_t is the Pauli matrix σ^y acting on the target qubit.

The unitary matrix representation is:

$$U = \begin{pmatrix} \cos(|\Delta|) & 0 & 0 & \mathrm{i}\sin(|\Delta|)e^{\mathrm{i}\cdot\mathrm{angle}(\Delta)} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \mathrm{i}\sin(|\Delta|)e^{-\mathrm{i}\cdot\mathrm{angle}(\Delta)} & 0 & 0 & \cos(|\Delta|) \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) –
- **delta_real** (*CalculatorFloat*) – The real part of the complex Bogoliubov interaction strength $Re(\Delta)$.
- **delta_imag** (*CalculatorFloat*) – The imaginary part of the complex Bogoliubov interaction strength $Im(\Delta)$.

control()

Returns control qubit of the two-qubit operation

delta_imag()

Returns value of attribute delta_imag

delta_real()

Returns value of attribute delta_real

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target()`

Returns target qubit of the two-qubit operation

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**CNOT**

The controlled NOT quantum operation.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of NOT on the target qubit.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit NOT is applied to.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**ComplexPMInteraction**

The complex hopping gate.

$e^{-\mathrm{i}[Re(\theta)\cdot(X_cX_t+Y_cY_t)-Im(\theta)\cdot(X_cY_t-Y_cX_t)]}$

Where X_c is the Pauli matrix σ^x acting on the control qubit and Y_t is the Pauli matrix σ^y acting on the target qubit.

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) –
- **t_real** (*CalculatorFloat*) – The real part of the strength of the rotation $Re(\theta)$.
- **t_imag** (*CalculatorFloat*) – The imaginary part of the strength of the rotation $Im(\theta)$.

control()

Returns control qubit of the two-qubit operation

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`t_imag()`

Returns value of attribute t_imag

`t_real()`

Returns value of attribute t_real

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**ControlledControlledPauliZ**

Implements the double-controlled PauliZ gate.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

Parameters

- **control_0** (*int*) – The index of the most significant qubit in the unitary representation. Here, the first controlling qubit of the operation.
- **control_1** (*int*) – The index of the second most significant qubit in the unitary representation. Here, the second controlling qubit of the operation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit PauliZ is applied to.

circuit()

Returns circuit implementing the ThreeQubitGateOperation

Returns
Circuit

control_0()

Returns control_0 qubit of the three-qubit operation

`control_1()`

Returns control_1 qubit of the three-qubit operation

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target()`

Returns target qubit of the three-qubit operation

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

`class qoqo.operations.ControlledControlledPhaseShift`

Implements the double-controlled PhaseShift gate.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\theta} \end{pmatrix}$$

Parameters

- **control_0** (*int*) – The index of the most significant qubit in the unitary representation. Here, the first controlling qubit of the operation.
- **control_1** (*int*) – The index of the second most significant qubit in the unitary representation. Here, the second controlling qubit of the operation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit the phase-shift is applied to.
- **theta** (*float*) – The rotation angle .

`circuit()`

Returns circuit implementing the ThreeQubitGateOperation

Returns
Circuit

`control_0()`

Returns control_0 qubit of the three-qubit operation

`control_1()`

Returns control_1 qubit of the three-qubit operation

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`powercf(power)`

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target()`

Returns target qubit of the three-qubit operation

`theta()`

Returns angle of rotation

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

`class qoqo.operations.ControlledPauliY`

The controlled PauliY quantum operation

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of PauliY gate on the target qubit.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit PauliY is applied to.

`control()`

Returns control qubit of the two-qubit operation

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target()`

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**ControlledPauliZ**

The controlled PauliZ quantum operation

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of PauliZ gate on the target qubit.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit PauliZ is applied to.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**ControlledPhaseShift**

The controlled-PhaseShift quantum operation.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of the phase-shift on the target qubit.

- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit phase-shift is applied to.
- **theta** (*CalculatorFloat*) – The rotation angle θ.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**ControlledRotateX**

Implements the controlled RotateX operation.

The unitary matrix representation is:

$$U = /beginpmatrix 1 0$$
$$0 0$$

$$0 1$$
$$0 0$$

$$0 0$$
$$\cos(\frac{\theta}{2}) - i\sin(\frac{\theta}{2})$$

$$0 0$$
$$-i\sin(\frac{\theta}{2})\cos(\frac{\theta}{2})$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of the Rotatex Operation on the target qubit.
- **target** (*int*) –
- **theta** (*CalculatorFloat*) – The angle $theta$ of the rotation.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**ControlledRotateXY**

Implements the controlled RotateXY operation.

The unitary matrix representation is:

$$U = /beginpmatrix 1 0$$
$$0 0$$

$$0 1$$
$$0 0$$

$$0 0$$
$$\cos(\frac{\theta}{2}) - ie^{-i\phi}\sin(\frac{\theta}{2})$$

$$0 0$$
$$-ie^{-i\phi}\sin(\frac{\theta}{2})\cos(\frac{\theta}{2})$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of the Rotatex Operation on the target qubit.
- **target** (*int*) –
- **theta** (*CalculatorFloat*) – The angle $theta$ of the rotation.
- **phi** (*CalculatorFloat*) – The rotation axis, in spherical coordinates ϕ_{sph} gives the angle in the x-y plane.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

phi()

Returns value of attribute phi

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**DefinitionBit**

DefinitionBit is the Definition for a Bit type register.

Parameters

- **name** (*string*) – The name of the register that is defined.
- **length** (*int*) – The length of the register that is defined, usually the number of qubits to be measured.
- **is_output** (*bool*) – True/False if the variable is an output to the program.

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_output()`

Get value of struct field is_output

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`length()`

Get value of struct field length

`name()`

Return name of definition operation.

Returns
str

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`class qoqo.operations.DefinitionComplex`

DefinitionComplex is the Definition for a Complex type register.

Parameters

- **name** (*string*) – The name of the register that is defined.
- **length** (*int*) – The length of the register that is defined, usually the number of qubits to be measured.
- **is_output** (*bool*) – True/False if the variable is an output to the program.

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_output()`

Get value of struct field is_output

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`length()`

Get value of struct field length

`name()`

Return name of definition operation.

Returns
str

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**DefinitionFloat**

DefinitionFloat is the Definition for a Float type register.

Parameters

- **name** (*string*) – The name of the register that is defined.
- **length** (*int*) – The length of the register that is defined, usually the number of qubits to be measured.
- **is_output** (*bool*) – True/False if the variable is an output to the program.

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_output()`

Get value of struct field is_output

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`length()`

Get value of struct field length

`name()`

Return name of definition operation.

Returns
str

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**DefinitionUsize**

DefinitionUsize is the Definition for an Integer type register.

Parameters

- **name** (*string*) – The name of the register that is defined.
- **length** (*int*) – The length of the register that is defined, usually the number of qubits to be measured.
- **is_output** (*bool*) – True/False if the variable is an output to the program.

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_output()

Get value of struct field is_output

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

length()

Get value of struct field length

name()

Return name of definition operation.

Returns
str

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**FSwap**

The controlled fermionic SWAP gate.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target()`

Returns target qubit of the two-qubit operation

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**Fsim**

The fermionic qubit simulation (Fsim) gate.

$$U = \begin{pmatrix} \cos(\Delta) & 0 & 0 & i\sin(\Delta) \\ 0 & -i\sin(t) & \cos(t) & 0 \\ 0 & \cos(t) & -i\sin(t) & 0 \\ -\sin(\Delta)\cdot e^{-iU} & 0 & 0 & -\cos(\Delta)\cdot e^{-iU} \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) –
- **t** (*CalculatorFloat*) – The hopping strength.
- **u** (*CalculatorFloat*) – The interaction strength.
- **delta** (*CalculatorFloat*) – The Bogoliubov interaction strength Δ.

Note: The qubits have to be adjacent, i.e., $|i - j| = 1$ has to hold. This is the only case in which the gate is valid as a two-qubit gate (due to the Jordan-Wigner transformation).

control()

Returns control qubit of the two-qubit operation

delta()

Returns value of attribute delta

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

t()

Returns value of attribute t

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

u()

Returns value of attribute u

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**GivensRotation**

The Givens rotation interaction gate in big endian notation: $e^{-\mathrm{i}\theta(X_c Y_t - Y_c X_t)}$.

Where X_c is the Pauli matrix σ^x acting on the control qubit and Y_t is the Pauli matrix σ^y acting on the target qubit.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) \cdot e^{i\phi} & \sin(\theta) & 0 \\ 0 & -\sin(\theta) \cdot e^{i\phi} & \cos(\theta) & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.
- **theta** (*CalculatorFloat*) – The rotation angle θ.
- **phase** (*CalculatorFloat*) – The phase ϕ of the rotation.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

phi()

Returns value of attribute phi

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target`()

Returns target qubit of the two-qubit operation

`theta`()

Returns angle of rotation

`unitary_matrix`()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**GivensRotationLittleEndian**

The Givens rotation interaction gate in little endian notation: $e^{-\mathrm{i}\theta(X_c Y_t - Y_c X_t)}$.

Where X_c is the Pauli matrix σ^x acting on the control qubit and Y_t is the Pauli matrix σ^y acting on the target qubit.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) \cdot e^{i\phi} & \cos(\theta) \cdot e^{i\phi} & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.
- **theta** (*CalculatorFloat*) – The rotation angle θ.
- **phase** (*CalculatorFloat*) – The phase ϕ of the rotation.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

phi()

Returns value of attribute phi

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target`()

Returns target qubit of the two-qubit operation

`theta`()

Returns angle of rotation

`unitary_matrix`()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**Hadamard**

The Hadamard gate.

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Parameters
qubit (*int*) – The qubit the unitary gate is applied to.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate].(*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**ISwap**

The controlled ISwap quantum operation.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**InputBit**

InputBit sets a certain bit in an existing BitRegister of the circuit.

Parameters

- **name** (*string*) – The name of the register that is defined.
- **index** (*int*) – The index in the register that is set.
- **value** (*int*) – The value the bit is set to.

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

index()

Get value of struct field index

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

name()

Return name of definition operation.

Returns
str

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`value`()

Get value of struct field value

`class qoqo.operations.InputSymbolic`

InputSymbolic is the Definition for a Float which will replace a certain symbolic parameter.

Parameters

- **name** (*string*) – The name of the register that is defined.
- **input** (*float*) – The float by which to replace the quantities marked as "name".

`hqslang`()

Returns hqslang name of Operation

Returns
The name

Return type
str

input()

Get value of struct field input

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

name()

Return name of definition operation.

Returns
str

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**InvSqrtISwap**

The controlled inverse square root ISwap quantum operation.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-i}{\sqrt{2}} & 0 \\ 0 & \frac{-i}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**InvSqrtPauliX**

The inverse square root XPower gate $e^{i\frac{\pi}{2}\sigma^x}$.

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}$$

Parameters
qubit (*int*) – The qubit the unitary gate is applied to.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`alpha_r()`

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_i()`

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

`class qoqo.operations.MeasureQubit`

Measurement gate operation.

This Operation acts on one qubit writing the result of the measurement into a readout. The classical register for the readout needs to be defined in advance by using a Definition operation.

Parameters

- **qubit** (*int*) – The measured qubit.
- **readout** (*string*) – The classical register for the readout.
- **readout_index** (*int*) – The index in the readout the result is saved to.

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

qubit()

Return the qubit the operation acts on

Returns
int

readout()

Get value of struct field readout

readout_index()

Get value of struct field readout_index

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**MolmerSorensenXX**

The fixed phase MolmerSorensen XX gate. <http://arxiv.org/abs/1705.02771>

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & -i \\ 0 & 1 & -i & 0 \\ 0 & -i & 1 & 0 \\ -i & 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. The gate is symmetric under the exchange of qubits.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. The gate is symmetric under the exchange of qubits.

`control()`

Returns control qubit of the two-qubit operation

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**MultiQubitMS**

The Molmer-Sorensen gate between multiple qubits.

The gate applies the rotation under the product of Pauli X operators on multiple qubits. In mathematical terms the gate applies exp(-i * theta/2 * X_i0 * X_i1 * … * X_in).

circuit()

Return circuit implementing MultiQubitGateOperation

Returns
Circuit

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`powercf(power)`

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`qubits()`

Return list of qubits of the multi qubit operation in order of descending significance

Returns
list[int]

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**MultiQubitZZ**

The multi qubit Pauli-Z-Product gate.

The gate applies the rotation under the product of Pauli Z operators on multiple qubits. In mathematical terms the gate applies exp(-i * theta/2 * Z_i0 * Z_i1 * … * Z_in).

circuit()

Return circuit implementing MultiQubitGateOperation

Returns
Circuit

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`qubits()`

Return list of qubits of the multi qubit operation in order of descending significance

Returns
list[int]

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`theta()`

Returns angle of rotation

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**PMInteraction**

The transversal interaction gate.

$e^{-i\theta(X_cX_t+Y_cY_t)} = e^{-i\theta(\sigma_c^+\sigma_t^- + \sigma_c^-\sigma_t^+)}$

Where X_c is the Pauli matrix σ^x acting on the control qubit and Y_t is the Pauli matrix σ^y acting on the target qubit.

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) –
- **t** (*CalculatorFloat*) – The strength of the rotation θ.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

t()

Returns value of attribute t

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**PauliX**

The Pauli X gate.

$$U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Parameters
qubit (*int*) – The qubit the unitary gate is applied to.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`alpha_r()`

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_i()`

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`mul(other)`

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

`qubit()`

Return the qubit the operation acts on

Returns
int

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

`class qoqo.operations.PauliY`

The Pauli Y gate.

$$U = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

Parameters
qubit (*int*) – The qubit the unitary gate is applied to.

`alpha_i()`

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`alpha_r()`

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

`class qoqo.operations.PauliZ`

The Pauli Z gate.

$$U = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Parameters
qubit (*int*) – The qubit the unitary gate is applied to.

`alpha_i()`

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`alpha_r()`

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_i()`

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate].(*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

`qubit()`

Return the qubit the operation acts on

Returns
int

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**PhaseShiftState0**

The phase shift gate applied on state |0>.

Rotation around Z-axis by an arbitrary angle $theta$ (AC Stark shift of the state |0>).

$$U = \begin{pmatrix} e^{i\theta} & 0 \\ 0 & 1 \end{pmatrix}$$

Parameters

- **qubit** (*int*) – The qubit the unitary gate is applied to.
- **theta** (*CalculatorFloat*) – The angle θ of the rotation.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`qubit`()

Return the qubit the operation acts on

Returns
int

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`theta`()

Returns angle of rotation

`unitary_matrix`()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**PhaseShiftState1**

The phase shift gate applied on state |1>.

Rotation around Z-axis by an arbitrary angle $theta$ (AC Stark shift of the state |1>).

$$U = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

Parameters

- **qubit** (*int*) – The qubit the unitary gate is applied to.
- **theta** (*CalculatorFloat*) – The angle θ of the rotation.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`qubit()`

Return the qubit the operation acts on

Returns
int

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`theta()`

Returns angle of rotation

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**PhaseShiftedControlledPhase**

Implements the phase-shifted controlled PhaseShift gate.

The unitary matrix representation is:

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\phi} & 0 & 0 \\ 0 & 0 & e^{i\phi} & 0 \\ 0 & 0 & 0 & e^{i(2\cdot\phi+\theta)} \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of the phase-shift on the target qubit.
- **target** (*int*) –
- **theta** (*CalculatorFloat*) – The phase rotation $theta$.
- **phi** (*CalculatorFloat*) – The single qubit phase phi.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

phi()

Returns value of attribute phi

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target`()

Returns target qubit of the two-qubit operation

`theta`()

Returns angle of rotation

`unitary_matrix`()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**PhaseShiftedControlledZ**

The phased-shifted controlled-Z gate.

Modified, i.e. phase-shifted ControlledPauliZ two-qubit gate (<*https://arxiv.org/pdf/1908.06101.pdf eq.(1)*>).

The unitary matrix representation is:

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\phi} & 0 & 0 \\ 0 & 0 & e^{i\phi} & 0 \\ 0 & 0 & 0 & e^{i(2\cdot\phi+\pi)} \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. Here, the qubit that controls the application of the phase-shift on the target qubit.
- **target** (*int*) –
- **phi** (*CalculatorFloat*) – The single qubit phase phi.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

phi()

Returns value of attribute phi

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**PragmaActiveReset**

This PRAGMA operation resets the chosen qubit to the zero state.

Parameters
qubit (*int*) – The qubit to be reset.

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`qubit()`

Return the qubit the operation acts on

Returns
int

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`class qoqo.operations.PragmaBoostNoise`

This PRAGMA operation boosts noise and overrotations in the circuit.

Parameters
noise_coefficient (*CalculatorFloat*) – The coefficient by which the noise is boosted.

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

noise_coefficient()

Returns value of attribute noise_coefficient

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**PragmaChangeDevice**

A wrapper around backend specific PRAGMA operations capable of changing a device.

This PRAGMA is a thin wrapper around device specific operations that can change device properties.

hqslang()

Return hqslang name of the operation.

Returns
The hqslang name of the operation.

Return type
str

involved_qubits()

List all involved qubits.

Returns
The involved qubits of the PRAGMA operation.

Return type
set[int]

is_parametrized()

Return true when the operation has symbolic parameters.

Returns
True if the operation contains symbolic parameters, False if it does not.

Return type
is_parametrized (bool)

remap_qubits(*mapping*)

Remap qubits in a clone of the PRAGMA operation.

Parameters
mapping (*dict[int, int]*) – The dictionary containing the {qubit: qubit} mapping to use in the PRAGMA operation.

Returns
The PRAGMA operation with the qubits remapped.

Return type
self

Raises
RuntimeError – The qubit remapping failed.

substitute_parameters(*substitution_parameters*)

Substitute the symbolic parameters in a clone of the PRAGMA operation according to the input.

Parameters
substitution_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the PRAGMA operation.

Returns
The PRAGMA operation with the parameters substituted.

Return type
self

Raises
RuntimeError – The parameter substitution failed.

`tags()`

Return tags classifying the type of the operation.

Used for the type based dispatch in ffi interfaces.

Returns
The tags of the Operation.

Return type
list[str]

`wrapped_hqslang()`

Return the hqslang name of the wrapped operations.

Returns
The name of the wrapped operation.

Return type
str

`wrapped_operation()`

Return the binary representation of the wrapped operations.

Returns
The the binary representation of the wrapped operation.

Return type
ByteArray

`wrapped_tags()`

Return the tags of the wrapped operations.

Returns
The list of tags.

Return type
List[str]

`class qoqo.operations.PragmaConditional`

The conditional PRAGMA operation.

This PRAGMA executes a circuit when the condition bit/bool stored in a classical bit register is true.

Parameters

- **condition_register** (*str*) – The name of the bit register containting the condition bool value.
- **condition_index** (*int*) –
  - The index in the bit register containting the condition bool value.
- **circuit** (Circuit) –
  - The circuit executed if the condition is met.

`circuit()`

Get value of struct field circuit

`condition_index()`

Get value of struct field condition_index

`condition_register()`

Get value of struct field condition_register

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**PragmaDamping**

The damping PRAGMA noise operation.

This PRAGMA operation applies a pure damping error corresponding to zero temperature environments.

Note

Damping means going from state |*1*> to |*0*> and corresponds to zero-temperature in a physical device where |*0*> is the ground state. With respect to the definition of the Pauli operator *Z*, |*0*> is the excited state and damping leads to an increase in energy.

Parameters

- **qubit** (*int*) – The qubit on which to apply the damping.
- **gate_time** (*CalculatorFloat*) – The time (in seconds) the gate takes to be applied to the qubit on the (simulated) hardware
- **rate** (*CalculatorFloat*) – The error rate of the damping (in 1/second).

gate_time()

Returns value of attribute gate_time

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Return the power of the noise gate

Parameters
power (*CalculatorFloat*) – exponent in the power operation of the noise gate

Returns
Self

probability()

Returns the probability associated with the noise operation

Returns
CalculatorFloat

qubit()

Return the qubit the operation acts on

Returns
int

rate()

Returns value of attribute rate

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

superoperator()

Return the superoperator defining the evolution of the density matrix under the noise gate

Returns
np.ndarray

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**PragmaDephasing**

The dephasing PRAGMA noise operation.

This PRAGMA operation applies a pure dephasing error.

Parameters

- **qubit** (*int*) – The qubit on which to apply the dephasing.
- **gate_time** (*CalculatorFloat*) – The time (in seconds) the gate takes to be applied to the qubit on the (simulated) hardware
- **rate** (*CalculatorFloat*) – The error rate of the dephasing (in 1/second).

gate_time()

Returns value of attribute gate_time

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Return the power of the noise gate

Parameters
power (*CalculatorFloat*) – exponent in the power operation of the noise gate

Returns
Self

probability()

Returns the probability associated with the noise operation

Returns
CalculatorFloat

`qubit`()

Return the qubit the operation acts on

Returns
int

`rate`()

Returns value of attribute rate

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`superoperator`()

Return the superoperator defining the evolution of the density matrix under the noise gate

Returns
np.ndarray

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`class qoqo.operations.PragmaDepolarising`

The depolarising PRAGMA noise operation.

This PRAGMA operation applies a depolarising error corresponding to infinite temperature environments.

Parameters

- **qubit** (*int*) – The qubit on which to apply the depolarising.
- **gate_time** (*CalculatorFloat*) – The time (in seconds) the gate takes to be applied to the qubit on the (simulated) hardware
- **rate** (*CalculatorFloat*) – The error rate of the depolarisation (in 1/second).

gate_time()

Returns value of attribute gate_time

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Return the power of the noise gate

Parameters
power (*CalculatorFloat*) – exponent in the power operation of the noise gate

Returns
Self

probability()

Returns the probability associated with the noise operation

Returns
CalculatorFloat

qubit()

Return the qubit the operation acts on

Returns
int

rate()

Returns value of attribute rate

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

superoperator()

Return the superoperator defining the evolution of the density matrix under the noise gate

Returns
np.ndarray

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**PragmaGeneralNoise**

The general noise PRAGMA operation.

This PRAGMA operation applies a noise term according to the given operators.

Parameters

- **qubit** (*int*) – The qubit the PRAGMA operation is applied to.
- **gate_time** (*CalculatorFloat*) – The time (in seconds) the gate takes to be applied to the qubit on the (simulated) hardware
- **Rates** – The rates representing the general noise matrix M (a 3x3 matrix as 2d array).

gate_time()

Return the *gate_time* of the PRAGMA operation.

Returns
The gate time of the PRAGMA operation.

Return type
CalculatorFloat

`hqslang()`

Return hqslang name of the operation.

Returns
The hqslang name of the operation.

Return type
str

`involved_qubits()`

List all involved qubits.

Returns
The involved qubits of the PRAGMA operation.

Return type
set[int]

`is_parametrized()`

Return true when the operation has symbolic parameters.

Returns
True if the operation contains symbolic parameters, False if it does not.

Return type
is_parametrized (bool)

`qubit()`

Return the qubit on which the PRAGMA operation is applied.

Returns
The qubit of the PRAGMA operation.

Return type
int

`rates()`

Return the rates of the PRAGMA operation.

Returns
The rates of the PRAGMA operation.

Return type
np.ndarray

`remap_qubits(`*mapping*`)`

Remap qubits in a clone of the PRAGMA operation.

Parameters
mapping (*dict[int, int]*) – The dictionary containing the {qubit: qubit} mapping to use in the PRAGMA operation.

Returns
The PRAGMA operation with the qubits remapped.

Return type
self

Raises
RuntimeError – The qubit remapping failed.

substitute_parameters(*substitution_parameters*)

Substitute the symbolic parameters in a clone of the PRAGMA operation according to the input.

Parameters
substitution_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the PRAGMA operation.

Returns
The PRAGMA operation with the parameters substituted.

Return type
self

Raises
RuntimeError – The parameter substitution failed.

superoperator()

Return the superoperator of the PRAGMA operation.

Returns
The matrix form of the superoperator of the PRAGMA operation.

Return type
np.ndarray

tags()

Return tags classifying the type of the operation.

Used for the type based dispatch in ffi interfaces.

Returns
The tags of the Operation.

Return type
list[str]

class qoqo.operations.**PragmaGetDensityMatrix**

This PRAGMA measurement operation returns the density matrix of a quantum register.

Parameters

- **readout** (*string*) – The name of the classical readout register.
- **circuit** (*Optional[Circuit]*) – The measurement preparation Circuit, applied on a copy of the register before measurement.

circuit()

Get value of struct field circuit

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`readout()`

Get value of struct field readout

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`class qoqo.operations.PragmaGetOccupationProbability`

This PRAGMA measurement operation returns the vector of the occupation probabilities.

Occupation probabilities in the context of this PRAGMA operation are probabilities of finding the quantum register in each σ_z basis state. The quantum register remains unchanged by this PRAGMA measurement operation.

Parameters

- **readout** (*string*) – The name of the classical readout register.
- **circuit** (*Optional[Circuit]*) – The Circuit used to rotate the qureg.

`circuit()`

Get value of struct field circuit

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`readout()`

Get value of struct field readout

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.PragmaGetPauliProduct

This PRAGMA measurement operation returns a Pauli product expectation value.

This PRAGMA operation returns a Pauli product expectation value after applying a Rotate to another basis. It performs all of the operation on a clone of the quantum register, sothat the actual quantum register remains unchanged.

Parameters

- **qubit_paulis** (*dict[int, int]*) – The dictionary of the pauli matrix to apply to each qubit in the form {qubit: pauli}. Allowed values to be provided for 'pauli' are: 0 = identity, 1 = PauliX, 2 = PauliY, 3 = PauliZ.
- **readout** (*string*) – The name of the classical readout register.
- **circuit** (Circuit) – The measurement preparation Circuit, applied on a copy of the register before measurement.

circuit()

Get value of struct field circuit

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

qubit_paulis()

Get value of struct field qubit_paulis

readout()

Get value of struct field readout

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**PragmaGetStateVector**

This PRAGMA measurement operation returns the statevector of a quantum register.

Parameters

- **readout** (*string*) – The name of the classical readout register.
- **circuit** (*Optional[Circuit]*) – The measurement preparation Circuit, applied on a copy of the register before measurement.

circuit()

Get value of struct field circuit

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`readout()`

Get value of struct field readout

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**PragmaGlobalPhase**

The global phase PRAGMA operation.

This PRAGMA operation signals that the quantum register picks up a global phase, i.e. it provides information that there is a global phase to be considered.

Parameters
phase (*CalculatorFloat*) – The picked up global phase.

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

phase()

Returns value of attribute phase

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`class qoqo.operations.PragmaLoop`

This PRAGMA measurement operation returns the statevector of a quantum register.

Parameters

- **repetitions** (*CalculatorFloat*) – The number of repetitions as a symbolic float. At evaluation the floor of any float value is taken
- **circuit** (Circuit) – The Circuit that is looped.

`circuit()`

Get value of struct field circuit

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

repetitions()

Returns value of attribute repetitions

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**PragmaOverrotation**

The statistical overrotation PRAGMA operation.

This PRAGMA applies a statistical overrotation to the next rotation gate in the circuit, which matches the hqslang name in the *gate* parameter of PragmaOverrotation and the involved qubits in *qubits*.

The applied overrotation corresponds to adding a random number to the rotation angle. The random number is drawn from a normal distribution with mean *0* and standard deviation *variance* and is multiplied by the *amplitude*.

Parameters

- **gate** (*str*) – The unique hqslang name of the gate to overrotate.
- **qubits** (*list[int]*) – The qubits of the gate to overrotate.
- **amplitude** (*float*) – The amplitude the random number is multiplied by.
- **variance** (*float*) – The standard deviation of the normal distribution the random number is drawn from.

amplitude()

Get value of struct field amplitude

gate_hqslang()

Get value of struct field gate_hqslang

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`qubits()`

Return list of qubits of the multi qubit operation in order of descending significance

Returns
list[int]

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`variance()`

Get value of struct field variance

`class qoqo.operations.PragmaRandomNoise`

The random noise PRAGMA operation.

This PRAGMA operation applies a pure damping error corresponding to zero temperature environments.

Parameters

- **qubit** (*int*) – The qubit on which to apply the damping.
- **gate_time** (*CalculatorFloat*) – The time (in seconds) the gate takes to be applied to the qubit on the (simulated) hardware
- **depolarising_rate** (*CalculatorFloat*) – The error rate of the depolarisation (in 1/second).
- **dephasing_rate** (*CalculatorFloat*) – The error rate of the dephasing (in 1/second).

`dephasing_rate()`

Returns value of attribute dephasing_rate

`depolarising_rate()`

Returns value of attribute depolarising_rate

`gate_time()`

Returns value of attribute gate_time

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`powercf(power)`

Return the power of the noise gate

Parameters
power (*CalculatorFloat*) – exponent in the power operation of the noise gate

Returns
Self

`probability`()

Returns the probability associated with the noise operation

Returns
CalculatorFloat

`qubit`()

Return the qubit the operation acts on

Returns
int

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`superoperator`()

Return the superoperator defining the evolution of the density matrix under the noise gate

Returns
np.ndarray

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.PragmaRepeatGate

The repeated gate PRAGMA operation.

This PRAGMA operation repeats the next gate in the circuit the given number of times to increase the rate for error mitigation.

Parameters
repetition_coefficient (*int*) – The number of times the following gate is repeated.

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

repetition_coefficient()

Get value of struct field repetition_coefficient

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**PragmaRepeatedMeasurement**

This PRAGMA measurement operation returns a measurement record for N repeated measurements.

Parameters

- **readout** (*string*) – The name of the classical readout register.
- **qubit_mapping** (*dict[int, int]*) – The mapping of qubits to indices in readout register.
- **number_measurements** (*int*) – The number of times to repeat the measurement.

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

number_measurements()

Get value of struct field number_measurements

qubit_mapping()

Get value of struct field qubit_mapping

readout()

Get value of struct field readout

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**PragmaSetDensityMatrix**

This PRAGMA operation sets the density matrix of a quantum register.

The Circuit() module automatically initializes the qubits in the |0> state, so this PRAGMA operation allows you to set the state of the qubits by setting a density matrix of your choosing.

Parameters
density_matrix (*a 2d array of complex numbers*) – The density matrix that is initialized.

density_matrix()

Return the set density matrix.

Returns
The density matrix (2d array) representing the qubit register.

Return type
np.ndarray

hqslang()

Return hqslang name of the operation.

Returns
The hqslang name of the operation.

Return type
str

`involved_qubits()`

List all involved qubits (here, all).

Returns
The involved qubits of the PRAGMA operation.

Return type
set[int]

`is_parametrized()`

Return true when the operation has symbolic parameters.

Returns
True if the operation contains symbolic parameters, False if it does not.

Return type
bool

`remap_qubits(mapping)`

Remap qubits in a clone of the PRAGMA operation.

Parameters
mapping (*dict[int, int]*) – The dictionary containing the {qubit: qubit} mapping to use in the PRAGMA operation.

Returns
The PRAGMA operation with the qubits remapped.

Return type
self

Raises
RuntimeError – The qubit remapping failed.

`substitute_parameters(substitution_parameters)`

Substitute the symbolic parameters in a clone of the PRAGMA operation according to the input.

Parameters
substitution_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the PRAGMA operation.

Returns
The PRAGMA operation with the parameters substituted.

Return type
self

Raises
RuntimeError – The parameter substitution failed.

`tags()`

Return tags classifying the type of the operation.

Used for type based dispatch in ffi interfaces.

Returns
The tags of the Operation.

Return type
list[str]

class qoqo.operations.PragmaSetNumberOfMeasurements

Wrap function automatically generates functions in these traits. This PRAGMA operation sets the number of measurements of the circuit.

This is used for backends that allow setting the number of tries. However, setting the number of measurements does not allow access to the underlying wavefunction or density matrix.

Parameters

- **number_measurements** (*uint*) – The number of measurements.
- **readout** (*string*) – The register for the readout.

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

number_measurements()

Get value of struct field number_measurements

readout()

Get value of struct field readout

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`class qoqo.operations.PragmaSetStateVector`

This PRAGMA operation sets the statevector of a quantum register.

The Circuit() module automatically initializes the qubits in the |0> state, so this PRAGMA operation allows you to set the state of the qubits to a state of your choosing. For instance, to initialize the psi-minus Bell state, we pass the following vector to the PRAGMA:

vector = np.array([0, 1 / np.sqrt(2), -1 / np.sqrt(2), 0])

Parameters
internal (PragmaSetStateVector) – The statevector that is initialized.

`hqslang()`

Return hqslang name of the operation.

Returns
The hqslang name of the operation.

Return type
str

`involved_qubits()`

List all involved qubits (here, all).

Returns
The involved qubits of the PRAGMA operation.

Return type
set[int]

`is_parametrized()`

Return true when the operation has symbolic parameters.

Returns
True if the operation contains symbolic parameters, False if it does not.

Return type
bool

remap_qubits(*mapping*)

Remap qubits in a clone of the PRAGMA operation.

Parameters
mapping (*dict[int, int]*) – The dictionary containing the {qubit: qubit} mapping to use in the PRAGMA operation.

Returns
The PRAGMA operation with the qubits remapped.

Return type
self

Raises
RuntimeError – The qubit remapping failed.

statevector()

Return the statevector.

Returns
The statevector representing the qubit register.

Return type
np.ndarray

substitute_parameters(*substitution_parameters*)

Substitute the symbolic parameters in a clone of the PRAGMA operation according to the substitution_parameters input.

Parameters
substitution_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the PRAGMA operation.

Returns
The PRAGMA operation operation with the parameters substituted.

Return type
self

Raises
RuntimeError – The parameter substitution failed.

tags()

Return tags classifying the type of the operation.

Used for the type based dispatch in ffi interfaces.

Returns
The tags of the operation.

Return type
list[str]

class qoqo.operations.**PragmaSleep**

This PRAGMA operation makes the quantum hardware wait a given amount of time.

This PRAGMA operation is used for error mitigation reasons, for instance. It can be used to boost the noise on the qubits since it gets worse with time.

Parameters

- **qubits** (*list[int]*) – The qubits involved in the sleep block.

- **sleep_time** (*CalculatorFloat*) – The time for the execution of the block in seconds.

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

qubits()

Return list of qubits of the multi qubit operation in order of descending significance

Returns
list[int]

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

sleep_time()

Returns value of attribute sleep_time

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**PragmaStartDecompositionBlock**

This PRAGMA operation signals the START of a decomposition block.

Parameters

- **qubits** (*list[int]*) – The qubits involved in the decomposition block.
- **dict[int** (*reordering_dictionary*) – The reordering dictionary of the block.
- **int])** – The reordering dictionary of the block.

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

qubits()

Return list of qubits of the multi qubit operation in order of descending significance

Returns
list[int]

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`reordering_dictionary()`

Get value of struct field reordering_dictionary

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`class qoqo.operations.PragmaStopDecompositionBlock`

This PRAGMA operation signals the STOP of a decomposition block.

Parameters
qubits (*list[int]*) – The qubits involved in the decomposition block.

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`qubits()`

Return list of qubits of the multi qubit operation in order of descending significance

Returns
list[int]

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`class qoqo.operations.PragmaStopParallelBlock`

This PRAGMA operation signals the STOP of a parallel execution block.

Parameters

- **qubits** (*list[int]*) – The qubits involved in parallel execution block.
- **execution_time** (*CalculatorFloat*) – The time for the execution of the block in seconds.

`execution_time()`

Returns value of attribute execution_time

`hqslang`()

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits`()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized`()

Returns true if operation contains symbolic parameters

Returns
bool

`qubits`()

Return list of qubits of the multi qubit operation in order of descending significance

Returns
list[int]

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

class qoqo.operations.**Qsim**

The qubit simulation (Qsim) gate.

$$U = \begin{pmatrix} \cos(x-y)\cdot e^{-iz} & 0 & 0 & -i\sin(x-y)\cdot e^{-iz} \\ 0 & -i\sin(x+y)\cdot e^{iz} & \cos(x+y)\cdot e^{iz} & 0 \\ 0 & \cos(x+y)\cdot e^{iz} & -i\sin(x+y)\cdot e^{iz} & 0 \\ -\sin(x-y)\cdot e^{-iz} & 0 & 0 & \cos(x-y)\cdot e^{-iz} \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) –
- **x** (*CalculatorFloat*) – The prefactor of the XX interaction.
- **y** (*CalculatorFloat*) – The prefactor of the YY interaction.
- **z** (*CalculatorFloat*) – The prefactor of the ZZ interaction.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

x()

Returns value of attribute x

y()

Returns value of attribute y

z()

Returns value of attribute z

class qoqo.operations.**RotateAroundSphericalAxis**

Implements a rotation around an axis in the x-y plane in spherical coordinates.

$$U = \begin{pmatrix} \cos(\frac{\theta}{2}) & 0 \\ 0 & \cos(\frac{\theta}{2}) \end{pmatrix} + \begin{pmatrix} -i\sin(\frac{\theta}{2})v_z & \sin(\frac{\theta}{2})(-iv_x - v_y) \\ \sin(\frac{\theta}{2})(-iv_x + v_y) & i\sin(\frac{\theta}{2})v_z \end{pmatrix}$$

with

$$\begin{aligned} v_x &= \sin(\theta_{sph})\cos(\phi_{sph}) \, , \\ v_y &= \sin(\theta_{sph})\sin(\phi_{sph}) \, , \\ v_z &= \cos(\theta_{sph}) \, . \end{aligned}$$

Parameters

- **qubit** (*int*) – The qubit the unitary gate is applied to.
- **theta** (*CalculatorFloat*) – The angle θ of the rotation.
- **spherical_theta** (*CalculatorFloat*) – The rotation axis, unit-vector spherical coordinates θ_{sph}.
- **spherical_phi** (*CalculatorFloat*) – The rotation axis, unit-vector spherical coordinates ϕ_{sph} gives the angle in the x-y plane.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate].(*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

spherical_phi()

Returns value of attribute spherical_phi

spherical_theta()

Returns value of attribute spherical_theta

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**RotateX**

The XPower gate $e^{-i\frac{\theta}{2}\sigma^x}$.

$$U = \begin{pmatrix} \cos(\frac{\theta}{2}) & 0 \\ 0 & \cos(\frac{\theta}{2}) \end{pmatrix} + \begin{pmatrix} 0 & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & 0 \end{pmatrix}$$

Parameters

- **qubit** (*int*) – The qubit the unitary gate is applied to.
- **theta** (*CalculatorFloat*) – The angle θ of the rotation.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i\cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i\cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`mul(other)`

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**RotateXY**

Implements a rotation around an axis in the x-y plane in spherical coordinates.

U = begin{pmatrix} cos(frac{theta}{2}) & -i e^{-i phi} sin(frac{theta}{2})\\ -i e^{i phi} sin(frac{theta}{2}) & cos(frac{theta}{2}) end{pmatrix}

Parameters

- **qubit** (*int*) – The qubit the unitary gate is applied to.
- **theta** (*CalculatorFloat*) – The angle θ of the rotation.
- **phi** (*CalculatorFloat*) – The rotation axis, in spherical coordinates ϕ_{sph} gives the angle in the x-y plane.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

phi()

Returns value of attribute phi

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`theta()`

Returns angle of rotation

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

`class qoqo.operations.RotateY`

The YPower gate $e^{-i\frac{\theta}{2}\sigma^y}$.

$$U = \begin{pmatrix} \cos(\frac{\theta}{2}) & 0 \\ 0 & \cos(\frac{\theta}{2}) \end{pmatrix} + \begin{pmatrix} 0 & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & 0 \end{pmatrix}$$

Parameters

- **qubit** (*int*) – The qubit the unitary gate is applied to.
- **theta** (*CalculatorFloat*) – The angle θ of the rotation.

`alpha_i()`

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`alpha_r()`

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_i()`

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`theta()`

Returns angle of rotation

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

`class qoqo.operations.RotateZ`

The ZPower gate $e^{-i\frac{\theta}{2}\sigma^z}$.

$$U = \begin{pmatrix} \cos(\frac{\theta}{2}) & 0 \\ 0 & \cos(\frac{\theta}{2}) \end{pmatrix} + \begin{pmatrix} -i\sin(\frac{\theta}{2}) & 0 \\ 0 & i\sin(\frac{\theta}{2}) \end{pmatrix}$$

Parameters

- **qubit** (*int*) – The qubit the unitary gate is applied to.
- **theta** (*CalculatorFloat*) – The angle θ of the rotation.

`alpha_i()`

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i\cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`alpha_r()`

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i\cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`theta()`

Returns angle of rotation

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

`class qoqo.operations.SGate`

The S gate.

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

Parameters
qubit (*int*) – The qubit the unitary gate is applied to.

`alpha_i()`

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`alpha_r()`

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

`class qoqo.operations.SWAP`

The controlled SWAP quantum operation.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.

`control()`

Returns control qubit of the two-qubit operation

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**SingleQubitGate**

The general single qubit unitary gate.

$$U = \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Parameters

- **qubit** – The qubit that the unitary gate is applied to.
- **alpha_r** – The real part of the on-diagonal elements of the single-qubit unitary.
- **alpha_i** – The imaginary part of the on-diagonal elements of the single-qubit unitary.
- **beta_r** – The real part of the off-diagonal elements of the single-qubit unitary.

- **beta_i** – The imaginary part of the off-diagonal elements of the single-qubit unitary.
- **global_phase** – The global phase of the single-qubit unitary.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**SpinInteraction**

The generalized, anisotropic XYZ Heisenberg interaction between spins.

$e^{-\mathrm{i}(x \cdot X_c X_t + y \cdot Y_c Y_t + z \cdot Z_c Z_t)}$

Where x, y, z are prefactors of the X_cX_t, Y_cY_t, Z_cZ_t Pauliproducts acting on control and target qubit, with $XX \equiv \sigma_x\sigma_x$, $YY \equiv \sigma_y\sigma_y$ and $ZZ \equiv \sigma_z\sigma_z$.

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) –
- **x** (*CalculatorFloat*) – The prefactor of the XX interaction.
- **y** (*CalculatorFloat*) – The prefactor of the YY interaction.
- **z** (*CalculatorFloat*) – The prefactor of the ZZ interaction.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target()`

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

x()

Returns value of attribute x

y()

Returns value of attribute y

z()

Returns value of attribute z

class qoqo.operations.**SqrtISwap**

The controlled square root ISwap quantum operation.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} & 0 \\ 0 & \frac{i}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**SqrtPauliX**

The square root of the XPower gate $e^{-i\frac{\pi}{4}\sigma^x}$.

$$U = \frac{1}{\sqrt{(2)}} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$$

Parameters
qubit (*int*) – The qubit the unitary gate is applied to.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_i()

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

beta_r()

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

global_phase()

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**TGate**

The T gate.

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$$

Parameters
qubit (*int*) – The qubit the unitary gate is applied to.

alpha_i()

Return the property alpha_i α_i of a unitary gate acting on one qubit

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

alpha_r()

Return the property alpha_r α_r of a unitary gate acting on one qubit

Here alpha_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_i()`

Returns the property beta_i β_i of a unitary gate acting on one qubit

Here beta_i is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`beta_r()`

Return the property beta_r β_r of a unitary gate acting on one qubit

Here beta_r is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`global_phase()`

Return the global phase g of a unitary gate acting on one qubit

Here global_phase is defined by

$$U = e^{i \cdot g} \begin{pmatrix} \alpha_r + i\alpha_i & -\beta_r + i\beta_i \\ \beta_r + i\beta_i & \alpha_r - i\alpha_i \end{pmatrix}$$

Returns
CalculatorFloat

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

mul(*other*)

Multiplies two compatible operations implementing OperateSingleQubitGate.

Does not consume the two operations being multiplied. Only Operations

Parameters
[OperateSingleQubitGate]. (*other - An Operation implementing*) –

Returns
Result of the multiplication, i.e. the multiplied single qubit gate.

Return type
PyResult

Example: ``` from qoqo.operations import RotateZ, RotateX

gate1 = RotateZ(qubit=0, theta=1) gate2 = RotateX(qubit=0, theta=1) multiplied = gate1.mul(gate2) print("Multiplied gate: ", multiplied) ```

qubit()

Return the qubit the operation acts on

Returns
int

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`unitary_matrix()`

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

`class qoqo.operations.Toffoli`

Implements Toffoli gate.

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Parameters

- **control_0** (*int*) – The index of the most significant qubit in the unitary representation. Here, the first controlling qubit of the operation.
- **control_1** (*int*) – The index of the second most significant qubit in the unitary representation. Here, the second controlling qubit of the operation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. Here, the qubit the PauliX gate is applied to.

`circuit()`

Returns circuit implementing the ThreeQubitGateOperation

Returns
Circuit

`control_0()`

Returns control_0 qubit of the three-qubit operation

`control_1()`

Returns control_1 qubit of the three-qubit operation

`hqslang()`

Returns hqslang name of Operation

Returns
The name

Return type
str

`involved_qubits()`

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

`is_parametrized()`

Returns true if operation contains symbolic parameters

Returns
bool

`remap_qubits(mapping)`

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters(substitution_parameters)`

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags()`

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target()`

Returns target qubit of the three-qubit operation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**VariableMSXX**

The variable-angle MolmerSorensen XX gate.

$$U = \begin{pmatrix} \cos(\theta/2) & 0 & 0 & -i\sin(\theta/2) \\ 0 & \cos(\theta/2) & -i\sin(\theta/2) & 0 \\ 0 & -i\sin(\theta/2) & \cos(\theta/2) & 0 \\ -i\sin(\theta/2) & 0 & 0 & \cos(\theta/2) \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation. The gate is symmetric under the exchange of qubits.
- **target** (*int*) – The index of the least significant qubit in the unitary representation. The gate is symmetric under the exchange of qubits.
- **theta** (*CalculatorFloat*) – The rotation angle θ.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

`remap_qubits`(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

`substitute_parameters`(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

`tags`()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

`target`()

Returns target qubit of the two-qubit operation

`theta`()

Returns angle of rotation

`unitary_matrix`()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

class qoqo.operations.**XY**

The controlled XY quantum operation

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta/2) & isin(\theta/2) & 0 \\ 0 & isin(\theta/2) & cos(\theta/2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Parameters

- **control** (*int*) – The index of the most significant qubit in the unitary representation.
- **target** (*int*) – The index of the least significant qubit in the unitary representation.
- **theta** (*CalculatorFloat*) – The rotation angle θ.

control()

Returns control qubit of the two-qubit operation

hqslang()

Returns hqslang name of Operation

Returns
The name

Return type
str

involved_qubits()

List all involved Qubits

Returns
The involved qubits as a set or 'ALL' if all qubits are involved

Return type
Union[set[int], str]

is_parametrized()

Returns true if operation contains symbolic parameters

Returns
bool

powercf(*power*)

Returns Rotated gate raised to power

Parameters
power (*CalculatorFloat*) – exponent of the power operation.

Returns
gate raised to the power of *power*

Return type
Self

remap_qubits(*mapping*)

Remap qubits

Parameters
mapping (*dict[int, int]*) – The mapping

Returns
The operation with the remapped qubits

Return type
Operation

Raises
RuntimeError – Qubit remapping failed

substitute_parameters(*substitution_parameters*)

Substitutes internal symbolic parameters with float values

Only available when all symbolic expressions can be evaluated to float with the provided parameters.

Parameters
substitution_parameters (*dict[str, float]*) – The substituted free parameters

Returns
The operation with the parameters substituted

Return type
Operation

Raises
RuntimeError – Parameter Substitution failed

tags()

Returns tags identifying the Operation

Returns
The tags identifying the operation

Return type
list[str]

target()

Returns target qubit of the two-qubit operation

theta()

Returns angle of rotation

unitary_matrix()

Return unitary matrix of gate.

Returns
np.ndarray

Raises
ValueError – Error symbolic operation cannot return float unitary matrix

1.1.3 qoqo.measurements

Measurements

Classes

Cheated	Collected information for executing a cheated measurement.
CheatedInput	Provides Necessary Information to run a cheated measurement.
CheatedPauliZProduct	Collected information for executing a cheated measurement of PauliZ product.
CheatedPauliZProductInput	Collected information for executing a cheated basis rotation measurement.
ClassicalRegister	Collected information for executing a classical register.
PauliZProduct	Collected information for executing a measurement of PauliZ product.
PauliZProductInput	Provides Necessary Information to run a [roqoqo::measurements::PauliZProduct] measurement.

class qoqo.measurements.**Cheated**

Collected information for executing a cheated measurement.

circuits()

Return the collection of quantum circuits for the separate cheated measurements.

Returns
The quantum circuits.

Return type
list[*Circuit*]

constant_circuit()

Returns constant circuit that is executed before any Circuit in circuits.

Returns
The constant Circuit (None if not defined).

Return type
Optional[*Circuit*]

evaluate(*input_bit_registers*, *float_registers*, *complex_registers*)

Execute the cheated measurement.

Parameters

- **input_bit_registers** (*dict[str, Union[list[list[int]], list[list[bool]]]]*) – The classical bit registers with the register name as key.
- **float_registers** (*dict[str, list[list[float]]*) – The classical float registers as a dictionary with the register name as key.
- **complex_registers** (*dict[str, list[list[complex]]*) – The classical complex registers as a dictionary with the register name as key.

Returns
The evaluated expectation values.

Return type
Optional[dict[str, float]

Raises

- **RuntimeError** – Unexpected repetition of key in bit_register.
- **RuntimeError** – Error evaluating cheated measurement.

static from_bincode(*input*)

Convert the bincode representation of the Cheated to a Cheated using the [bincode] crate.

Parameters
input (*ByteArray*) – The serialized Cheated (in [bincode] form).

Returns
The deserialized Cheated.

Return type
Cheated

Raises

- **TypeError** – Input cannot be converted to byte array.
- **ValueError** – Input cannot be deserialized to Cheated.

static from_json(*json_string*)

Deserialize the Cheated measurement from json form.

Returns
The deserialized Cheated Measurement.

Return type
Cheated

Raises
RuntimeError – Cannot deserialize string to Cheated.

input()

Returns the measurement input data defining how to construct expectation values from measurements.

Returns
The input of Cheated measurement

Return type
CheatedInput

measurement_type()

Returns the type of the measurement in string form.

Returns
The type of the measurement.

Return type
str

substitute_parameters(*substituted_parameters*)

Return copy of Measurement with symbolic parameters replaced.

Parameters
substituted_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the Circuit.

Raises
RuntimeError – Error substituting symbolic parameters.

`to_bincode()`

Return the bincode representation of the Cheated using the [bincode] crate.

Returns
The serialized Cheated (in [bincode] form).

Return type
ByteArray

Raises
ValueError – Cannot serialize Cheated to bytes.

`to_json()`

Serialize the Cheated measurement to json form.

Returns
The serialized Cheated measurement.

Return type
str

Raises
RuntimeError – Unexpected error serializing Cheated.

`class qoqo.measurements.CheatedInput`

Provides Necessary Information to run a cheated measurement.

`add_operator_exp_val(name, operator, readout)`

Add operator based expectation value to measurement input.

Adds an expectation value that is defined by an operator on the Hilbert space.

Parameters

- **name** (*str*) – The name of the expectation value.
- **operator** (*list[(int, int, complex)]*) – The measured operator on the Hilbert space, given as a list of sparse matrix entries of the form (row, col, value).
- **readout** (*str*) – The mame of the readout register that contains the density matrix or satevector.

Raises
RuntimeError – Failed to add operator based expectation value.

`static from_bincode(input)`

Convert the bincode representation of the CheatedInput to a CheatedInput using the [bincode] crate.

Parameters
input (*ByteArray*) – The serialized CheatedInput (in [bincode] form).

Returns
The deserialized CheatedInput.

Return type
CheatedInput

Raises

- **TypeError** – Input cannot be converted to byte array.
- **ValueError** – Input cannot be deserialized to CheatedInput.

static from_json(*json_string*)

Deserialize the CheatedInput from json form.

Returns
The deserialized CheatedInput.

Return type
CheatedInput

Raises
PyRuntimeError – Cannot deserialize string to CheatedInput.

to_bincode()

Return the bincode representation of the CheatedInput using the [bincode] crate.

Returns
The serialized CheatedInput (in [bincode] form).

Return type
ByteArray

Raises
ValueError – Cannot serialize CheatedInput to bytes.

to_json()

Serialize the CheatedInput to json form.

Returns
The serialized CheatedInput.

Return type
str

Raises
PyRuntimeError – Unexpected error serializing CheatedInput.

class qoqo.measurements.**CheatedPauliZProduct**

Collected information for executing a cheated measurement of PauliZ product.

circuits()

Returns the collection of quantum circuits for the separate basis rotations.

Returns
The quantum circuits.

Return type
list[*Circuit*]

constant_circuit()

Returns constant circuit that is executed before any Circuit in circuits.

Returns
The constant Circuit (None if not defined).

Return type
Optional[*Circuit*]

evaluate(*input_bit_registers*, *float_registers*, *complex_registers*)

Executes the cheated PauliZ product measurement.

Parameters

- **input_bit_registers** (*dict[str, Union[list[list[int]], list[list[bool]]]]*) – The classical bit registers with the register name as key
- **float_registers** (*dict[str, list[list[float]]]*) – The classical float registers as a dictionary with the register name as key
- **complex_registers** (*dict[str, list[list[complex]]]*) – The classical complex registers as a dictionary with the register name as key

Returns
The evaluated measurement.

Return type
Optional[dict[str, float]]

Raises

- **RuntimeError** – Unexpected repetition of key in bit_register.
- **RuntimeError** – Error evaluating cheated PauliZ product measurement.

static from_bincode(*input*)

Convert the bincode representation of the CheatedPauliZProduct to a CheatedPauliZProduct using the [bincode] crate.

Parameters
input (*ByteArray*) – The serialized CheatedPauliZProduct (in [bincode] form).

Returns
The deserialized CheatedPauliZProduct.

Return type
CheatedPauliZProduct

Raises

- **TypeError** – Input cannot be converted to byte array.
- **ValueError** – Input cannot be deserialized to CheatedPauliZProduct.

static from_json(*json_string*)

Deserialize the CheatedPauliZProduct from json form using the [serde_json] crate.

Returns
the deserialized CheatedPauliZProduct.

Return type
CheatedPauliZProduct

Raises
RuntimeError – Cannot deserialize string to CheatedPauliZProduct.

input()

Returns the measurement input data defining how to construct expectation values from measurements.

Returns
The measurment input of CheatedPauliZProduct.

Return type
CheatedPauliZProductInput

measurement_type()

Returns the type of the measurement in string form.

Returns
The type of the measurement.

Return type
str

`substitute_parameters`(*substituted_parameters*)

Returns clone of Measurement with symbolic parameters replaced

Parameters
substituted_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the Circuit.

`to_bincode`()

Return the bincode representation of the CheatedPauliZProduct using the [bincode] crate.

Returns
The serialized CheatedPauliZProduct (in [bincode] form).

Return type
ByteArray

Raises
ValueError – Cannot serialize CheatedPauliZProduct to bytes.

`to_json`()

Serializes the CheatedPauliZProduct to json form using the [serde_json] crate.

Returns
The serialized CheatedPauliZProduct.

Return type
str

Raises
RuntimeError – Unexpected error serializing CheatedPauliZProduct.

`class qoqo.measurements.CheatedPauliZProductInput`

Collected information for executing a cheated basis rotation measurement.

`add_linear_exp_val`(*name*, *linear*)

Add linear definition of expectation value to measurement input.

Adds an expectation value that is defined by a linear combination of expectation values of Pauli products.

Parameters

- **name** (*str*) – The name of the expectation value.
- **linear** (*dict[int, float]*) – The linear combination of expectation values as a map between Pauli product index and coefficient.

Raises
RuntimeError – Failed to add linear expectation value.

`add_pauliz_product`(*readout*)

Add measured Pauli product to CheatedPauliZProductInput and returns index of Pauli product.

When the pauli product is already in the measurement input the function only returns its index.

Parameters
readout (*str*) – The name of the readout register containing the pauli_product expectation value.

Returns
The index of the added Pauli product in the list of all Pauli products.

Return type
int

`add_symbolic_exp_val`(*name*, *symbolic*)

Add symbolic definition of expectation value to measurement input.

Adds an expectation value that is defined by a symbolic combination of expectation values of Pauli products.

Parameters

- **name** (*str*) – The name of the expectation value.
- **symbolic** (*str*) – The symbolic expression for the expectation values given by [qoqo_calculator::CalculatorFloat].

Raises
RuntimeError – Failed to add symbolic expectation value.

The i-th PauliProducts are hardcoded as variables pauli_product_i in the string expression of CalculatorFloat.

`static from_bincode`(*input*)

Convert the bincode representation of the CheatedPauliZProductInput to a CheatedPauliZProductInput using the [bincode] crate.

Parameters
input (*ByteArray*) – The serialized CheatedPauliZProductInput (in [bincode] form).

Returns
The deserialized CheatedPauliZProductInput.

Return type
CheatedPauliZProductInput

Raises

- **TypeError** – Input cannot be converted to byte array.
- **ValueError** – Input cannot be deserialized to CheatedPauliZProductInput.

`static from_json`(*json_string*)

Deserialize the CheatedPauliZProductInput from json form.

Returns
The deserialized CheatedPauliZProductInput.

Return type
CheatedPauliZProductInput

Raises
PyRuntimeError – Cannot deserialize string to CheatedPauliZProductInput.

`to_bincode`()

Return the bincode representation of the CheatedPauliZProductInput using the [bincode] crate.

Returns
The serialized CheatedPauliZProductInput (in [bincode] form).

Return type
ByteArray

Raises
ValueError – Cannot serialize CheatedPauliZProductInput to bytes.

`to_json()`

Serialize the CheatedPauliZProductInput to json form.

Returns
The serialized CheatedPauliZProductInput.

Return type
str

Raises
PyRuntimeError – Unexpected error serializing CheatedPauliZProductInput.

`class qoqo.measurements.ClassicalRegister`

Collected information for executing a classical register.

`circuits()`

Return the collection of quantum circuits that make up the total measurement.

Returns
The quantum circuits.

Return type
list[*Circuit*]

`constant_circuit()`

Returns constant circuit that is executed before any Circuit in circuits.

Returns
The constant Circuit (None if not defined).

Return type
Optional[*Circuit*]

`static from_bincode(input)`

Convert the bincode representation of the ClassicalRegister to a ClassicalRegister using the [bincode] crate.

Parameters
input (*ByteArray*) – The serialized ClassicalRegister (in [bincode] form).

Returns
The deserialized ClassicalRegister.

Return type
ClassicalRegister

Raises

- **TypeError** – Input cannot be converted to byte array.
- **ValueError** – Input cannot be deserialized to ClassicalRegister.

`static from_json(json_string)`

Deserialize the ClassicalRegister measurement from json form.

Returns
The deserialized ClassicalRegister Measurement.

Return type
ClassicalRegister

Raises
PyRuntimeError – Cannot deserialize string to ClassicalRegister.

`measurement_type()`

Returns the type of the measurement in string form.

Returns
The type of the measurement.

Return type
str

`substitute_parameters(substituted_parameters)`

Return copy of Measurement with symbolic parameters replaced.

Parameters
substituted_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the Circuit.

Raises
RuntimeError – Error substituting symbolic parameters.

`to_bincode()`

Return the bincode representation of the ClassicalRegister using the [bincode] crate.

Returns
The serialized ClassicalRegister (in [bincode] form).

Return type
ByteArray

Raises
ValueError – Cannot serialize ClassicalRegister to bytes.

`to_json()`

Serialize the ClassicalRegister measurement to json form.

Returns
The serialized ClassicalRegister measurement.

Return type
str

Raises
PyRuntimeError – Unexpected error serializing ClassicalRegister.

`class qoqo.measurements.PauliZProduct`

Collected information for executing a measurement of PauliZ product.

`circuits()`

Return the collection of quantum circuits for the separate basis rotations.

Returns
The quantum circuits.

Return type
list[*Circuit*]

`constant_circuit()`

Return constant circuit that is executed before any Circuit in circuits.

Returns
The constant Circuit (None if not defined).

Return type
Optional[*Circuit*]

evaluate(*input_bit_registers*, *float_registers*, *complex_registers*)

Execute the PauliZ product measurement.

Parameters

- **input_bit_registers** (*dict[str, Union[list[list[int]], list[list[bool]]]]*) – The classical bit registers with the register name as key
- **float_registers** (*dict[str, list[list[float]]]*) – The classical float registers as a dictionary with the register name as key
- **complex_registers** (*dict[str, list[list[complex]]]*) – The classical complex registers as a dictionary with the register name as key

Returns
The evaluated measurement.

Return type
Optional[dict[str, float]]

Raises

- **RuntimeError** – Unexpected repetition of key in bit_register.
- **RuntimeError** – Error evaluating PauliZ product measurement.

static from_bincode(*input*)

Convert the bincode representation of the PauliZProduct to a PauliZProduct using the [bincode] crate.

Parameters
input (*ByteArray*) – The serialized PauliZProduct (in [bincode] form).

Returns
The deserialized PauliZProduct.

Return type
PauliZProduct

Raises

- **TypeError** – Input cannot be converted to byte array.
- **ValueError** – Input cannot be deserialized to PauliZProduct.

static from_json(*json_string*)

Deserialize the PauliZProduct from json form using the [serde_json] crate.

Returns
The deserialized PauliZProduct.

Return type
PauliZProduct

Raises
RuntimeError – Cannot deserialize string to PauliZProduct.

input()

Returns the measurement input data defining how to construct expectation values from measurements.

Returns
The measurment input of PauliZProduct.

Return type
PauliZProductInput

measurement_type()

Returns the type of the measurement in string form.

Returns
The type of the measurement.

Return type
str

substitute_parameters(*substituted_parameters*)

Return clone of Measurement with symbolic parameters replaced.

Parameters
substituted_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the Circuit.

to_bincode()

Return the bincode representation of the PauliZProduct using the [bincode] crate.

Returns
The serialized PauliZProduct (in [bincode] form).

Return type
ByteArray

Raises
ValueError – Cannot serialize PauliZProduct to bytes.

to_json()

Serialize the PauliZProduct to json form using the [serde_json] crate.

Returns
The serialized PauliZProduct.

Return type
str

Raises
RuntimeError – Unexpected error serializing PauliZProduct.

class qoqo.measurements.**PauliZProductInput**

Provides Necessary Information to run a [roqoqo::measurements::PauliZProduct] measurement.

add_linear_exp_val(*name*, *linear*)

Add linear definition of expectation value to measurement input.

Adds an expectation value that is defined by a linear combination of expectation values of Pauli products.

Parameters

- **name** (*str*) – The name of the expectation value.
- **linear** (*dict[int, float]*) – The linear combination of expectation values as a map between Pauli product index and coefficient.

Raises
RuntimeError – Failed to add linear expectation value.

add_pauliz_product(*readout*, *pauli_product_mask*)

Add measured Pauli product to PauliZProductInput and returns index of Pauli product.

When the pauli product is already in the measurement input the function only returns it index.

Parameters

- **readout** (*str*) – The name of the readout register the pauli_product is defined on.
- **pauli_product_mask** (*list[int]*) – List of the qubits involved in the Pauli produc measurement.

Returns
The index of the added Pauli product in the list of all Pauli products.

Return type
int

Raises
RuntimeError – Failed to add pauli product.

add_symbolic_exp_val(*name*, *symbolic*)

Add symbolic definition of expectation value to measurement input.

Adds an expectation value that is defined by a symbolic combination of expectation values of Pauli products.

Parameters

- **name** (*str*) – The name of the expectation value.
- **symbolic** (*str*) – The symbolic expression for the expectation values given by [qoqo_calculator::CalculatorFloat].

Raises
RuntimeError – Failed to add symbolic expectation value.

The i-th PauliProducts are hardcoded as variables pauli_product_i in the string expression of CalculatorFloat.

static from_bincode(*input*)

Convert the bincode representation of the PauliZProductInput to a PauliZProductInput using the [bincode] crate.

Parameters
input (*ByteArray*) – The serialized PauliZProductInput (in [bincode] form).

Returns
The deserialized PauliZProductInput.

Return type
PauliZProductInput

Raises

- **TypeError** – Input cannot be converted to byte array.
- **ValueError** – Input cannot be deserialized to PauliZProductInput.

static from_json(*json_string*)

Deserialize the PauliZProductInput from json form.

Returns
The deserialized PauliZProductInput.

Return type
PauliZProductInput

Raises
PyRuntimeError – Cannot deserialize string to PauliZProductInput.

`to_bincode()`

Return the bincode representation of the PauliZProductInput using the [bincode] crate.

Returns
The serialized PauliZProductInput (in [bincode] form).

Return type
ByteArray

Raises
ValueError – Cannot serialize PauliZProductInput to bytes.

`to_json()`

Serialize the PauliZProductInput to json form.

Returns
The serialized PauliZProductInput.

Return type
str

Raises
PyRuntimeError – Unexpected error serializing PauliZProductInput.

1.1.4 qoqo.QuantumProgram

`class qoqo.QuantumProgram`

Represents a quantum program evaluating measurements based on a one or more free float parameters.

The main use of QuantumProgram is to contain a Measurements implementing [crate::measurements::Measure] that measures expectation values or output registers of [crate::Circuit] quantum circuits that contain symbolic parameters. Circuit with symbolic parameters can not be simulated or executed on real hardware. The symbolic parameters need to be replaced with real floating point numbers first. A QuantumProgram contains a list of the free parameters (*input_parameter_names*) and can automatically replace the parameters with its *run* methods and return the result.

The QuantumProgram should correspond as closely as possible to a normal multi-parameter function in classical computing that can be called with a set of parameters and returns a result. It is the intended way to interface between normal program code and roqoqo based quantum programs.

`__init__()`

Methods

__init__()	
from_bincode(input)	Convert the bincode representation of the QuantumProgram to a QuantumProgram using the [bincode] crate.
from_json(input)	Convert the json representation of a QuantumProgram to a QuantumProgram.
input_parameter_names()	Returns the input_parameter_names attribute of the qoqo QuantumProgram.
measurement()	Returns the measurement attribute of the QuantumProgram as Python object.
run(backend)	Runs the QuantumProgram and returns expectation values.
run_registers(backend)	Runs the QuantumProgram and returns the classical registers of the quantum program.
to_bincode()	Return the bincode representation of the QuantumProgram using the [bincode] crate.
to_json()	Return the json representation of the QuantumProgram.

static from_bincode(*input*)

Convert the bincode representation of the QuantumProgram to a QuantumProgram using the [bincode] crate.

Parameters
input (*ByteArray*) – The serialized QuantumProgram (in [bincode] form).

Returns
The deserialized QuantumProgram.

Return type
QuantumProgram

Raises

- **TypeError** – Input cannot be converted to byte array.
- **ValueError** – Input cannot be deserialized to QuantumProgram.

static from_json(*input*)

Convert the json representation of a QuantumProgram to a QuantumProgram.

Parameters
input (*str*) – The serialized QuantumProgram in json form.

Returns
The deserialized QuantumProgram.

Return type
QuantumProgram

Raises
ValueError – Input cannot be deserialized to QuantumProgram.

input_parameter_names()

Returns the input_parameter_names attribute of the qoqo QuantumProgram.

Returns
List of input parameter names.

`measurement()`

Returns the measurement attribute of the QuantumProgram as Python object.

Returns
PyObject corresponding to the qoqo measurement type of the QuantumProgram, i.e. PauliZProduct, CheatedPauliZProduct, Cheated or ClassicalRegister.

`run(backend)`

Runs the QuantumProgram and returns expectation values.

Runs the quantum programm for a given set of parameters passed in the same order as the parameters listed in *input_parameter_names* and returns expectation values.

Parameters

- **backend** (*Backend*) – The backend the program is executed on.
- **parameters** (*Optional[List[float]]*) – List of float parameters of the function call in order of *input_parameter_names*

`run_registers(backend)`

Runs the QuantumProgram and returns the classical registers of the quantum program.

Runs the quantum programm for a given set of parameters passed in the same order as the parameters listed in *input_parameter_names* and returns the classical register output. The classical registers usually contain a record of measurement values for the repeated execution of a [crate::Circuit] quantum circuit for real quantum hardware or the readout of the statevector or the density matrix for simulators.

Parameters

- **backend** (*Backend*) – The backend the program is executed on.
- **parameters** (*Optional[List[float]]*) – List of float parameters of the function call in order of *input_parameter_names*

`to_bincode()`

Return the bincode representation of the QuantumProgram using the [bincode] crate.

Returns
The serialized QuantumProgram (in [bincode] form).

Return type
ByteArray

Raises
ValueError – Cannot serialize QuantumProgram to bytes.

`to_json()`

Return the json representation of the QuantumProgram.

Returns
The serialized form of QuantumProgram.

Return type
str

Raises
ValueError – Cannot serialize QuantumProgram to json.

Classes

Circuit	Circuit of Operations.
CircuitDag(node_number, edge_number, /)	Represents the Direct Acyclic Graph (DAG) of a Circuit.
QuantumProgram	Represents a quantum program evaluating measurements based on a one or more free float parameters.

class qoqo.**Circuit**

Circuit of Operations.

A quantum program is represented as a linear sequence of Operations.

add(*op*)

Add an Operation to Circuit.

Parameters
op (*Operation*) – The Operation to add to the Circuit.

count_occurences(*operations*)

Count the number of occurences of a set of operation tags in the circuit.

Parameters
operations (*list[str]*) – List of operation tags that should be counted.

Returns
The number of occurences of these operation tags.

Return type
int

definitions()

Return a list of definitions in the Circuit.

Definitions need to be unique.

Returns
A vector of the definitions in the Circuit.

Return type
list[Operation]

filter_by_tag(*tag*)

Return a list of operations with given tag.

Parameters
tag (*str*) – tag by which to filter operations.

Returns
A vector of the operations with the specified tag in the Circuit.

Return type
list[Operation]

static from_bincode(*input*)

Convert the bincode representation of the Circuit to a Circuit using the [bincode] crate.

Parameters
input (*ByteArray*) – The serialized Circuit (in [bincode] form).

Returns
The deserialized Circuit.

Return type
Circuit

Raises

- **TypeError** – Input cannot be converted to byte array.
- **ValueError** – Input cannot be deserialized to Circuit.

static from_json(*json_string*)

Convert the json representation of a Circuit to a Circuit.

Parameters
input (*str*) – The serialized Circuit in json form.

Returns
The deserialized Circuit.

Return type
Circuit

Raises
ValueError – Input cannot be deserialized to Circuit.

get(*index*)

Return a copy of the Operation at a certain index of the Circuit.

Parameters
index (*int*) – The index of the Operation to get in the Circuit.

Returns
The operation at the given index (if it exists).

Return type
Operation

Raises
IndexError – Index out of range.

get_operation_types()

Return a list of the hqslang names of all operations occuring in the circuit.

Returns
The operation types in the Circuit.

Return type
set[str]

get_slice()

Return the copy of a slice of the Circuit.

Parameters

- **start** (*Optional[int]*) – The starting index of the slice (inclusive).
- **stop** (*Optional[int]*) – The stopping index of the slice (exclusive).

Returns
The slice of the operations in the Circuit with the specified indices.

Return type
Circuit

Raises

- **IndexError** – Stop index smaller than start index.
- **IndexError** – Stop index out of range.
- **IndexError** – Start index out of range.

operations()

Return a list of all operations in the Circuit.

Returns
A vector of the operations in the Circuit.

Return type
list[Operation]

overrotate()

Return clone of the circuit with all overrotation Pragmas applied.

Returns
Circuit with the overrotation applied

Return type
Circuit

Raises
RuntimeError – Error applying PragmaOverrotation in circuit.

Example:

```
>>> circuit = Circuit()
>>> circuit += PragmaOverrotation("RotateY", [1,], 20.0, 30.0)
>>> circuit += RotateX(0, 0.0)
>>> circuit += RotateY(0, 1.0)
>>> circuit += RotateY(1, 2.0)
>>> circuit += RotateY(1, 3.0)
>>> circuit_overrotated = circuit.overrotate()
print(circuit)
print(circuit_overrotated)
```

remap_qubits(*mapping*)

Remap qubits in operations in clone of Circuit.

Parameters
mapping (*dict[int, int]*) – The dictionary containing the {qubit: qubit} mapping to use in the Circuit.

Returns
The Circuit with the qubits remapped.

Return type
self

Raises
RuntimeError – The qubit remapping failed.

substitute_parameters(*substitution_parameters*)

Substitute the symbolic parameters in a clone of the Circuit according to the substitution_parameters input.

Parameters
substitution_parameters (*dict[str, float]*) – The dictionary containing the substitutions to use in the Circuit.

Returns
The Circuit with the parameters substituted.

Return type
self

Raises
RuntimeError – The parameter substitution failed.

`to_bincode()`

Return the bincode representation of the Circuit using the [bincode] crate.

Returns
The serialized Circuit (in [bincode] form).

Return type
ByteArray

Raises
ValueError – Cannot serialize Circuit to bytes.

`to_json()`

Return the json representation of the Circuit.

Returns
The serialized form of Circuit.

Return type
str

Raises
ValueError – Cannot serialize Circuit to json.

`class qoqo.CircuitDag(node_number, edge_number, /)`

Represents the Direct Acyclic Graph (DAG) of a Circuit.

`add_to_back(op)`

Add an Operation to the back of the CircuitDag, if necessary.

Parameters
op (*Operation*) – The Operation to add to the back of the CircuitDag.

Raises
TypeError – The Python Object cannot be converted to Operation.

`add_to_front(op)`

Add an Operation to the front of the CircuitDag, if necessary.

Parameters
op (*Operation*) – The Operation to add to the front of the CircuitDag.

Raises
TypeError – The Python Object cannot be converted to Operation.

`blocking_predecessors(already_executed, to_be_executed)`

Checks which of the direct predecessors of an Operation in the CircuitDag blocks the execution.

Warning: This method can only be used to determine if an operation can be executed when *already_executed* is consistent. When the list *already_executed* is inconsistent (a n operation is reported as executed that could not have been executed yet) this method returning an empty vector does not imply that the *to_be_executed* operation can be executed.

Parameters

- **already_executed** (*list[int]*) – List of NodeIndices of Nodes that have already been executed in the Circuit.
- **to_be_executed** (*int*) – NodeIndex of the Operation that should be executed next.

Returns
List containing the sorted blocking elements.

Return type
list[int]

commuting_operations()

Returns the list of nodes of commuting operations in CircuitDag.

Returns
The list of nodes of commuting operations.

Return type
list[int]

execution_blocked(*already_executed*, *to_be_executed*)

Checks if executing an operation is blocked by any not-yet executed operation.

Parameters

- **already_executed** (*list[int]*) – List of NodeIndices of Nodes that have already been executed in the Circuit.
- **to_be_executed** (*int*) – NodeIndex of the operation that should be executed next.

Returns
List containing the sorted blocking elements.

Return type
list[int]

first_operation_involving_classical()

Returns a dictionary where a key is composed by the name and the size of the classical register and its value represents the first node that involves that register.

Returns
The dictionary of {(str, int), int} elements.

Return type
dict[(str, int), int]

first_operation_involving_qubit()

Returns a dictionary where a key represents a qubit and its value represents the first node that involves that qubit.

Returns
The dictionary of {qubit: node} elements.

Return type
dict[int, int]

first_parallel_block()

Returns a set containing the nodes in the first parallel block.

Returns
The set of nodes in the first parallel block.

Return type
set[int]

`static from_bincode(input)`

Convert the bincode representation of the CircuitDag to a CircuitDag using the [bincode] crate.

Parameters
input (*ByteArray*) – The serialized CircuitDag (in [bincode] form).

Returns
The deserialized CircuitDag.

Return type
CircuitDag

Raises

- **TypeError** – Input cannot be converted to byte array.
- **ValueError** – Input cannot be deserialized to CircuitDag.

`from_circuit()`

Create a CircuitDag from a given Circuit;

Parameters
circuit (Circuit) – The Circuit to build the new CircuitDag from.

Returns
The new CircuitDag.

Return type
self

`get(index)`

Given a NodeIndex, returns the Operation contained in the node of the CircuitDag.

Parameters
index (*int*) – The index of the node to get from the CircuitDag.

Returns
The Operation at the given index (if it exists).

Return type
Operation

Raises
IndexError – Index out of range.

`last_operation_involving_classical()`

Returns a dictionary where a key is composed by the name and the size of the classical register and its value represents the last node that involves that register.

Returns
The dictionary of {(str, int), int} elements.

Return type
dict[(str, int), int]

`last_operation_involving_qubit()`

Returns a dictionary where a key represents a qubit and its value represents the last node that involves that qubit.

Returns
The dictionary of {qubit: node} elements.

Return type
dict[int, int]

`last_parallel_block()`

Returns a set containing the nodes in the last parallel block.

Returns
The set of nodes in the last parallel block.

Return type
set[int]

`new_front_layer(already_executed, current_front_layer, to_be_executed)`

Returns a new front-layer after executing an operation from the current front layer.

Returns an error if operation to be executed is not in the current front layer.

Parameters

- **already_executed** (*list[int]*) – List of NodeIndices of Nodes that have already been executed in the Circuit.
- **current_front_layer** (*list[int]*) – List of NodeIndices in the current front layer ready to be executed if physically possible.
- **to_be_executed** (*int*) – NodeIndex of the operation that should be executed next.

`parallel_blocks()`

Returns an iterator over the possible parallel blocks in circuit that can be executed simultaneously

Returns an Iterator over Vectors of references to the NodeIndices in the parallel block as well as references to the Operation in the blocks

`successors(node)`

Returns the list of the successors of a given node in the CircuitDag.

`to_bincode()`

Return the bincode representation of the CircuitDag using the [bincode] crate.

Returns
The serialized CircuitDag (in [bincode] form).

Return type
ByteArray

Raises
ValueError – Cannot serialize CircuitDag to bytes.

`to_circuit()`

Transforms the CircuitDag into a Circuit.

`class qoqo.QuantumProgram`

Represents a quantum program evaluating measurements based on a one or more free float parameters.

The main use of QuantumProgram is to contain a Measurements implementing [crate::measurements::Measure] that measures expectation values or output registers of [crate::Circuit] quantum circuits that contain symbolic parameters. Circuit with symbolic parameters can not be simulated or executed on real hardware. The symbolic parameters need to be replaced with real floating point numbers first. A QuantumProgram contains a list of the

free parameters (*input_parameter_names*) and can automatically replace the parameters with its *run* methods and return the result.

The QuantumProgram should correspond as closely as possible to a normal multi-parameter function in classical computing that can be called with a set of parameters and returns a result. It is the intended way to interface between normal program code and roqoqo based quantum programs.

`static from_bincode(input)`

Convert the bincode representation of the QuantumProgram to a QuantumProgram using the [bincode] crate.

Parameters
input (*ByteArray*) – The serialized QuantumProgram (in [bincode] form).

Returns
The deserialized QuantumProgram.

Return type
QuantumProgram

Raises

- **TypeError** – Input cannot be converted to byte array.
- **ValueError** – Input cannot be deserialized to QuantumProgram.

`static from_json(input)`

Convert the json representation of a QuantumProgram to a QuantumProgram.

Parameters
input (*str*) – The serialized QuantumProgram in json form.

Returns
The deserialized QuantumProgram.

Return type
QuantumProgram

Raises
ValueError – Input cannot be deserialized to QuantumProgram.

`input_parameter_names()`

Returns the input_parameter_names attribute of the qoqo QuantumProgram.

Returns
List of input parameter names.

`measurement()`

Returns the measurement attribute of the QuantumProgram as Python object.

Returns
PyObject corresponding to the qoqo measurement type of the QuantumProgram, i.e. PauliZProduct, CheatedPauliZProduct, Cheated or ClassicalRegister.

`run(backend)`

Runs the QuantumProgram and returns expectation values.

Runs the quantum programm for a given set of parameters passed in the same order as the parameters listed in *input_parameter_names* and returns expectation values.

Parameters

- **backend** (*Backend*) – The backend the program is executed on.

- **parameters** (*Optional[List[float]]*) – List of float parameters of the function call in order of *input_parameter_names*

run_registers(*backend*)

Runs the QuantumProgram and returns the classical registers of the quantum program.

Runs the quantum programm for a given set of parameters passed in the same order as the parameters listed in *input_parameter_names* and returns the classical register output. The classical registers usually contain a record of measurement values for the repeated execution of a [crate::Circuit] quantum circuit for real quantum hardware or the readout of the statevector or the density matrix for simulators.

Parameters

- **backend** (*Backend*) – The backend the program is executed on.
- **parameters** (*Optional[List[float]]*) – List of float parameters of the function call in order of *input_parameter_names*

to_bincode()

Return the bincode representation of the QuantumProgram using the [bincode] crate.

Returns
The serialized QuantumProgram (in [bincode] form).

Return type
ByteArray

Raises
ValueError – Cannot serialize QuantumProgram to bytes.

to_json()

Return the json representation of the QuantumProgram.

Returns
The serialized form of QuantumProgram.

Return type
str

Raises
ValueError – Cannot serialize QuantumProgram to json.

CHAPTER

TWO

DOCUMENTATION INDEX

- genindex

PYTHON MODULE INDEX

q

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z