
qoqo Documentation

Release 0.0

HQS Quantum Simulations GmbH

Apr 28, 2023

CONTENTS:

1	qoqo documentation	3
1.1	qoqo	3
2	Documentation Index	347
	Python Module Index	349
	Index	351

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

Introduction to qoqo with a jupyter [notebook](#).

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.

<i>Circuit</i>	Circuit of Operations.
<i>operations</i>	Operations are the atomic instructions in any quantum program that can be represented by qoqo.
<i>measurements</i>	Measurements
<i>QuantumProgram</i>	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

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

Count the number of occurrences 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 occurrences 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 occurring 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 Pragas 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 (MeasureQubit) or on a simulation of a quantum computing program (PRAGMA measurement operations).

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

continues on next page

Table 1 – continued from previous page

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

continues on next page

Table 1 – continued from previous page

<i>ComplexPMInteraction</i>	The complex hopping gate.
<i>MultiQubitMS</i>	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

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

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

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

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

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

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.

__init__()

Methods

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

<code>__init__()</code>	
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>number_measurements()</code>	Get value of struct field number_measurements
<code>readout()</code>	Get value of struct field readout
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	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 $|0\rangle$ 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

<code>__init__()</code>	
<code>hqslang()</code>	Return hqslang name of the operation.
<code>involved_qubits()</code>	List all involved qubits (here, all).
<code>is_parametrized()</code>	Return true when the operation has symbolic parameters.
<code>remap_qubits(mapping)</code>	Remap qubits in a clone of the PRAGMA operation.
<code>statevector()</code>	Return the statevector.
<code>substitute_parameters(substitution_parameters)</code>	Substitute the symbolic parameters in a clone of the PRAGMA operation according to the substitution_parameters input.
<code>tags()</code>	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 $|0\rangle$ 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

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

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

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

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

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

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

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

<code>__init__()</code>	
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>qubits()</code>	Return list of qubits of the multi qubit operation in order of descending significance
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	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 $|1\rangle$ to $|0\rangle$ and corresponds to zero-temperature in a physical device where $|0\rangle$ is the ground state. With respect to the definition of the Pauli operator Z , $|0\rangle$ 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

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

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

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

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

<code>__init__()</code>	
<code>gate_time()</code>	Return the <i>gate_time</i> of the PRAGMA operation.
<code>hqslang()</code>	Return hqslang name of the operation.
<code>involved_qubits()</code>	List all involved qubits.
<code>is_parametrized()</code>	Return true when the operation has symbolic parameters.
<code>qubit()</code>	Return the qubit on which the PRAGMA operation is applied.
<code>rates()</code>	Return the rates of the PRAGMA operation.
<code>remap_qubits(mapping)</code>	Remap qubits in a clone of the PRAGMA operation.
<code>substitute_parameters(substitution_parameters)</code>	Substitute the symbolic parameters in a clone of the PRAGMA operation according to the input.
<code>superoperator()</code>	Return the superoperator of the PRAGMA operation.
<code>tags()</code>	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 containing the condition bool value.
- **condition_index** (*int*) –
 - The index in the bit register containing the condition bool value.
- **circuit** (*Circuit*) –
 - The circuit executed if the condition is met.

__init__()

Methods

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

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

<code>__init__()</code>	
<code>control()</code>	Returns control qubit of the two-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the two-qubit operation
<code>unitary_matrix()</code>	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

<code>__init__()</code>	
<code>control()</code>	Returns control qubit of the two-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the two-qubit operation
<code>unitary_matrix()</code>	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

<code>__init__()</code>	
<code>control()</code>	Returns control qubit of the two-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the two-qubit operation
<code>unitary_matrix()</code>	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

<code>__init__()</code>	
<code>control()</code>	Returns control qubit of the two-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the two-qubit operation
<code>unitary_matrix()</code>	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

<code>__init__()</code>	
<code>control()</code>	Returns control qubit of the two-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the two-qubit operation
<code>unitary_matrix()</code>	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

<code>__init__()</code>	
<code>control()</code>	Returns control qubit of the two-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the two-qubit operation
<code>unitary_matrix()</code>	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

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

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

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

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

<code>__init__()</code>	
<code>control()</code>	Returns control qubit of the two-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the two-qubit operation
<code>unitary_matrix()</code>	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

<code>__init__()</code>	
<code>control()</code>	Returns control qubit of the two-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the two-qubit operation
<code>unitary_matrix()</code>	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:

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\frac{\theta}{2}) - i \sin(\frac{\theta}{2}) & 0 \\ 0 & 0 & -i \sin(\frac{\theta}{2}) \cos(\frac{\theta}{2}) & 1 \end{pmatrix}$$

Parameters

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

`__init__()`

Methods

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

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\frac{\theta}{2}) - ie^{-i\phi} \sin(\frac{\theta}{2}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -ie^{-i\phi} \sin(\frac{\theta}{2}) \cos(\frac{\theta}{2}) \end{pmatrix}$$

Parameters

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

`__init__()`**Methods**

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

<code>__init__()</code>	
<code>circuit()</code>	Returns circuit implementing the ThreeQubitGate-Operation
<code>control_0()</code>	Returns control_0 qubit of the three-qubit operation
<code>control_1()</code>	Returns control_1 qubit of the three-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the three-qubit operation
<code>unitary_matrix()</code>	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

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

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

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 the PauliX gate is applied to.

`__init__()`

Methods

<code>__init__()</code>	
<code>circuit()</code>	Returns circuit implementing the ThreeQubitGate-Operation
<code>control_0()</code>	Returns control_0 qubit of the three-qubit operation
<code>control_1()</code>	Returns control_1 qubit of the three-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the three-qubit operation
<code>unitary_matrix()</code>	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

<code>__init__()</code>	
<code>control()</code>	Returns control qubit of the two-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the two-qubit operation
<code>unitary_matrix()</code>	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

<code>__init__()</code>	
<code>control()</code>	Returns control qubit of the two-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>powercf(power)</code>	Returns Rotated gate raised to power
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the two-qubit operation
<code>theta()</code>	Returns angle of rotation
<code>unitary_matrix()</code>	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: $e^{-i\theta(X_c Y_t - Y_c X_t)}$.

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

<code>__init__()</code>	
<code>control()</code>	Returns control qubit of the two-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>phi()</code>	Returns value of attribute phi
<code>powercf(power)</code>	Returns Rotated gate raised to power
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the two-qubit operation
<code>theta()</code>	Returns angle of rotation
<code>unitary_matrix()</code>	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: $e^{-i\theta(X_c Y_t - Y_c X_t)}$.

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

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

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

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

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

Where x, y, z are prefactors of the $X_c X_t$, $Y_c Y_t$, $Z_c Z_t$ Pauliproduts 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

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

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

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:

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

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

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

$$e^{-i\theta(X_c X_t + Y_c Y_t)} = e^{-i\theta(\sigma_c^+ \sigma_t^- + \sigma_c^- \sigma_t^+)}$$

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

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

$$e^{-i[Re(\theta) \cdot (X_c X_t + Y_c Y_t) - Im(\theta) \cdot (X_c Y_t - Y_c X_t)]}$$

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.

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

<code>__init__()</code>	
<code>control()</code>	Returns control qubit of the two-qubit operation
<code>hqslang()</code>	Returns hqslang name of Operation
<code>involved_qubits()</code>	List all involved Qubits
<code>is_parametrized()</code>	Returns true if operation contains symbolic parameters
<code>remap_qubits(mapping)</code>	Remap qubits
<code>substitute_parameters(substitution_parameters)</code>	Substitutes internal symbolic parameters with float values
<code>t_imag()</code>	Returns value of attribute t_imag
<code>t_real()</code>	Returns value of attribute t_real
<code>tags()</code>	Returns tags identifying the Operation
<code>target()</code>	Returns target qubit of the two-qubit operation
<code>unitary_matrix()</code>	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 $\exp(-i * \text{theta}/2 * X_{i0} * X_{i1} * \dots * X_{in})$.

__init__()

Methods

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

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

continues on next page

Table 2 – continued from previous page

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

continues on next page

Table 2 – continued from previous page

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

class qoqo.operations.Bogoliubov

The Bogoliubov DeGennes interaction gate.

$$e^{-i\text{Re}(\Delta)(X_c X_t - Y_c Y_t)/2 + i\text{Im}(\Delta)(X_c Y_t + Y_c X_t)/2}$$

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:

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

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 $\text{Re}(\Delta)$.
- **delta_imag** (*CalculatorFloat*) – The imaginary part of the complex Bogoliubov interaction strength $\text{Im}(\Delta)$.

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.

$$e^{-i[Re(\theta) \cdot (X_c X_t + Y_c Y_t) - Im(\theta) \cdot (X_c Y_t - Y_c X_t)]}$$

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.

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:

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\frac{\theta}{2}) - i \sin(\frac{\theta}{2}) & 0 \\ 0 & 0 & -i \sin(\frac{\theta}{2}) \cos(\frac{\theta}{2}) & 1 \end{pmatrix}$$

Parameters

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

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\frac{\theta}{2}) - ie^{-i\phi} \sin(\frac{\theta}{2}) & 0 \\ 0 & 0 & ie^{-i\phi} \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$$

Parameters

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

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: $e^{-i\theta(X_c Y_t - Y_c X_t)}$.

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: $e^{-i\theta(X_c Y_t - Y_c X_t)}$.

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 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 α_r of a unitary gate acting on one qubit

Here α_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 β_i of a unitary gate acting on one qubit

Here β_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 β_r of a unitary gate acting on one qubit

Here β_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 $\exp(-i * \text{theta}/2 * X_{i0} * X_{i1} * \dots * X_{in})$.

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 $\exp(-i * \theta/2 * Z_{i0} * Z_{i1} * \dots * Z_{in})$.

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.

$$e^{-i\theta(X_c X_t + Y_c Y_t)} = e^{-i\theta(\sigma_c^+ \sigma_t^- + \sigma_c^- \sigma_t^+)}$$

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 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 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 α_r of a unitary gate acting on one qubit

Here α_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 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 α_r of a unitary gate acting on one qubit

Here α_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 β_i of a unitary gate acting on one qubit

Here β_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 $|0\rangle$.

Rotation around Z-axis by an arbitrary angle θ (AC Stark shift of the state $|0\rangle$).

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

Parameters

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

alpha_i()

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

Here α_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 β_i of a unitary gate acting on one qubit

Here β_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 β_r of a unitary gate acting on one qubit

Here β_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 $|1\rangle$.

Rotation around Z-axis by an arbitrary angle θ (AC Stark shift of the state $|1\rangle$).

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

Parameters

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

alpha_i()

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

Here α_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 β_i of a unitary gate acting on one qubit

Here β_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 β_r of a unitary gate acting on one qubit

Here β_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 θ .
- **phi** (*CalculatorFloat*) – The single qubit phase ϕ .

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 containing the condition bool value.
- **condition_index** (*int*) –
 - The index in the bit register containing 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 $|I\rangle$ to $|0\rangle$ and corresponds to zero-temperature in a physical device where $|0\rangle$ is the ground state. With respect to the definition of the Pauli operator Z , $|0\rangle$ 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, so that 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 $|0\rangle$ 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 $|0\rangle$ 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.

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

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.

$U = \text{begin}\{\text{pmatrix}\} \cos(\frac{\theta}{2}) \& -i e^{-i \phi} \sin(\frac{\theta}{2}) \backslash -i e^{i \phi} \sin(\frac{\theta}{2}) \& \cos(\frac{\theta}{2}) \text{end}\{\text{pmatrix}\}$

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 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 α_r of a unitary gate acting on one qubit

Here α_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 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 α_r of a unitary gate acting on one qubit

Here α_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 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 α_r of a unitary gate acting on one qubit

Here α_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 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 α_r of a unitary gate acting on one qubit

Here α_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.

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

Where x, y, z are prefactors of the $X_c X_t$, $Y_c Y_t$, $Z_c Z_t$ Pauliproduts 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.

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

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 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

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

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

<i>Cheated</i>	Collected information for executing a cheated measurement.
<i>CheatedInput</i>	Provides Necessary Information to run a cheated measurement.
<i>CheatedPauliZProduct</i>	Collected information for executing a cheated measurement of PauliZ product.
<i>CheatedPauliZProductInput</i>	Collected information for executing a cheated basis rotation measurement.
<i>ClassicalRegister</i>	Collected information for executing a classical register.
<i>PauliZProduct</i>	Collected information for executing a measurement of PauliZ product.
<i>PauliZProductInput</i>	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 name 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 product 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

<code>__init__()</code>	
<code>from_bincode(input)</code>	Convert the bincode representation of the QuantumProgram to a QuantumProgram using the [bincode] crate.
<code>from_json(input)</code>	Convert the json representation of a QuantumProgram to a QuantumProgram.
<code>input_parameter_names()</code>	Returns the input_parameter_names attribute of the qoqo QuantumProgram.
<code>measurement()</code>	Returns the measurement attribute of the QuantumProgram as Python object.
<code>run(backend)</code>	Runs the QuantumProgram and returns expectation values.
<code>run_registers(backend)</code>	Runs the QuantumProgram and returns the classical registers of the quantum program.
<code>to_bincode()</code>	Return the bincode representation of the QuantumProgram using the [bincode] crate.
<code>to_json()</code>	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

<i>Circuit</i>	Circuit of Operations.
<i>CircuitDag</i> (node_number, edge_number, /)	Represents the Direct Acyclic Graph (DAG) of a Circuit.
<i>QuantumProgram</i>	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 occurring 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 Pragas 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 program 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 program 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.

DOCUMENTATION INDEX

- `genindex`

PYTHON MODULE INDEX

q

`qoqo`, [3](#)

`qoqo.measurements`, [321](#)

`qoqo.operations`, [8](#)

Symbols

- `__init__()` (*qoqo.Circuit* method), 3
- `__init__()` (*qoqo.QuantumProgram* method), 334
- `__init__()` (*qoqo.operations.Bogoliubov* method), 166
- `__init__()` (*qoqo.operations.CNOT* method), 114
- `__init__()` (*qoqo.operations.ComplexPMInteraction* method), 170
- `__init__()` (*qoqo.operations.ControlledControlledPauliZ* method), 141
- `__init__()` (*qoqo.operations.ControlledControlledPhaseShift* method), 144
- `__init__()` (*qoqo.operations.ControlledPauliY* method), 131
- `__init__()` (*qoqo.operations.ControlledPauliZ* method), 133
- `__init__()` (*qoqo.operations.ControlledPhaseShift* method), 129
- `__init__()` (*qoqo.operations.ControlledRotateX* method), 136
- `__init__()` (*qoqo.operations.ControlledRotateXY* method), 139
- `__init__()` (*qoqo.operations.DefinitionBit* method), 59
- `__init__()` (*qoqo.operations.DefinitionComplex* method), 63
- `__init__()` (*qoqo.operations.DefinitionFloat* method), 61
- `__init__()` (*qoqo.operations.DefinitionUsize* method), 57
- `__init__()` (*qoqo.operations.FSwap* method), 118
- `__init__()` (*qoqo.operations.Fsim* method), 161
- `__init__()` (*qoqo.operations.GivensRotation* method), 154
- `__init__()` (*qoqo.operations.GivensRotationLittleEndian* method), 156
- `__init__()` (*qoqo.operations.Hadamard* method), 47
- `__init__()` (*qoqo.operations.ISwap* method), 120
- `__init__()` (*qoqo.operations.InputSymbolic* method), 65
- `__init__()` (*qoqo.operations.InvSqrtISwap* method), 124
- `__init__()` (*qoqo.operations.InvSqrtPauliX* method), 43
- `__init__()` (*qoqo.operations.MeasureQubit* method), 66
- `__init__()` (*qoqo.operations.MolmerSorensenXX* method), 149
- `__init__()` (*qoqo.operations.MultiQubitMS* method), 172
- `__init__()` (*qoqo.operations.PMInteraction* method), 168
- `__init__()` (*qoqo.operations.PauliX* method), 36
- `__init__()` (*qoqo.operations.PauliY* method), 33
- `__init__()` (*qoqo.operations.PauliZ* method), 29
- `__init__()` (*qoqo.operations.PragmaActiveReset* method), 94
- `__init__()` (*qoqo.operations.PragmaBoostNoise* method), 87
- `__init__()` (*qoqo.operations.PragmaConditional* method), 110
- `__init__()` (*qoqo.operations.PragmaDamping* method), 98
- `__init__()` (*qoqo.operations.PragmaDephasing* method), 103
- `__init__()` (*qoqo.operations.PragmaDepolarising* method), 100
- `__init__()` (*qoqo.operations.PragmaGeneralNoise* method), 108
- `__init__()` (*qoqo.operations.PragmaGetDensityMatrix* method), 70
- `__init__()` (*qoqo.operations.PragmaGetOccupationProbability* method), 72
- `__init__()` (*qoqo.operations.PragmaGetPauliProduct* method), 74
- `__init__()` (*qoqo.operations.PragmaGetStateVector* method), 68
- `__init__()` (*qoqo.operations.PragmaGlobalPhase* method), 91
- `__init__()` (*qoqo.operations.PragmaLoop* method), 112
- `__init__()` (*qoqo.operations.PragmaOverrotation* method), 85
- `__init__()` (*qoqo.operations.PragmaRandomNoise* method), 105
- `__init__()` (*qoqo.operations.PragmaRepeatGate* method), 105

method), 83

`__init__()` (*qoqo.operations.PragmaRepeatedMeasurement* method), 76

`__init__()` (*qoqo.operations.PragmaSetDensityMatrix* method), 81

`__init__()` (*qoqo.operations.PragmaSetNumberOfMeasurements* method), 77

`__init__()` (*qoqo.operations.PragmaSetStateVector* method), 79

`__init__()` (*qoqo.operations.PragmaSleep* method), 93

`__init__()` (*qoqo.operations.PragmaStopDecompositionBlock* method), 96

`__init__()` (*qoqo.operations.PragmaStopParallelBlock* method), 89

`__init__()` (*qoqo.operations.Qsim* method), 159

`__init__()` (*qoqo.operations.RotateAroundSphericalAxis* method), 25

`__init__()` (*qoqo.operations.RotateX* method), 21

`__init__()` (*qoqo.operations.RotateY* method), 17

`__init__()` (*qoqo.operations.RotateZ* method), 13

`__init__()` (*qoqo.operations.SGate* method), 54

`__init__()` (*qoqo.operations.SWAP* method), 116

`__init__()` (*qoqo.operations.SingleQubitGate* method), 10

`__init__()` (*qoqo.operations.SpinInteraction* method), 164

`__init__()` (*qoqo.operations.SqrtISwap* method), 122

`__init__()` (*qoqo.operations.SqrtPauliX* method), 40

`__init__()` (*qoqo.operations.TGate* method), 50

`__init__()` (*qoqo.operations.Toffoli* method), 147

`__init__()` (*qoqo.operations.VariableMSXX* method), 151

`__init__()` (*qoqo.operations.XY* method), 126

A

`add()` (*qoqo.Circuit* method), 4, 337

`add_linear_exp_val()`
(*qoqo.measurements.CheatedPauliZProductInput* method), 327

`add_linear_exp_val()`
(*qoqo.measurements.PauliZProductInput* method), 332

`add_operator_exp_val()`
(*qoqo.measurements.CheatedInput* method), 324

`add_pauliz_product()`
(*qoqo.measurements.CheatedPauliZProductInput* method), 327

`add_pauliz_product()`
(*qoqo.measurements.PauliZProductInput* method), 333

`add_symbolic_exp_val()`
(*qoqo.measurements.CheatedPauliZProductInput* method), 328

`add_symbolic_exp_val()`
(*qoqo.measurements.PauliZProductInput* method), 333

`add_to_back()` (*qoqo.CircuitDag* method), 340

`add_to_front()` (*qoqo.CircuitDag* method), 340

`alpha_i()` (*qoqo.operations.Hadamard* method), 47, 209

`alpha_i()` (*qoqo.operations.InvSqrtPauliX* method), 44, 217

`alpha_i()` (*qoqo.operations.PauliX* method), 37, 228

`alpha_i()` (*qoqo.operations.PauliY* method), 33, 231

`alpha_i()` (*qoqo.operations.PauliZ* method), 30, 234

`alpha_i()` (*qoqo.operations.PhaseShiftState0* method), 237

`alpha_i()` (*qoqo.operations.PhaseShiftState1* method), 240

`alpha_i()` (*qoqo.operations.RotateAroundSphericalAxis* method), 26, 285

`alpha_i()` (*qoqo.operations.RotateX* method), 22, 288

`alpha_i()` (*qoqo.operations.RotateXY* method), 291

`alpha_i()` (*qoqo.operations.RotateY* method), 18, 294

`alpha_i()` (*qoqo.operations.RotateZ* method), 14, 297

`alpha_i()` (*qoqo.operations.SGate* method), 54, 300

`alpha_i()` (*qoqo.operations.SingleQubitGate* method), 10, 305

`alpha_i()` (*qoqo.operations.SqrtPauliX* method), 40, 310

`alpha_i()` (*qoqo.operations.TGate* method), 51, 313

`alpha_r()` (*qoqo.operations.Hadamard* method), 47, 209

`alpha_r()` (*qoqo.operations.InvSqrtPauliX* method), 44, 217

`alpha_r()` (*qoqo.operations.PauliX* method), 37, 228

`alpha_r()` (*qoqo.operations.PauliY* method), 33, 231

`alpha_r()` (*qoqo.operations.PauliZ* method), 30, 234

`alpha_r()` (*qoqo.operations.PhaseShiftState0* method), 237

`alpha_r()` (*qoqo.operations.PhaseShiftState1* method), 240

`alpha_r()` (*qoqo.operations.RotateAroundSphericalAxis* method), 26, 285

`alpha_r()` (*qoqo.operations.RotateX* method), 22, 288

`alpha_r()` (*qoqo.operations.RotateXY* method), 291

`alpha_r()` (*qoqo.operations.RotateY* method), 18, 294

`alpha_r()` (*qoqo.operations.RotateZ* method), 14, 297

`alpha_r()` (*qoqo.operations.SGate* method), 54, 300

`alpha_r()` (*qoqo.operations.SingleQubitGate* method), 11, 305

`alpha_r()` (*qoqo.operations.SqrtPauliX* method), 40, 311

`alpha_r()` (*qoqo.operations.TGate* method), 51, 313

`amplitude()` (*qoqo.operations.PragmaOverrotation* method), 86, 267

B

`beta_i()` (*qoqo.operations.Hadamard method*), 48, 209
`beta_i()` (*qoqo.operations.InvSqrtPauliX method*), 44, 218
`beta_i()` (*qoqo.operations.PauliX method*), 37, 229
`beta_i()` (*qoqo.operations.PauliY method*), 34, 231
`beta_i()` (*qoqo.operations.PauliZ method*), 30, 234
`beta_i()` (*qoqo.operations.PhaseShiftState0 method*), 237
`beta_i()` (*qoqo.operations.PhaseShiftState1 method*), 240
`beta_i()` (*qoqo.operations.RotateAroundSphericalAxis method*), 26, 285
`beta_i()` (*qoqo.operations.RotateX method*), 22, 288
`beta_i()` (*qoqo.operations.RotateXY method*), 291
`beta_i()` (*qoqo.operations.RotateY method*), 18, 295
`beta_i()` (*qoqo.operations.RotateZ method*), 14, 297
`beta_i()` (*qoqo.operations.SGate method*), 55, 300
`beta_i()` (*qoqo.operations.SingleQubitGate method*), 11, 305
`beta_i()` (*qoqo.operations.SqrtPauliX method*), 41, 311
`beta_i()` (*qoqo.operations.TGate method*), 51, 314
`beta_r()` (*qoqo.operations.Hadamard method*), 48, 209
`beta_r()` (*qoqo.operations.InvSqrtPauliX method*), 45, 218
`beta_r()` (*qoqo.operations.PauliX method*), 38, 229
`beta_r()` (*qoqo.operations.PauliY method*), 34, 232
`beta_r()` (*qoqo.operations.PauliZ method*), 31, 234
`beta_r()` (*qoqo.operations.PhaseShiftState0 method*), 237
`beta_r()` (*qoqo.operations.PhaseShiftState1 method*), 240
`beta_r()` (*qoqo.operations.RotateAroundSphericalAxis method*), 27, 285
`beta_r()` (*qoqo.operations.RotateX method*), 23, 289
`beta_r()` (*qoqo.operations.RotateXY method*), 292
`beta_r()` (*qoqo.operations.RotateY method*), 19, 295
`beta_r()` (*qoqo.operations.RotateZ method*), 15, 298
`beta_r()` (*qoqo.operations.SGate method*), 55, 301
`beta_r()` (*qoqo.operations.SingleQubitGate method*), 11, 305
`beta_r()` (*qoqo.operations.SqrtPauliX method*), 41, 311
`beta_r()` (*qoqo.operations.TGate method*), 52, 314
`blocking_predecessors()` (*qoqo.CircuitDag method*), 340
Bogoliubov (*class in qoqo.operations*), 166, 177

C

Cheated (*class in qoqo.measurements*), 322
CheatedInput (*class in qoqo.measurements*), 324
CheatedPauliZProduct (*class in qoqo.measurements*), 325
CheatedPauliZProductInput (*class in qoqo.measurements*), 327

Circuit (*class in qoqo*), 3, 337
`circuit()` (*qoqo.operations.ControlledControlledPauliZ method*), 142, 182
`circuit()` (*qoqo.operations.ControlledControlledPhaseShift method*), 144, 184
`circuit()` (*qoqo.operations.MultiQubitMS method*), 173, 223
`circuit()` (*qoqo.operations.MultiQubitZZ method*), 225
`circuit()` (*qoqo.operations.PragmaConditional method*), 111, 250
`circuit()` (*qoqo.operations.PragmaGetDensityMatrix method*), 70, 259
`circuit()` (*qoqo.operations.PragmaGetOccupationProbability method*), 72, 261
`circuit()` (*qoqo.operations.PragmaGetPauliProduct method*), 74, 262
`circuit()` (*qoqo.operations.PragmaGetStateVector method*), 69, 263
`circuit()` (*qoqo.operations.PragmaLoop method*), 113, 266
`circuit()` (*qoqo.operations.Toffoli method*), 147, 316
CircuitDag (*class in qoqo*), 340
`circuits()` (*qoqo.measurements.Cheated method*), 322
`circuits()` (*qoqo.measurements.CheatedPauliZProduct method*), 325
`circuits()` (*qoqo.measurements.ClassicalRegister method*), 329
`circuits()` (*qoqo.measurements.PauliZProduct method*), 330
ClassicalRegister (*class in qoqo.measurements*), 329
CNOT (*class in qoqo.operations*), 114, 179
`commuting_operations()` (*qoqo.CircuitDag method*), 341
ComplexPMInteraction (*class in qoqo.operations*), 170, 180
`condition_index()` (*qoqo.operations.PragmaConditional method*), 111, 251
`condition_register()` (*qoqo.operations.PragmaConditional method*), 111, 251
`constant_circuit()` (*qoqo.measurements.Cheated method*), 322
`constant_circuit()` (*qoqo.measurements.CheatedPauliZProduct method*), 325
`constant_circuit()` (*qoqo.measurements.ClassicalRegister method*), 329
`constant_circuit()` (*qoqo.measurements.PauliZProduct method*), 330
`control()` (*qoqo.operations.Bogoliubov method*), 166, 177
`control()` (*qoqo.operations.CNOT method*), 115, 179
`control()` (*qoqo.operations.ComplexPMInteraction method*), 171, 180
`control()` (*qoqo.operations.ControlledPauliY method*),

- 132, 186
- `control()` (*qoqo.operations.ControlledPauliZ* method), 134, 188
- `control()` (*qoqo.operations.ControlledPhaseShift* method), 129, 190
- `control()` (*qoqo.operations.ControlledRotateX* method), 136, 192
- `control()` (*qoqo.operations.ControlledRotateXY* method), 139, 194
- `control()` (*qoqo.operations.Fsim* method), 162, 203
- `control()` (*qoqo.operations.FSwap* method), 119, 201
- `control()` (*qoqo.operations.GivensRotation* method), 154, 205
- `control()` (*qoqo.operations.GivensRotationLittleEndian* method), 157, 207
- `control()` (*qoqo.operations.InvSqrtISwap* method), 125, 216
- `control()` (*qoqo.operations.ISwap* method), 121, 212
- `control()` (*qoqo.operations.MolmerSorensenXX* method), 150, 222
- `control()` (*qoqo.operations.PhaseShiftedControlledPhase* method), 243
- `control()` (*qoqo.operations.PhaseShiftedControlledZ* method), 245
- `control()` (*qoqo.operations.PMInteraction* method), 169, 227
- `control()` (*qoqo.operations.Qsim* method), 159, 283
- `control()` (*qoqo.operations.SpinInteraction* method), 164, 307
- `control()` (*qoqo.operations.SqrtISwap* method), 123, 309
- `control()` (*qoqo.operations.SWAP* method), 117, 303
- `control()` (*qoqo.operations.VariableMSXX* method), 152, 318
- `control()` (*qoqo.operations.XY* method), 127, 320
- `control_0()` (*qoqo.operations.ControlledControlledPauliZ* method), 142, 182
- `control_0()` (*qoqo.operations.ControlledControlledPhaseShift* method), 145, 184
- `control_0()` (*qoqo.operations.Toffoli* method), 147, 316
- `control_1()` (*qoqo.operations.ControlledControlledPauliZ* method), 142, 182
- `control_1()` (*qoqo.operations.ControlledControlledPhaseShift* method), 145, 184
- `control_1()` (*qoqo.operations.Toffoli* method), 148, 316
- `ControlledControlledPauliZ` (class in *qoqo.operations*), 141, 182
- `ControlledControlledPhaseShift` (class in *qoqo.operations*), 143, 184
- `ControlledPauliY` (class in *qoqo.operations*), 131, 186
- `ControlledPauliZ` (class in *qoqo.operations*), 133, 188
- `ControlledPhaseShift` (class in *qoqo.operations*), 129, 189
- `ControlledRotateX` (class in *qoqo.operations*), 136, 191
- `ControlledRotateXY` (class in *qoqo.operations*), 138, 193
- `count_occurrences()` (*qoqo.Circuit* method), 4, 337
- ## D
- `DefinitionBit` (class in *qoqo.operations*), 59, 195
- `DefinitionComplex` (class in *qoqo.operations*), 63, 197
- `DefinitionFloat` (class in *qoqo.operations*), 61, 198
- `definitions()` (*qoqo.Circuit* method), 4, 337
- `DefinitionUsize` (class in *qoqo.operations*), 57, 200
- `delta()` (*qoqo.operations.Fsim* method), 162, 203
- `delta_imag()` (*qoqo.operations.Bogoliubov* method), 166, 177
- `delta_real()` (*qoqo.operations.Bogoliubov* method), 167, 177
- `density_matrix()` (*qoqo.operations.PragmaSetDensityMatrix* method), 82, 273
- `dephasing_rate()` (*qoqo.operations.PragmaRandomNoise* method), 106, 269
- `depolarising_rate()` (*qoqo.operations.PragmaRandomNoise* method), 106, 269
- ## E
- `evaluate()` (*qoqo.measurements.Cheated* method), 322
- `evaluate()` (*qoqo.measurements.CheatedPauliZProduct* method), 325
- `evaluate()` (*qoqo.measurements.PauliZProduct* method), 331
- `execution_blocked()` (*qoqo.CircuitDag* method), 341
- `execution_time()` (*qoqo.operations.PragmaStopParallelBlock* method), 89, 281
- ## F
- `filter_by_tag()` (*qoqo.Circuit* method), 5, 337
- `first_operation_involving_classical()` (*qoqo.CircuitDag* method), 341
- `first_operation_involving_qubit()` (*qoqo.CircuitDag* method), 341
- `first_parallel_block()` (*qoqo.CircuitDag* method), 341
- `from_bincode()` (*qoqo.Circuit* static method), 5, 337
- `from_bincode()` (*qoqo.CircuitDag* static method), 342
- `from_bincode()` (*qoqo.measurements.Cheated* static method), 323
- `from_bincode()` (*qoqo.measurements.CheatedInput* static method), 324
- `from_bincode()` (*qoqo.measurements.CheatedPauliZProduct* static method), 326
- `from_bincode()` (*qoqo.measurements.CheatedPauliZProductInput* static method), 328
- `from_bincode()` (*qoqo.measurements.ClassicalRegister* static method), 329

[from_bincode\(\)](#) (*qoqo.measurements.PauliZProduct static method*), 331
[from_bincode\(\)](#) (*qoqo.measurements.PauliZProductInput static method*), 333
[from_bincode\(\)](#) (*qoqo.QuantumProgram static method*), 335, 344
[from_circuit\(\)](#) (*qoqo.CircuitDag method*), 342
[from_json\(\)](#) (*qoqo.Circuit static method*), 5, 338
[from_json\(\)](#) (*qoqo.measurements.Cheated static method*), 323
[from_json\(\)](#) (*qoqo.measurements.CheatedInput static method*), 324
[from_json\(\)](#) (*qoqo.measurements.CheatedPauliZProduct static method*), 326
[from_json\(\)](#) (*qoqo.measurements.CheatedPauliZProductInput static method*), 328
[from_json\(\)](#) (*qoqo.measurements.ClassicalRegister static method*), 329
[from_json\(\)](#) (*qoqo.measurements.PauliZProduct static method*), 331
[from_json\(\)](#) (*qoqo.measurements.PauliZProductInput static method*), 333
[from_json\(\)](#) (*qoqo.QuantumProgram static method*), 335, 344
[Fsim](#) (*class in qoqo.operations*), 161, 203
[FSwap](#) (*class in qoqo.operations*), 118, 201

G

[gate_hqslang\(\)](#) (*qoqo.operations.PragmaOverrotation method*), 86, 267
[gate_time\(\)](#) (*qoqo.operations.PragmaDamping method*), 98, 252
[gate_time\(\)](#) (*qoqo.operations.PragmaDephasing method*), 103, 254
[gate_time\(\)](#) (*qoqo.operations.PragmaDepolarising method*), 101, 256
[gate_time\(\)](#) (*qoqo.operations.PragmaGeneralNoise method*), 108, 257
[gate_time\(\)](#) (*qoqo.operations.PragmaRandomNoise method*), 106, 269
[get\(\)](#) (*qoqo.Circuit method*), 5, 338
[get\(\)](#) (*qoqo.CircuitDag method*), 342
[get_operation_types\(\)](#) (*qoqo.Circuit method*), 5, 338
[get_slice\(\)](#) (*qoqo.Circuit method*), 6, 338
[GivensRotation](#) (*class in qoqo.operations*), 154, 204
[GivensRotationLittleEndian](#) (*class in qoqo.operations*), 156, 206
[global_phase\(\)](#) (*qoqo.operations.Hadamard method*), 48, 209
[global_phase\(\)](#) (*qoqo.operations.InvSqrtPauliX method*), 45, 218
[global_phase\(\)](#) (*qoqo.operations.PauliX method*), 38, 229

[global_phase\(\)](#) (*qoqo.operations.PauliY method*), 34, 232
[global_phase\(\)](#) (*qoqo.operations.PauliZ method*), 31, 235
[global_phase\(\)](#) (*qoqo.operations.PhaseShiftState0 method*), 237
[global_phase\(\)](#) (*qoqo.operations.PhaseShiftState1 method*), 240
[global_phase\(\)](#) (*qoqo.operations.RotateAroundSphericalAxis method*), 27, 286
[global_phase\(\)](#) (*qoqo.operations.RotateX method*), 23, 289
[global_phase\(\)](#) (*qoqo.operations.RotateXY method*), 292
[global_phase\(\)](#) (*qoqo.operations.RotateY method*), 19, 295
[global_phase\(\)](#) (*qoqo.operations.RotateZ method*), 15, 298
[global_phase\(\)](#) (*qoqo.operations.SGate method*), 55, 301
[global_phase\(\)](#) (*qoqo.operations.SingleQubitGate method*), 11, 305
[global_phase\(\)](#) (*qoqo.operations.SqrtPauliX method*), 41, 311
[global_phase\(\)](#) (*qoqo.operations.TGate method*), 52, 314

H

[Hadamard](#) (*class in qoqo.operations*), 47, 208
[hqslang\(\)](#) (*qoqo.operations.Bogoliubov method*), 167, 177
[hqslang\(\)](#) (*qoqo.operations.CNOT method*), 115, 179
[hqslang\(\)](#) (*qoqo.operations.ComplexPMInteraction method*), 171, 180
[hqslang\(\)](#) (*qoqo.operations.ControlledControlledPauliZ method*), 142, 183
[hqslang\(\)](#) (*qoqo.operations.ControlledControlledPhaseShift method*), 145, 185
[hqslang\(\)](#) (*qoqo.operations.ControlledPauliY method*), 132, 186
[hqslang\(\)](#) (*qoqo.operations.ControlledPauliZ method*), 134, 188
[hqslang\(\)](#) (*qoqo.operations.ControlledPhaseShift method*), 129, 190
[hqslang\(\)](#) (*qoqo.operations.ControlledRotateX method*), 137, 192
[hqslang\(\)](#) (*qoqo.operations.ControlledRotateXY method*), 139, 194
[hqslang\(\)](#) (*qoqo.operations.DefinitionBit method*), 59, 195
[hqslang\(\)](#) (*qoqo.operations.DefinitionComplex method*), 63, 197
[hqslang\(\)](#) (*qoqo.operations.DefinitionFloat method*), 61, 198

`hqslang()` (*qoqo.operations.DefinitionUsize method*), 58, 200
`hqslang()` (*qoqo.operations.Fsim method*), 162, 203
`hqslang()` (*qoqo.operations.FSwap method*), 119, 201
`hqslang()` (*qoqo.operations.GivensRotation method*), 154, 205
`hqslang()` (*qoqo.operations.GivensRotationLittleEndian method*), 157, 207
`hqslang()` (*qoqo.operations.Hadamard method*), 48, 210
`hqslang()` (*qoqo.operations.InputBit method*), 213
`hqslang()` (*qoqo.operations.InputSymbolic method*), 65, 214
`hqslang()` (*qoqo.operations.InvSqrtISwap method*), 125, 216
`hqslang()` (*qoqo.operations.InvSqrtPauliX method*), 45, 218
`hqslang()` (*qoqo.operations.ISwap method*), 121, 212
`hqslang()` (*qoqo.operations.MeasureQubit method*), 67, 220
`hqslang()` (*qoqo.operations.MolmerSorensenXX method*), 150, 222
`hqslang()` (*qoqo.operations.MultiQubitMS method*), 173, 223
`hqslang()` (*qoqo.operations.MultiQubitZZ method*), 225
`hqslang()` (*qoqo.operations.PauliX method*), 38, 229
`hqslang()` (*qoqo.operations.PauliY method*), 34, 232
`hqslang()` (*qoqo.operations.PauliZ method*), 31, 235
`hqslang()` (*qoqo.operations.PhaseShiftedControlledPhase method*), 243
`hqslang()` (*qoqo.operations.PhaseShiftedControlledZ method*), 245
`hqslang()` (*qoqo.operations.PhaseShiftState0 method*), 238
`hqslang()` (*qoqo.operations.PhaseShiftState1 method*), 241
`hqslang()` (*qoqo.operations.PMInteraction method*), 169, 227
`hqslang()` (*qoqo.operations.PragmaActiveReset method*), 95, 246
`hqslang()` (*qoqo.operations.PragmaBoostNoise method*), 88, 248
`hqslang()` (*qoqo.operations.PragmaChangeDevice method*), 249
`hqslang()` (*qoqo.operations.PragmaConditional method*), 111, 251
`hqslang()` (*qoqo.operations.PragmaDamping method*), 98, 252
`hqslang()` (*qoqo.operations.PragmaDephasing method*), 103, 254
`hqslang()` (*qoqo.operations.PragmaDepolarising method*), 101, 256
`hqslang()` (*qoqo.operations.PragmaGeneralNoise method*), 108, 257
`hqslang()` (*qoqo.operations.PragmaGetDensityMatrix method*), 71, 259
`hqslang()` (*qoqo.operations.PragmaGetOccupationProbability method*), 72, 261
`hqslang()` (*qoqo.operations.PragmaGetPauliProduct method*), 74, 262
`hqslang()` (*qoqo.operations.PragmaGetStateVector method*), 69, 263
`hqslang()` (*qoqo.operations.PragmaGlobalPhase method*), 91, 265
`hqslang()` (*qoqo.operations.PragmaLoop method*), 113, 266
`hqslang()` (*qoqo.operations.PragmaOverrotation method*), 86, 267
`hqslang()` (*qoqo.operations.PragmaRandomNoise method*), 106, 269
`hqslang()` (*qoqo.operations.PragmaRepeatedMeasurement method*), 76, 272
`hqslang()` (*qoqo.operations.PragmaRepeatGate method*), 84, 271
`hqslang()` (*qoqo.operations.PragmaSetDensityMatrix method*), 82, 273
`hqslang()` (*qoqo.operations.PragmaSetNumberOfMeasurements method*), 78, 275
`hqslang()` (*qoqo.operations.PragmaSetStateVector method*), 80, 276
`hqslang()` (*qoqo.operations.PragmaSleep method*), 93, 278
`hqslang()` (*qoqo.operations.PragmaStartDecompositionBlock method*), 279
`hqslang()` (*qoqo.operations.PragmaStopDecompositionBlock method*), 96, 280
`hqslang()` (*qoqo.operations.PragmaStopParallelBlock method*), 90, 281
`hqslang()` (*qoqo.operations.Qsim method*), 159, 283
`hqslang()` (*qoqo.operations.RotateAroundSphericalAxis method*), 27, 286
`hqslang()` (*qoqo.operations.RotateX method*), 23, 289
`hqslang()` (*qoqo.operations.RotateXY method*), 292
`hqslang()` (*qoqo.operations.RotateY method*), 19, 295
`hqslang()` (*qoqo.operations.RotateZ method*), 15, 298
`hqslang()` (*qoqo.operations.SGate method*), 55, 301
`hqslang()` (*qoqo.operations.SingleQubitGate method*), 11, 305
`hqslang()` (*qoqo.operations.SpinInteraction method*), 164, 307
`hqslang()` (*qoqo.operations.SqrtISwap method*), 123, 309
`hqslang()` (*qoqo.operations.SqrtPauliX method*), 41, 311
`hqslang()` (*qoqo.operations.SWAP method*), 117, 303
`hqslang()` (*qoqo.operations.TGate method*), 52, 314
`hqslang()` (*qoqo.operations.Toffoli method*), 148, 316
`hqslang()` (*qoqo.operations.VariableMSXX method*),

152, 318
 hqslang() (*qoqo.operations.XY method*), 127, 320

I

index() (*qoqo.operations.InputBit method*), 213
 input() (*qoqo.measurements.Cheated method*), 323
 input() (*qoqo.measurements.CheatedPauliZProduct method*), 326
 input() (*qoqo.measurements.PauliZProduct method*), 331
 input() (*qoqo.operations.InputSymbolic method*), 65, 214
 input_parameter_names() (*qoqo.QuantumProgram method*), 335, 344
 InputBit (class in *qoqo.operations*), 213
 InputSymbolic (class in *qoqo.operations*), 65, 214
 involved_qubits() (*qoqo.operations.Bogoliubov method*), 167, 177
 involved_qubits() (*qoqo.operations.CNOT method*), 115, 179
 involved_qubits() (*qoqo.operations.ComplexPMInteraction method*), 171, 181
 involved_qubits() (*qoqo.operations.ControlledControlledPauliZ method*), 142, 183
 involved_qubits() (*qoqo.operations.ControlledControlledPauliX method*), 145, 185
 involved_qubits() (*qoqo.operations.ControlledPauliY method*), 132, 187
 involved_qubits() (*qoqo.operations.ControlledPauliZ method*), 134, 188
 involved_qubits() (*qoqo.operations.ControlledPhaseShift method*), 129, 190
 involved_qubits() (*qoqo.operations.ControlledRotateX method*), 137, 192
 involved_qubits() (*qoqo.operations.ControlledRotateXY method*), 139, 194
 involved_qubits() (*qoqo.operations.DefinitionBit method*), 59, 196
 involved_qubits() (*qoqo.operations.DefinitionComplex method*), 63, 197
 involved_qubits() (*qoqo.operations.DefinitionFloat method*), 61, 198
 involved_qubits() (*qoqo.operations.DefinitionUsize method*), 58, 200
 involved_qubits() (*qoqo.operations.Fsim method*), 162, 203
 involved_qubits() (*qoqo.operations.FSwap method*), 119, 201
 involved_qubits() (*qoqo.operations.GivensRotation method*), 155, 205
 involved_qubits() (*qoqo.operations.GivensRotationLittleEndian method*), 157, 207
 involved_qubits() (*qoqo.operations.Hadamard method*), 48, 210
 involved_qubits() (*qoqo.operations.InputBit method*), 213
 involved_qubits() (*qoqo.operations.InputSymbolic method*), 65, 215
 involved_qubits() (*qoqo.operations.InvSqrtISwap method*), 125, 216
 involved_qubits() (*qoqo.operations.InvSqrtPauliX method*), 45, 218
 involved_qubits() (*qoqo.operations.ISwap method*), 121, 212
 involved_qubits() (*qoqo.operations.MeasureQubit method*), 67, 220
 involved_qubits() (*qoqo.operations.MolmerSorensenXX method*), 150, 222
 involved_qubits() (*qoqo.operations.MultiQubitMS method*), 173, 223
 involved_qubits() (*qoqo.operations.MultiQubitZZ method*), 225
 involved_qubits() (*qoqo.operations.PauliX method*), 38, 229
 involved_qubits() (*qoqo.operations.PauliY method*), 34, 232
 involved_qubits() (*qoqo.operations.PauliZ method*), 31, 235
 involved_qubits() (*qoqo.operations.PhaseShiftedControlledPhase method*), 243
 involved_qubits() (*qoqo.operations.PhaseShiftedControlledZ method*), 245
 involved_qubits() (*qoqo.operations.PhaseShiftState0 method*), 238
 involved_qubits() (*qoqo.operations.PhaseShiftState1 method*), 241
 involved_qubits() (*qoqo.operations.PMInteraction method*), 169, 227
 involved_qubits() (*qoqo.operations.PragmaActiveReset method*), 95, 246
 involved_qubits() (*qoqo.operations.PragmaBoostNoise method*), 88, 248
 involved_qubits() (*qoqo.operations.PragmaChangeDevice method*), 249
 involved_qubits() (*qoqo.operations.PragmaConditional method*), 111, 251
 involved_qubits() (*qoqo.operations.PragmaDamping method*), 99, 252
 involved_qubits() (*qoqo.operations.PragmaDephasing method*), 103, 254
 involved_qubits() (*qoqo.operations.PragmaDepolarising method*), 101, 256
 involved_qubits() (*qoqo.operations.PragmaGeneralNoise method*), 109, 258
 involved_qubits() (*qoqo.operations.PragmaGetDensityMatrix method*), 71, 259
 involved_qubits() (*qoqo.operations.PragmaGetOccupationProbability method*), 72, 261

involved_qubits() (*qoqo.operations.PragmaGetPauliPrimal* method), 74, 262

involved_qubits() (*qoqo.operations.PragmaGetStateVecin* method), 69, 263

involved_qubits() (*qoqo.operations.PragmaGlobalPhase* method), 91, 265

involved_qubits() (*qoqo.operations.PragmaLoop* method), 113, 266

involved_qubits() (*qoqo.operations.PragmaOverrotation* method), 86, 268

involved_qubits() (*qoqo.operations.PragmaRandomNoise* method), 106, 269

involved_qubits() (*qoqo.operations.PragmaRepeatedMeasure* method), 76, 272

involved_qubits() (*qoqo.operations.PragmaRepeatGateis* method), 84, 271

involved_qubits() (*qoqo.operations.PragmaSetDensityMeasure* method), 82, 274

involved_qubits() (*qoqo.operations.PragmaSetNumberQSim* method), 78, 275

involved_qubits() (*qoqo.operations.PragmaSetStateVecin* method), 80, 276

involved_qubits() (*qoqo.operations.PragmaSleep* method), 93, 278

involved_qubits() (*qoqo.operations.PragmaStartDecomposition* method), 279

involved_qubits() (*qoqo.operations.PragmaStopDecomposition* method), 97, 280

involved_qubits() (*qoqo.operations.PragmaStopParallelIS* method), 90, 282

involved_qubits() (*qoqo.operations.Qsim* method), 160, 283

involved_qubits() (*qoqo.operations.RotateAroundSpheris* method), 27, 286

involved_qubits() (*qoqo.operations.RotateX* method), 23, 289

involved_qubits() (*qoqo.operations.RotateXY* method), 292

involved_qubits() (*qoqo.operations.RotateY* method), 19, 295

involved_qubits() (*qoqo.operations.RotateZ* method), 15, 298

involved_qubits() (*qoqo.operations.SGate* method), 55, 301

involved_qubits() (*qoqo.operations.SingleQubitGate* method), 12, 306

involved_qubits() (*qoqo.operations.SpinInteraction* method), 164, 308

involved_qubits() (*qoqo.operations.SqrtISwap* method), 123, 309

involved_qubits() (*qoqo.operations.SqrtPauliX* method), 41, 312

involved_qubits() (*qoqo.operations.SWAP* method), 117, 303

involved_qubits() (*qoqo.operations.TGate* method), 52, 314

involved_qubits() (*qoqo.operations.Toffoli* method), 148, 317

involved_qubits() (*qoqo.operations.VariableMSXX* method), 152, 318

involved_qubits() (*qoqo.operations.XY* method), 127, 320

InvSqrtISwap (class in *qoqo.operations*), 124, 216

InvSqrtPauliX (class in *qoqo.operations*), 43, 217

is_output() (*qoqo.operations.DefinitionBit* method), 60, 196

is_output() (*qoqo.operations.DefinitionComplex* method), 63, 197

is_output() (*qoqo.operations.DefinitionFloat* method), 61, 199

is_output() (*qoqo.operations.DefinitionUseize* method), 58, 200

is_parametrized() (*qoqo.operations.Bogoliubov* method), 167, 178

is_parametrized() (*qoqo.operations.CNOT* method), 115, 179

is_parametrized() (*qoqo.operations.ComplexPMInteraction* method), 171, 181

is_parametrized() (*qoqo.operations.ControlledControlledPauliZ* method), 142, 183

is_parametrized() (*qoqo.operations.ControlledControlledPhaseShift* method), 145, 185

is_parametrized() (*qoqo.operations.ControlledPauliY* method), 132, 187

is_parametrized() (*qoqo.operations.ControlledPauliZ* method), 134, 188

is_parametrized() (*qoqo.operations.ControlledPhaseShift* method), 130, 190

is_parametrized() (*qoqo.operations.ControlledRotateX* method), 137, 192

is_parametrized() (*qoqo.operations.ControlledRotateXY* method), 139, 194

is_parametrized() (*qoqo.operations.DefinitionBit* method), 60, 196

is_parametrized() (*qoqo.operations.DefinitionComplex* method), 63, 197

is_parametrized() (*qoqo.operations.DefinitionFloat* method), 62, 199

is_parametrized() (*qoqo.operations.DefinitionUseize* method), 58, 200

is_parametrized() (*qoqo.operations.Fsim* method), 162, 203

is_parametrized() (*qoqo.operations.FSwap* method), 119, 202

is_parametrized() (*qoqo.operations.GivensRotation* method), 155, 205

is_parametrized() (*qoqo.operations.GivensRotationLittleEndian* method), 157, 207

`is_parametrized()` (*qoqo.operations.Hadamard method*), 49, 210
`is_parametrized()` (*qoqo.operations.InputBit method*), 213
`is_parametrized()` (*qoqo.operations.InputSymbolic method*), 65, 215
`is_parametrized()` (*qoqo.operations.InvSqrtISwap method*), 125, 216
`is_parametrized()` (*qoqo.operations.InvSqrtPauliX method*), 45, 219
`is_parametrized()` (*qoqo.operations.ISwap method*), 121, 212
`is_parametrized()` (*qoqo.operations.MeasureQubit method*), 67, 220
`is_parametrized()` (*qoqo.operations.MolmerSorensenXXis method*), 150, 222
`is_parametrized()` (*qoqo.operations.MultiQubitMS method*), 173, 224
`is_parametrized()` (*qoqo.operations.MultiQubitZZ method*), 225
`is_parametrized()` (*qoqo.operations.PauliX method*), 38, 230
`is_parametrized()` (*qoqo.operations.PauliY method*), 35, 232
`is_parametrized()` (*qoqo.operations.PauliZ method*), 31, 235
`is_parametrized()` (*qoqo.operations.PhaseShiftedControlledPhase method*), 243
`is_parametrized()` (*qoqo.operations.PhaseShiftedControlledZ method*), 245
`is_parametrized()` (*qoqo.operations.PhaseShiftState0 method*), 238
`is_parametrized()` (*qoqo.operations.PhaseShiftState1 method*), 241
`is_parametrized()` (*qoqo.operations.PMInteraction method*), 169, 227
`is_parametrized()` (*qoqo.operations.PragmaActiveReset method*), 95, 247
`is_parametrized()` (*qoqo.operations.PragmaBoostNoise method*), 88, 248
`is_parametrized()` (*qoqo.operations.PragmaChangeDevice method*), 249
`is_parametrized()` (*qoqo.operations.PragmaConditionalis method*), 111, 251
`is_parametrized()` (*qoqo.operations.PragmaDamping method*), 99, 252
`is_parametrized()` (*qoqo.operations.PragmaDephasing method*), 104, 254
`is_parametrized()` (*qoqo.operations.PragmaDepolarising method*), 101, 256
`is_parametrized()` (*qoqo.operations.PragmaGeneralNoise method*), 109, 258
`is_parametrized()` (*qoqo.operations.PragmaGetDensityMatrix method*), 71, 260
`is_parametrized()` (*qoqo.operations.PragmaGetOccupationProbability method*), 73, 261
`is_parametrized()` (*qoqo.operations.PragmaGetPauliProduct method*), 75, 262
`is_parametrized()` (*qoqo.operations.PragmaGetStateVector method*), 69, 264
`is_parametrized()` (*qoqo.operations.PragmaGlobalPhase method*), 92, 265
`is_parametrized()` (*qoqo.operations.PragmaLoop method*), 113, 266
`is_parametrized()` (*qoqo.operations.PragmaOverrotation method*), 86, 268
`is_parametrized()` (*qoqo.operations.PragmaRandomNoise method*), 106, 269
`is_parametrized()` (*qoqo.operations.PragmaRepeatedMeasurement method*), 76, 272
`is_parametrized()` (*qoqo.operations.PragmaRepeatGate method*), 84, 271
`is_parametrized()` (*qoqo.operations.PragmaSetDensityMatrix method*), 82, 274
`is_parametrized()` (*qoqo.operations.PragmaSetNumberOfMeasurement method*), 78, 275
`is_parametrized()` (*qoqo.operations.PragmaSetStateVector method*), 80, 276
`is_parametrized()` (*qoqo.operations.PragmaSleep method*), 93, 278
`is_parametrized()` (*qoqo.operations.PragmaStartDecompositionBlock method*), 279
`is_parametrized()` (*qoqo.operations.PragmaStopDecompositionBlock method*), 97, 280
`is_parametrized()` (*qoqo.operations.PragmaStopParallelBlock method*), 90, 282
`is_parametrized()` (*qoqo.operations.Qsim method*), 160, 283
`is_parametrized()` (*qoqo.operations.RotateAroundSphericalAxis method*), 27, 286
`is_parametrized()` (*qoqo.operations.RotateX method*), 23, 289
`is_parametrized()` (*qoqo.operations.RotateXY method*), 292
`is_parametrized()` (*qoqo.operations.RotateY method*), 19, 295
`is_parametrized()` (*qoqo.operations.RotateZ method*), 15, 298
`is_parametrized()` (*qoqo.operations.SGate method*), 56, 301
`is_parametrized()` (*qoqo.operations.SingleQubitGate method*), 12, 306
`is_parametrized()` (*qoqo.operations.SpinInteraction method*), 164, 308
`is_parametrized()` (*qoqo.operations.SqrtISwap method*), 123, 309
`is_parametrized()` (*qoqo.operations.SqrtPauliX method*), 42, 312

`is_parametrized()` (*qoqo.operations.SWAP method*), 117, 303
`is_parametrized()` (*qoqo.operations.TGate method*), 52, 314
`is_parametrized()` (*qoqo.operations.Toffoli method*), 148, 317
`is_parametrized()` (*qoqo.operations.VariableMSXX method*), 152, 318
`is_parametrized()` (*qoqo.operations.XY method*), 127, 320
`ISwap` (class in *qoqo.operations*), 120, 211

L

`last_operation_involving_classical()` (*qoqo.CircuitDag method*), 342
`last_operation_involving_qubit()` (*qoqo.CircuitDag method*), 342
`last_parallel_block()` (*qoqo.CircuitDag method*), 343
`length()` (*qoqo.operations.DefinitionBit method*), 60, 196
`length()` (*qoqo.operations.DefinitionComplex method*), 64, 197
`length()` (*qoqo.operations.DefinitionFloat method*), 62, 199
`length()` (*qoqo.operations.DefinitionUsize method*), 58, 200

M

`measurement()` (*qoqo.QuantumProgram method*), 336, 344
`measurement_type()` (*qoqo.measurements.Cheated method*), 323
`measurement_type()` (*qoqo.measurements.CheatedPauliZProduct method*), 326
`measurement_type()` (*qoqo.measurements.ClassicalRegister method*), 330
`measurement_type()` (*qoqo.measurements.PauliZProduct method*), 332
`MeasureQubit` (class in *qoqo.operations*), 66, 220
`module`
 qoqo, 3
 qoqo.measurements, 321
 qoqo.operations, 8
`MolmerSorensenXX` (class in *qoqo.operations*), 149, 221
`mul()` (*qoqo.operations.Hadamard method*), 49, 210
`mul()` (*qoqo.operations.InvSqrtPauliX method*), 45, 219
`mul()` (*qoqo.operations.PauliX method*), 38, 230
`mul()` (*qoqo.operations.PauliY method*), 35, 232
`mul()` (*qoqo.operations.PauliZ method*), 31, 235
`mul()` (*qoqo.operations.PhaseShiftState0 method*), 238
`mul()` (*qoqo.operations.PhaseShiftState1 method*), 241
`mul()` (*qoqo.operations.RotateAroundSphericalAxis method*), 27, 286

`mul()` (*qoqo.operations.RotateX method*), 23, 289
`mul()` (*qoqo.operations.RotateXY method*), 292
`mul()` (*qoqo.operations.RotateY method*), 19, 295
`mul()` (*qoqo.operations.RotateZ method*), 15, 298
`mul()` (*qoqo.operations.SGate method*), 56, 301
`mul()` (*qoqo.operations.SingleQubitGate method*), 12, 306
`mul()` (*qoqo.operations.SqrtPauliX method*), 42, 312
`mul()` (*qoqo.operations.TGate method*), 52, 314
`MultiQubitMS` (class in *qoqo.operations*), 172, 223
`MultiQubitZZ` (class in *qoqo.operations*), 225

N

`name()` (*qoqo.operations.DefinitionBit method*), 60, 196
`name()` (*qoqo.operations.DefinitionComplex method*), 64, 197
`name()` (*qoqo.operations.DefinitionFloat method*), 62, 199
`name()` (*qoqo.operations.DefinitionUsize method*), 58, 200
`name()` (*qoqo.operations.InputBit method*), 213
`name()` (*qoqo.operations.InputSymbolic method*), 65, 215
`new_front_layer()` (*qoqo.CircuitDag method*), 343
`noise_coefficient()` (*qoqo.operations.PragmaBoostNoise method*), 88, 248
`number_measurements()` (*qoqo.operations.PragmaRepeatedMeasurement method*), 76, 272
`number_measurements()` (*qoqo.operations.PragmaSetNumberOfMeasurements method*), 78, 275

O

`operations()` (*qoqo.Circuit method*), 6, 339
`overrotate()` (*qoqo.Circuit method*), 6, 339

P

`parallel_blocks()` (*qoqo.CircuitDag method*), 343
`PauliX` (class in *qoqo.operations*), 36, 228
`PauliY` (class in *qoqo.operations*), 33, 231
`PauliZ` (class in *qoqo.operations*), 29, 234
`PauliZProduct` (class in *qoqo.measurements*), 330
`PauliZProductInput` (class in *qoqo.measurements*), 332
`phase()` (*qoqo.operations.PragmaGlobalPhase method*), 92, 265
`PhaseShiftedControlledPhase` (class in *qoqo.operations*), 242
`PhaseShiftedControlledZ` (class in *qoqo.operations*), 244
`PhaseShiftState0` (class in *qoqo.operations*), 236
`PhaseShiftState1` (class in *qoqo.operations*), 239

- `phi()` (*qoqo.operations.ControlledRotateXY* method), 140, 194
- `phi()` (*qoqo.operations.GivensRotation* method), 155, 205
- `phi()` (*qoqo.operations.GivensRotationLittleEndian* method), 157, 207
- `phi()` (*qoqo.operations.PhaseShiftedControlledPhase* method), 243
- `phi()` (*qoqo.operations.PhaseShiftedControlledZ* method), 245
- `phi()` (*qoqo.operations.RotateXY* method), 293
- `PMInteraction` (class in *qoqo.operations*), 168, 227
- `powercf()` (*qoqo.operations.ControlledControlledPhaseShift* method), 145, 185
- `powercf()` (*qoqo.operations.ControlledPhaseShift* method), 130, 190
- `powercf()` (*qoqo.operations.ControlledRotateX* method), 137, 192
- `powercf()` (*qoqo.operations.ControlledRotateXY* method), 140, 194
- `powercf()` (*qoqo.operations.GivensRotation* method), 155, 205
- `powercf()` (*qoqo.operations.GivensRotationLittleEndian* method), 157, 207
- `powercf()` (*qoqo.operations.MultiQubitMS* method), 173, 224
- `powercf()` (*qoqo.operations.MultiQubitZZ* method), 225
- `powercf()` (*qoqo.operations.PhaseShiftedControlledPhase* method), 243
- `powercf()` (*qoqo.operations.PhaseShiftState0* method), 238
- `powercf()` (*qoqo.operations.PhaseShiftState1* method), 241
- `powercf()` (*qoqo.operations.PragmaDamping* method), 99, 252
- `powercf()` (*qoqo.operations.PragmaDephasing* method), 104, 254
- `powercf()` (*qoqo.operations.PragmaDepolarising* method), 101, 256
- `powercf()` (*qoqo.operations.PragmaRandomNoise* method), 106, 269
- `powercf()` (*qoqo.operations.RotateAroundSphericalAxis* method), 28, 287
- `powercf()` (*qoqo.operations.RotateX* method), 24, 290
- `powercf()` (*qoqo.operations.RotateXY* method), 293
- `powercf()` (*qoqo.operations.RotateY* method), 20, 296
- `powercf()` (*qoqo.operations.RotateZ* method), 16, 299
- `powercf()` (*qoqo.operations.VariableMSXX* method), 152, 318
- `powercf()` (*qoqo.operations.XY* method), 127, 320
- `PragmaActiveReset` (class in *qoqo.operations*), 94, 246
- `PragmaBoostNoise` (class in *qoqo.operations*), 87, 247
- `PragmaChangeDevice` (class in *qoqo.operations*), 249
- `PragmaConditional` (class in *qoqo.operations*), 110, 250
- `PragmaDamping` (class in *qoqo.operations*), 98, 252
- `PragmaDephasing` (class in *qoqo.operations*), 103, 254
- `PragmaDepolarising` (class in *qoqo.operations*), 100, 255
- `PragmaGeneralNoise` (class in *qoqo.operations*), 108, 257
- `PragmaGetDensityMatrix` (class in *qoqo.operations*), 70, 259
- `PragmaGetOccupationProbability` (class in *qoqo.operations*), 72, 260
- `PragmaGetPauliProduct` (class in *qoqo.operations*), 74, 262
- `PragmaGetStateVector` (class in *qoqo.operations*), 68, 263
- `PragmaGlobalPhase` (class in *qoqo.operations*), 91, 264
- `PragmaLoop` (class in *qoqo.operations*), 112, 266
- `PragmaOverrotation` (class in *qoqo.operations*), 85, 267
- `PragmaRandomNoise` (class in *qoqo.operations*), 105, 269
- `PragmaRepeatedMeasurement` (class in *qoqo.operations*), 76, 272
- `PragmaRepeatGate` (class in *qoqo.operations*), 83, 270
- `PragmaSetDensityMatrix` (class in *qoqo.operations*), 81, 273
- `PragmaSetNumberOfMeasurements` (class in *qoqo.operations*), 77, 274
- `PragmaSetStateVector` (class in *qoqo.operations*), 79, 276
- `PragmaSleep` (class in *qoqo.operations*), 92, 277
- `PragmaStartDecompositionBlock` (class in *qoqo.operations*), 279
- `PragmaStopDecompositionBlock` (class in *qoqo.operations*), 96, 280
- `PragmaStopParallelBlock` (class in *qoqo.operations*), 89, 281
- `probability()` (*qoqo.operations.PragmaDamping* method), 99, 253
- `probability()` (*qoqo.operations.PragmaDephasing* method), 104, 254
- `probability()` (*qoqo.operations.PragmaDepolarising* method), 101, 256
- `probability()` (*qoqo.operations.PragmaRandomNoise* method), 107, 270
- ## Q
- `qoqo`
- module, 3
- `qoqo.measurements`
- module, 321
- `qoqo.operations`
- module, 8
- `Qsim` (class in *qoqo.operations*), 159, 283

QuantumProgram (class in qoqo), 334, 343
qubit() (qoqo.operations.Hadamard method), 49, 210
qubit() (qoqo.operations.InvSqrtPauliX method), 46, 219
qubit() (qoqo.operations.MeasureQubit method), 67, 220
qubit() (qoqo.operations.PauliX method), 39, 230
qubit() (qoqo.operations.PauliY method), 35, 233
qubit() (qoqo.operations.PauliZ method), 32, 236
qubit() (qoqo.operations.PhaseShiftState0 method), 239
qubit() (qoqo.operations.PhaseShiftState1 method), 242
qubit() (qoqo.operations.PragmaActiveReset method), 95, 247
qubit() (qoqo.operations.PragmaDamping method), 99, 253
qubit() (qoqo.operations.PragmaDephasing method), 104, 254
qubit() (qoqo.operations.PragmaDepolarising method), 102, 256
qubit() (qoqo.operations.PragmaGeneralNoise method), 109, 258
qubit() (qoqo.operations.PragmaRandomNoise method), 107, 270
qubit() (qoqo.operations.RotateAroundSphericalAxis method), 28, 287
qubit() (qoqo.operations.RotateX method), 24, 290
qubit() (qoqo.operations.RotateXY method), 293
qubit() (qoqo.operations.RotateY method), 20, 296
qubit() (qoqo.operations.RotateZ method), 16, 299
qubit() (qoqo.operations.SGate method), 56, 302
qubit() (qoqo.operations.SingleQubitGate method), 12, 306
qubit() (qoqo.operations.SqrtPauliX method), 42, 312
qubit() (qoqo.operations.TGate method), 53, 315
qubit_mapping() (qoqo.operations.PragmaRepeatedMeasurement method), 76, 272
qubit_paulis() (qoqo.operations.PragmaGetPauliProduct method), 75, 262
qubits() (qoqo.operations.MultiQubitMS method), 174, 224
qubits() (qoqo.operations.MultiQubitZZ method), 226
qubits() (qoqo.operations.PragmaOverrotation method), 86, 268
qubits() (qoqo.operations.PragmaSleep method), 93, 278
qubits() (qoqo.operations.PragmaStartDecompositionBlock method), 279
qubits() (qoqo.operations.PragmaStopDecompositionBlock method), 97, 281
qubits() (qoqo.operations.PragmaStopParallelBlock method), 90, 282

R

rate() (qoqo.operations.PragmaDamping method), 99, 253
rate() (qoqo.operations.PragmaDephasing method), 104, 255
rate() (qoqo.operations.PragmaDepolarising method), 102, 256
rates() (qoqo.operations.PragmaGeneralNoise method), 109, 258
readout() (qoqo.operations.MeasureQubit method), 67, 221
readout() (qoqo.operations.PragmaGetDensityMatrix method), 71, 260
readout() (qoqo.operations.PragmaGetOccupationProbability method), 73, 261
readout() (qoqo.operations.PragmaGetPauliProduct method), 75, 262
readout() (qoqo.operations.PragmaGetStateVector method), 69, 264
readout() (qoqo.operations.PragmaRepeatedMeasurement method), 77, 272
readout() (qoqo.operations.PragmaSetNumberOfMeasurements method), 78, 275
readout_index() (qoqo.operations.MeasureQubit method), 67, 221
remap_qubits() (qoqo.Circuit method), 7, 339
remap_qubits() (qoqo.operations.Bogoliubov method), 167, 178
remap_qubits() (qoqo.operations.CNOT method), 115, 179
remap_qubits() (qoqo.operations.ComplexPMInteraction method), 171, 181
remap_qubits() (qoqo.operations.ControlledControlledPauliZ method), 142, 183
remap_qubits() (qoqo.operations.ControlledControlledPhaseShift method), 145, 185
remap_qubits() (qoqo.operations.ControlledPauliY method), 132, 187
remap_qubits() (qoqo.operations.ControlledPauliZ method), 134, 188
remap_qubits() (qoqo.operations.ControlledPhaseShift method), 130, 190
remap_qubits() (qoqo.operations.ControlledRotateX method), 137, 192
remap_qubits() (qoqo.operations.ControlledRotateXY method), 140, 194
remap_qubits() (qoqo.operations.DefinitionBit method), 60, 196
remap_qubits() (qoqo.operations.DefinitionComplex method), 64, 197
remap_qubits() (qoqo.operations.DefinitionFloat method), 62, 199
remap_qubits() (qoqo.operations.DefinitionUsize method), 58, 200

`remap_qubits()` (*qoqo.operations.Fsim method*), 162, 203
`remap_qubits()` (*qoqo.operations.FSwap method*), 119, 202
`remap_qubits()` (*qoqo.operations.GivensRotation method*), 155, 206
`remap_qubits()` (*qoqo.operations.GivensRotationLittleEndian method*), 158, 208
`remap_qubits()` (*qoqo.operations.Hadamard method*), 49, 210
`remap_qubits()` (*qoqo.operations.InputBit method*), 214
`remap_qubits()` (*qoqo.operations.InputSymbolic method*), 66, 215
`remap_qubits()` (*qoqo.operations.InvSqrtISwap method*), 125, 216
`remap_qubits()` (*qoqo.operations.InvSqrtPauliX method*), 46, 219
`remap_qubits()` (*qoqo.operations.ISwap method*), 121, 212
`remap_qubits()` (*qoqo.operations.MeasureQubit method*), 67, 221
`remap_qubits()` (*qoqo.operations.MolmerSorensenXX method*), 150, 222
`remap_qubits()` (*qoqo.operations.MultiQubitMS method*), 174, 224
`remap_qubits()` (*qoqo.operations.MultiQubitZZ method*), 226
`remap_qubits()` (*qoqo.operations.PauliX method*), 39, 230
`remap_qubits()` (*qoqo.operations.PauliY method*), 35, 233
`remap_qubits()` (*qoqo.operations.PauliZ method*), 32, 236
`remap_qubits()` (*qoqo.operations.PhaseShiftedControlledZ method*), 244
`remap_qubits()` (*qoqo.operations.PhaseShiftedControlledZ method*), 245
`remap_qubits()` (*qoqo.operations.PhaseShiftState0 method*), 239
`remap_qubits()` (*qoqo.operations.PhaseShiftState1 method*), 242
`remap_qubits()` (*qoqo.operations.PMInteraction method*), 169, 227
`remap_qubits()` (*qoqo.operations.PragmaActiveReset method*), 95, 247
`remap_qubits()` (*qoqo.operations.PragmaBoostNoise method*), 88, 248
`remap_qubits()` (*qoqo.operations.PragmaChangeDevice method*), 249
`remap_qubits()` (*qoqo.operations.PragmaConditional method*), 111, 251
`remap_qubits()` (*qoqo.operations.PragmaDamping method*), 99, 253
`remap_qubits()` (*qoqo.operations.PragmaDephasing method*), 104, 255
`remap_qubits()` (*qoqo.operations.PragmaDepolarising method*), 102, 256
`remap_qubits()` (*qoqo.operations.PragmaGeneralNoise method*), 109, 258
`remap_qubits()` (*qoqo.operations.PragmaGetDensityMatrix method*), 71, 260
`remap_qubits()` (*qoqo.operations.PragmaGetOccupationProbability method*), 73, 261
`remap_qubits()` (*qoqo.operations.PragmaGetPauliProduct method*), 75, 262
`remap_qubits()` (*qoqo.operations.PragmaGetStateVector method*), 69, 264
`remap_qubits()` (*qoqo.operations.PragmaGlobalPhase method*), 92, 265
`remap_qubits()` (*qoqo.operations.PragmaLoop method*), 113, 266
`remap_qubits()` (*qoqo.operations.PragmaOverrotation method*), 86, 268
`remap_qubits()` (*qoqo.operations.PragmaRandomNoise method*), 107, 270
`remap_qubits()` (*qoqo.operations.PragmaRepeatedMeasurement method*), 77, 272
`remap_qubits()` (*qoqo.operations.PragmaRepeatGate method*), 84, 271
`remap_qubits()` (*qoqo.operations.PragmaSetDensityMatrix method*), 82, 274
`remap_qubits()` (*qoqo.operations.PragmaSetNumberOfMeasurements method*), 78, 275
`remap_qubits()` (*qoqo.operations.PragmaSetStateVector method*), 80, 276
`remap_qubits()` (*qoqo.operations.PragmaSleep method*), 93, 278
`remap_qubits()` (*qoqo.operations.PragmaStartDecompositionBlock method*), 279
`remap_qubits()` (*qoqo.operations.PragmaStopDecompositionBlock method*), 97, 281
`remap_qubits()` (*qoqo.operations.PragmaStopParallelBlock method*), 90, 282
`remap_qubits()` (*qoqo.operations.Qsim method*), 160, 283
`remap_qubits()` (*qoqo.operations.RotateAroundSphericalAxis method*), 28, 287
`remap_qubits()` (*qoqo.operations.RotateX method*), 24, 290
`remap_qubits()` (*qoqo.operations.RotateXY method*), 293
`remap_qubits()` (*qoqo.operations.RotateY method*), 20, 296
`remap_qubits()` (*qoqo.operations.RotateZ method*), 16, 299
`remap_qubits()` (*qoqo.operations.SGate method*), 56, 302

remap_qubits() (*qoqo.operations.SingleQubitGate* method), 12, 306
 remap_qubits() (*qoqo.operations.SpinInteraction* method), 164, 308
 remap_qubits() (*qoqo.operations.SqrtISwap* method), 123, 309
 remap_qubits() (*qoqo.operations.SqrtPauliX* method), 42, 312
 remap_qubits() (*qoqo.operations.SWAP* method), 117, 303
 remap_qubits() (*qoqo.operations.TGate* method), 53, 315
 remap_qubits() (*qoqo.operations.Toffoli* method), 148, 317
 remap_qubits() (*qoqo.operations.VariableMSXX* method), 153, 319
 remap_qubits() (*qoqo.operations.XY* method), 127, 320
 reordering_dictionary() (*qoqo.operations.PragmaStartDecompositionBlock* method), 280
 repetition_coefficient() (*qoqo.operations.PragmaRepeatGate* method), 84, 271
 repetitions() (*qoqo.operations.PragmaLoop* method), 113, 267
 RotateAroundSphericalAxis (class in *qoqo.operations*), 25, 284
 RotateX (class in *qoqo.operations*), 21, 288
 RotateXY (class in *qoqo.operations*), 291
 RotateY (class in *qoqo.operations*), 17, 294
 RotateZ (class in *qoqo.operations*), 13, 297
 run() (*qoqo.QuantumProgram* method), 336, 344
 run_registers() (*qoqo.QuantumProgram* method), 336, 345

S

SGate (class in *qoqo.operations*), 54, 300
 SingleQubitGate (class in *qoqo.operations*), 10, 304
 sleep_time() (*qoqo.operations.PragmaSleep* method), 94, 278
 spherical_phi() (*qoqo.operations.RotateAroundSphericalAxis* method), 28, 287
 spherical_theta() (*qoqo.operations.RotateAroundSphericalAxis* method), 28, 287
 SpinInteraction (class in *qoqo.operations*), 163, 307
 SqrtISwap (class in *qoqo.operations*), 122, 309
 SqrtPauliX (class in *qoqo.operations*), 40, 310
 statevector() (*qoqo.operations.PragmaSetStateVector* method), 80, 277
 substitute_parameters() (*qoqo.Circuit* method), 7, 339
 substitute_parameters() (*qoqo.measurements.Cheated* method), 323
 substitute_parameters() (*qoqo.measurements.CheatedPauliZProduct* method), 327
 substitute_parameters() (*qoqo.measurements.ClassicalRegister* method), 330
 substitute_parameters() (*qoqo.measurements.PauliZProduct* method), 332
 substitute_parameters() (*qoqo.operations.Bogoliubov* method), 167, 178
 substitute_parameters() (*qoqo.operations.CNOT* method), 115, 180
 substitute_parameters() (*qoqo.operations.ComplexPMInteraction* method), 172, 181
 substitute_parameters() (*qoqo.operations.ControlledControlledPauliZ* method), 143, 183
 substitute_parameters() (*qoqo.operations.ControlledControlledPhaseShift* method), 145, 185
 substitute_parameters() (*qoqo.operations.ControlledPauliY* method), 132, 187
 substitute_parameters() (*qoqo.operations.ControlledPauliZ* method), 134, 189
 substitute_parameters() (*qoqo.operations.ControlledPhaseShift* method), 130, 190
 substitute_parameters() (*qoqo.operations.ControlledRotateX* method), 137, 192
 substitute_parameters() (*qoqo.operations.ControlledRotateXY* method), 140, 195
 substitute_parameters() (*qoqo.operations.DefinitionBit* method), 60, 196
 substitute_parameters() (*qoqo.operations.DefinitionComplex* method), 64, 198
 substitute_parameters() (*qoqo.operations.DefinitionFloat* method), 62, 199
 substitute_parameters() (*qoqo.operations.DefinitionUsize* method), 58, 201
 substitute_parameters() (*qoqo.operations.Fsim* method), 163, 204
 substitute_parameters() (*qoqo.operations.FSwap* method), 119, 202

`substitute_parameters()`
 (`qoqo.operations.GivensRotation` method), 155, 206
`substitute_parameters()`
 (`qoqo.operations.GivensRotationLittleEndian` method), 158, 208
`substitute_parameters()`
 (`qoqo.operations.Hadamard` method), 49, 211
`substitute_parameters()` (`qoqo.operations.InputBit` method), 214
`substitute_parameters()`
 (`qoqo.operations.InputSymbolic` method), 66, 215
`substitute_parameters()`
 (`qoqo.operations.InvSqrtISwap` method), 125, 216
`substitute_parameters()`
 (`qoqo.operations.InvSqrtPauliX` method), 46, 219
`substitute_parameters()` (`qoqo.operations.ISwap` method), 121, 212
`substitute_parameters()`
 (`qoqo.operations.MeasureQubit` method), 68, 221
`substitute_parameters()`
 (`qoqo.operations.MolmerSorensenXX` method), 150, 222
`substitute_parameters()`
 (`qoqo.operations.MultiQubitMS` method), 174, 224
`substitute_parameters()`
 (`qoqo.operations.MultiQubitZZ` method), 226
`substitute_parameters()` (`qoqo.operations.PauliX` method), 39, 230
`substitute_parameters()` (`qoqo.operations.PauliY` method), 35, 233
`substitute_parameters()` (`qoqo.operations.PauliZ` method), 32, 236
`substitute_parameters()`
 (`qoqo.operations.PhaseShiftedControlledPhase` method), 244
`substitute_parameters()`
 (`qoqo.operations.PhaseShiftedControlledZ` method), 246
`substitute_parameters()`
 (`qoqo.operations.PhaseShiftState0` method), 239
`substitute_parameters()`
 (`qoqo.operations.PhaseShiftState1` method), 242
`substitute_parameters()`
 (`qoqo.operations.PMInteraction` method), 169, 227
`substitute_parameters()`
 (`qoqo.operations.PragmaActiveReset` method), 95, 247
`substitute_parameters()`
 (`qoqo.operations.PragmaBoostNoise` method), 88, 248
`substitute_parameters()`
 (`qoqo.operations.PragmaChangeDevice` method), 249
`substitute_parameters()`
 (`qoqo.operations.PragmaConditional` method), 112, 251
`substitute_parameters()`
 (`qoqo.operations.PragmaDamping` method), 99, 253
`substitute_parameters()`
 (`qoqo.operations.PragmaDephasing` method), 104, 255
`substitute_parameters()`
 (`qoqo.operations.PragmaDepolarising` method), 102, 257
`substitute_parameters()`
 (`qoqo.operations.PragmaGeneralNoise` method), 109, 258
`substitute_parameters()`
 (`qoqo.operations.PragmaGetDensityMatrix` method), 71, 260
`substitute_parameters()`
 (`qoqo.operations.PragmaGetOccupationProbability` method), 73, 261
`substitute_parameters()`
 (`qoqo.operations.PragmaGetPauliProduct` method), 75, 263
`substitute_parameters()`
 (`qoqo.operations.PragmaGetStateVector` method), 70, 264
`substitute_parameters()`
 (`qoqo.operations.PragmaGlobalPhase` method), 92, 265
`substitute_parameters()`
 (`qoqo.operations.PragmaLoop` method), 114, 267
`substitute_parameters()`
 (`qoqo.operations.PragmaOverrotation` method), 87, 268
`substitute_parameters()`
 (`qoqo.operations.PragmaRandomNoise` method), 107, 270
`substitute_parameters()`
 (`qoqo.operations.PragmaRepeatedMeasurement` method), 77, 273
`substitute_parameters()`
 (`qoqo.operations.PragmaRepeatGate` method),

- 84, 271
- `substitute_parameters()`
(*qoqo.operations.PragmaSetDensityMatrix* method), 83, 274
- `substitute_parameters()`
(*qoqo.operations.PragmaSetNumberOfMeasurements* method), 79, 275
- `substitute_parameters()`
(*qoqo.operations.PragmaSetStateVector* method), 81, 277
- `substitute_parameters()`
(*qoqo.operations.PragmaSleep* method), 94, 278
- `substitute_parameters()`
(*qoqo.operations.PragmaStartDecompositionBlock* method), 280
- `substitute_parameters()`
(*qoqo.operations.PragmaStopDecompositionBlock* method), 97, 281
- `substitute_parameters()`
(*qoqo.operations.PragmaStopParallelBlock* method), 90, 282
- `substitute_parameters()` (*qoqo.operations.Qsim* method), 160, 284
- `substitute_parameters()`
(*qoqo.operations.RotateAroundSphericalAxis* method), 28, 287
- `substitute_parameters()` (*qoqo.operations.RotateX* method), 24, 290
- `substitute_parameters()`
(*qoqo.operations.RotateXY* method), 293
- `substitute_parameters()` (*qoqo.operations.RotateY* method), 20, 296
- `substitute_parameters()` (*qoqo.operations.RotateZ* method), 16, 299
- `substitute_parameters()` (*qoqo.operations.SGate* method), 56, 302
- `substitute_parameters()`
(*qoqo.operations.SingleQubitGate* method), 12, 306
- `substitute_parameters()`
(*qoqo.operations.SpinInteraction* method), 165, 308
- `substitute_parameters()`
(*qoqo.operations.SqrtISwap* method), 123, 310
- `substitute_parameters()`
(*qoqo.operations.SqrtPauliX* method), 42, 312
- `substitute_parameters()` (*qoqo.operations.SWAP* method), 117, 304
- `substitute_parameters()` (*qoqo.operations.TGate* method), 53, 315
- `substitute_parameters()` (*qoqo.operations.Toffoli* method), 148, 317
- `substitute_parameters()`
(*qoqo.operations.VariableMSXX* method), 153, 319
- `substitute_parameters()` (*qoqo.operations.XY* method), 128, 321
- `successors()` (*qoqo.CircuitDag* method), 343
- `superoperator()` (*qoqo.operations.PragmaDamping* method), 100, 253
- `superoperator()` (*qoqo.operations.PragmaDephasing* method), 105, 255
- `superoperator()` (*qoqo.operations.PragmaDepolarising* method), 102, 257
- `superoperator()` (*qoqo.operations.PragmaGeneralNoise* method), 110, 259
- `superoperator()` (*qoqo.operations.PragmaRandomNoise* method), 107, 270
- `SWAP` (class in *qoqo.operations*), 116, 303
- ## T
- `t()` (*qoqo.operations.Fsim* method), 163, 204
- `t()` (*qoqo.operations.PMInteraction* method), 169, 228
- `t_imag()` (*qoqo.operations.ComplexPMInteraction* method), 172, 181
- `t_real()` (*qoqo.operations.ComplexPMInteraction* method), 172, 181
- `tags()` (*qoqo.operations.Bogoliubov* method), 167, 178
- `tags()` (*qoqo.operations.CNOT* method), 116, 180
- `tags()` (*qoqo.operations.ComplexPMInteraction* method), 172, 181
- `tags()` (*qoqo.operations.ControlledControlledPauliZ* method), 143, 183
- `tags()` (*qoqo.operations.ControlledControlledPhaseShift* method), 146, 186
- `tags()` (*qoqo.operations.ControlledPauliY* method), 133, 187
- `tags()` (*qoqo.operations.ControlledPauliZ* method), 135, 189
- `tags()` (*qoqo.operations.ControlledPhaseShift* method), 130, 191
- `tags()` (*qoqo.operations.ControlledRotateX* method), 138, 193
- `tags()` (*qoqo.operations.ControlledRotateXY* method), 140, 195
- `tags()` (*qoqo.operations.DefinitionBit* method), 60, 197
- `tags()` (*qoqo.operations.DefinitionComplex* method), 64, 198
- `tags()` (*qoqo.operations.DefinitionFloat* method), 62, 199
- `tags()` (*qoqo.operations.DefinitionUsize* method), 59, 201
- `tags()` (*qoqo.operations.Fsim* method), 163, 204
- `tags()` (*qoqo.operations.FSwap* method), 120, 202

- tags() (*qoqo.operations.GivensRotation* method), 156, 206
- tags() (*qoqo.operations.GivensRotationLittleEndian* method), 158, 208
- tags() (*qoqo.operations.Hadamard* method), 50, 211
- tags() (*qoqo.operations.InputBit* method), 214
- tags() (*qoqo.operations.InputSymbolic* method), 66, 215
- tags() (*qoqo.operations.InvSqrtISwap* method), 126, 217
- tags() (*qoqo.operations.InvSqrtPauliX* method), 46, 220
- tags() (*qoqo.operations.ISwap* method), 122, 212
- tags() (*qoqo.operations.MeasureQubit* method), 68, 221
- tags() (*qoqo.operations.MolmerSorensenXX* method), 151, 223
- tags() (*qoqo.operations.MultiQubitMS* method), 174, 224
- tags() (*qoqo.operations.MultiQubitZZ* method), 226
- tags() (*qoqo.operations.PauliX* method), 39, 231
- tags() (*qoqo.operations.PauliY* method), 36, 233
- tags() (*qoqo.operations.PauliZ* method), 32, 236
- tags() (*qoqo.operations.PhaseShiftedControlledPhase* method), 244
- tags() (*qoqo.operations.PhaseShiftedControlledZ* method), 246
- tags() (*qoqo.operations.PhaseShiftState0* method), 239
- tags() (*qoqo.operations.PhaseShiftState1* method), 242
- tags() (*qoqo.operations.PMInteraction* method), 169, 228
- tags() (*qoqo.operations.PragmaActiveReset* method), 96, 247
- tags() (*qoqo.operations.PragmaBoostNoise* method), 89, 248
- tags() (*qoqo.operations.PragmaChangeDevice* method), 250
- tags() (*qoqo.operations.PragmaConditional* method), 112, 252
- tags() (*qoqo.operations.PragmaDamping* method), 100, 253
- tags() (*qoqo.operations.PragmaDephasing* method), 105, 255
- tags() (*qoqo.operations.PragmaDepolarising* method), 102, 257
- tags() (*qoqo.operations.PragmaGeneralNoise* method), 110, 259
- tags() (*qoqo.operations.PragmaGetDensityMatrix* method), 71, 260
- tags() (*qoqo.operations.PragmaGetOccupationProbability* method), 73, 261
- tags() (*qoqo.operations.PragmaGetPauliProduct* method), 75, 263
- tags() (*qoqo.operations.PragmaGetStateVector* method), 70, 264
- tags() (*qoqo.operations.PragmaGlobalPhase* method), 92, 266
- tags() (*qoqo.operations.PragmaLoop* method), 114, 267
- tags() (*qoqo.operations.PragmaOverrotation* method), 87, 268
- tags() (*qoqo.operations.PragmaRandomNoise* method), 107, 270
- tags() (*qoqo.operations.PragmaRepeatedMeasurement* method), 77, 273
- tags() (*qoqo.operations.PragmaRepeatGate* method), 85, 272
- tags() (*qoqo.operations.PragmaSetDensityMatrix* method), 83, 274
- tags() (*qoqo.operations.PragmaSetNumberOfMeasurements* method), 79, 276
- tags() (*qoqo.operations.PragmaSetStateVector* method), 81, 277
- tags() (*qoqo.operations.PragmaSleep* method), 94, 279
- tags() (*qoqo.operations.PragmaStartDecompositionBlock* method), 280
- tags() (*qoqo.operations.PragmaStopDecompositionBlock* method), 97, 281
- tags() (*qoqo.operations.PragmaStopParallelBlock* method), 91, 282
- tags() (*qoqo.operations.Qsim* method), 160, 284
- tags() (*qoqo.operations.RotateAroundSphericalAxis* method), 29, 287
- tags() (*qoqo.operations.RotateX* method), 24, 290
- tags() (*qoqo.operations.RotateXY* method), 294
- tags() (*qoqo.operations.RotateY* method), 20, 297
- tags() (*qoqo.operations.RotateZ* method), 16, 300
- tags() (*qoqo.operations.SGate* method), 57, 302
- tags() (*qoqo.operations.SingleQubitGate* method), 13, 307
- tags() (*qoqo.operations.SpinInteraction* method), 165, 308
- tags() (*qoqo.operations.SqrtISwap* method), 124, 310
- tags() (*qoqo.operations.SqrtPauliX* method), 43, 313
- tags() (*qoqo.operations.SWAP* method), 118, 304
- tags() (*qoqo.operations.TGate* method), 53, 315
- tags() (*qoqo.operations.Toffoli* method), 148, 317
- tags() (*qoqo.operations.VariableMSXX* method), 153, 319
- tags() (*qoqo.operations.XY* method), 128, 321
- target() (*qoqo.operations.Bogoliubov* method), 168, 178
- target() (*qoqo.operations.CNOT* method), 116, 180
- target() (*qoqo.operations.ComplexPMInteraction* method), 172, 182
- target() (*qoqo.operations.ControlledControlledPauliZ* method), 143, 184
- target() (*qoqo.operations.ControlledControlledPhaseShift* method), 146, 186

[target\(\)](#) (*qogo.operations.ControlledPauliY method*), [133](#), [187](#)
[target\(\)](#) (*qogo.operations.ControlledPauliZ method*), [135](#), [189](#)
[target\(\)](#) (*qogo.operations.ControlledPhaseShift method*), [131](#), [191](#)
[target\(\)](#) (*qogo.operations.ControlledRotateX method*), [138](#), [193](#)
[target\(\)](#) (*qogo.operations.ControlledRotateXY method*), [140](#), [195](#)
[target\(\)](#) (*qogo.operations.Fsim method*), [163](#), [204](#)
[target\(\)](#) (*qogo.operations.FSwap method*), [120](#), [202](#)
[target\(\)](#) (*qogo.operations.GivensRotation method*), [156](#), [206](#)
[target\(\)](#) (*qogo.operations.GivensRotationLittleEndian method*), [158](#), [208](#)
[target\(\)](#) (*qogo.operations.InvSqrtISwap method*), [126](#), [217](#)
[target\(\)](#) (*qogo.operations.ISwap method*), [122](#), [213](#)
[target\(\)](#) (*qogo.operations.MolmerSorensenXX method*), [151](#), [223](#)
[target\(\)](#) (*qogo.operations.PhaseShiftedControlledPhase method*), [244](#)
[target\(\)](#) (*qogo.operations.PhaseShiftedControlledZ method*), [246](#)
[target\(\)](#) (*qogo.operations.PMInteraction method*), [170](#), [228](#)
[target\(\)](#) (*qogo.operations.Qsim method*), [160](#), [284](#)
[target\(\)](#) (*qogo.operations.SpinInteraction method*), [165](#), [308](#)
[target\(\)](#) (*qogo.operations.SqrtISwap method*), [124](#), [310](#)
[target\(\)](#) (*qogo.operations.SWAP method*), [118](#), [304](#)
[target\(\)](#) (*qogo.operations.Toffoli method*), [149](#), [317](#)
[target\(\)](#) (*qogo.operations.VariableMSXX method*), [153](#), [319](#)
[target\(\)](#) (*qogo.operations.XY method*), [128](#), [321](#)
[TGate](#) (class in *qogo.operations*), [50](#), [313](#)
[theta\(\)](#) (*qogo.operations.ControlledControlledPhaseShift method*), [146](#), [186](#)
[theta\(\)](#) (*qogo.operations.ControlledPhaseShift method*), [131](#), [191](#)
[theta\(\)](#) (*qogo.operations.ControlledRotateX method*), [138](#), [193](#)
[theta\(\)](#) (*qogo.operations.ControlledRotateXY method*), [140](#), [195](#)
[theta\(\)](#) (*qogo.operations.GivensRotation method*), [156](#), [206](#)
[theta\(\)](#) (*qogo.operations.GivensRotationLittleEndian method*), [158](#), [208](#)
[theta\(\)](#) (*qogo.operations.MultiQubitMS method*), [174](#), [225](#)
[theta\(\)](#) (*qogo.operations.MultiQubitZZ method*), [226](#)
[theta\(\)](#) (*qogo.operations.PhaseShiftedControlledPhase method*), [244](#)
[theta\(\)](#) (*qogo.operations.PhaseShiftState0 method*), [239](#)
[theta\(\)](#) (*qogo.operations.PhaseShiftState1 method*), [242](#)
[theta\(\)](#) (*qogo.operations.RotateAroundSphericalAxis method*), [29](#), [288](#)
[theta\(\)](#) (*qogo.operations.RotateX method*), [25](#), [291](#)
[theta\(\)](#) (*qogo.operations.RotateXY method*), [294](#)
[theta\(\)](#) (*qogo.operations.RotateY method*), [21](#), [297](#)
[theta\(\)](#) (*qogo.operations.RotateZ method*), [17](#), [300](#)
[theta\(\)](#) (*qogo.operations.VariableMSXX method*), [153](#), [319](#)
[theta\(\)](#) (*qogo.operations.XY method*), [128](#), [321](#)
[to_bincode\(\)](#) (*qogo.Circuit method*), [7](#), [340](#)
[to_bincode\(\)](#) (*qogo.CircuitDag method*), [343](#)
[to_bincode\(\)](#) (*qogo.measurements.Cheated method*), [323](#)
[to_bincode\(\)](#) (*qogo.measurements.CheatedInput method*), [325](#)
[to_bincode\(\)](#) (*qogo.measurements.CheatedPauliZProduct method*), [327](#)
[to_bincode\(\)](#) (*qogo.measurements.CheatedPauliZProductInput method*), [328](#)
[to_bincode\(\)](#) (*qogo.measurements.ClassicalRegister method*), [330](#)
[to_bincode\(\)](#) (*qogo.measurements.PauliZProduct method*), [332](#)
[to_bincode\(\)](#) (*qogo.measurements.PauliZProductInput method*), [334](#)
[to_bincode\(\)](#) (*qogo.QuantumProgram method*), [336](#), [345](#)
[to_circuit\(\)](#) (*qogo.CircuitDag method*), [343](#)
[to_json\(\)](#) (*qogo.Circuit method*), [7](#), [340](#)
[to_json\(\)](#) (*qogo.measurements.Cheated method*), [324](#)
[to_json\(\)](#) (*qogo.measurements.CheatedInput method*), [325](#)
[to_json\(\)](#) (*qogo.measurements.CheatedPauliZProduct method*), [327](#)
[to_json\(\)](#) (*qogo.measurements.CheatedPauliZProductInput method*), [329](#)
[to_json\(\)](#) (*qogo.measurements.ClassicalRegister method*), [330](#)
[to_json\(\)](#) (*qogo.measurements.PauliZProduct method*), [332](#)
[to_json\(\)](#) (*qogo.measurements.PauliZProductInput method*), [334](#)
[to_json\(\)](#) (*qogo.QuantumProgram method*), [336](#), [345](#)
[Toffoli](#) (class in *qogo.operations*), [146](#), [316](#)

U

[u\(\)](#) (*qogo.operations.Fsim method*), [163](#), [204](#)
[unitary_matrix\(\)](#) (*qogo.operations.Bogoliubov method*), [168](#), [178](#)

- `unitary_matrix()` (*qoqo.operations.CNOT* method), 116, 180
- `unitary_matrix()` (*qoqo.operations.ComplexPMInteraction* method), 172, 182
- `unitary_matrix()` (*qoqo.operations.ControlledControlledPauliZ* method), 143, 184
- `unitary_matrix()` (*qoqo.operations.ControlledControlledPauliY* method), 146, 186
- `unitary_matrix()` (*qoqo.operations.ControlledPauliY* method), 133, 187
- `unitary_matrix()` (*qoqo.operations.ControlledPauliZ* method), 135, 189
- `unitary_matrix()` (*qoqo.operations.ControlledPhaseShift* method), 131, 191
- `unitary_matrix()` (*qoqo.operations.ControlledRotateX* method), 138, 193
- `unitary_matrix()` (*qoqo.operations.ControlledRotateXY* method), 141, 195
- `unitary_matrix()` (*qoqo.operations.Fsim* method), 163, 204
- `unitary_matrix()` (*qoqo.operations.FSwap* method), 120, 202
- `unitary_matrix()` (*qoqo.operations.GivensRotation* method), 156, 206
- `unitary_matrix()` (*qoqo.operations.GivensRotationLittleEndian* method), 158, 208
- `unitary_matrix()` (*qoqo.operations.Hadamard* method), 50, 211
- `unitary_matrix()` (*qoqo.operations.InvSqrtISwap* method), 126, 217
- `unitary_matrix()` (*qoqo.operations.InvSqrtPauliX* method), 46, 220
- `unitary_matrix()` (*qoqo.operations.ISwap* method), 122, 213
- `unitary_matrix()` (*qoqo.operations.MolmerSorensenXX* method), 151, 223
- `unitary_matrix()` (*qoqo.operations.MultiQubitMS* method), 174, 225
- `unitary_matrix()` (*qoqo.operations.MultiQubitZZ* method), 226
- `unitary_matrix()` (*qoqo.operations.PauliX* method), 39, 231
- `unitary_matrix()` (*qoqo.operations.PauliY* method), 36, 234
- `unitary_matrix()` (*qoqo.operations.PauliZ* method), 32, 236
- `unitary_matrix()` (*qoqo.operations.PhaseShiftedControlledPhase* method), 244
- `unitary_matrix()` (*qoqo.operations.PhaseShiftedControlledZ* method), 246
- `unitary_matrix()` (*qoqo.operations.PhaseShiftState0* method), 239
- `unitary_matrix()` (*qoqo.operations.PhaseShiftState1* method), 242
- `unitary_matrix()` (*qoqo.operations.PMInteraction* method), 170, 228
- `unitary_matrix()` (*qoqo.operations.Qsim* method), 160, 284
- `unitary_matrix()` (*qoqo.operations.RotateAroundSphericalAxis* method), 29, 288
- `unitary_matrix()` (*qoqo.operations.RotateX* method), 25, 291
- `unitary_matrix()` (*qoqo.operations.RotateXY* method), 294
- `unitary_matrix()` (*qoqo.operations.RotateY* method), 21, 297
- `unitary_matrix()` (*qoqo.operations.RotateZ* method), 17, 300
- `unitary_matrix()` (*qoqo.operations.SGate* method), 57, 303
- `unitary_matrix()` (*qoqo.operations.SingleQubitGate* method), 13, 307
- `unitary_matrix()` (*qoqo.operations.SpinInteraction* method), 165, 308
- `unitary_matrix()` (*qoqo.operations.SqrtISwap* method), 124, 310
- `unitary_matrix()` (*qoqo.operations.SqrtPauliX* method), 43, 313
- `unitary_matrix()` (*qoqo.operations.SWAP* method), 118, 304
- `unitary_matrix()` (*qoqo.operations.TGate* method), 53, 316
- `unitary_matrix()` (*qoqo.operations.Toffoli* method), 149, 317
- `unitary_matrix()` (*qoqo.operations.VariableMSXX* method), 153, 319
- `unitary_matrix()` (*qoqo.operations.XY* method), 128, 321
- V**
- `value()` (*qoqo.operations.InputBit* method), 214
- `VariableMSXX` (class in *qoqo.operations*), 151, 318
- `variance()` (*qoqo.operations.PragmaOverrotation* method), 87, 269
- W**
- `wrapped_hqslang()` (*qoqo.operations.PragmaChangeDevice* method), 250
- `wrapped_operation()` (*qoqo.operations.PragmaChangeDevice* method), 250
- `wrapped_tags()` (*qoqo.operations.PragmaChangeDevice* method), 250
- X**
- `x()` (*qoqo.operations.Qsim* method), 161, 284
- `x()` (*qoqo.operations.SpinInteraction* method), 165, 309
- `XY` (class in *qoqo.operations*), 126, 319

Y

`y()` (*qoqo.operations.Qsim method*), [161](#), [284](#)

`y()` (*qoqo.operations.SpinInteraction method*), [165](#), [309](#)

Z

`z()` (*qoqo.operations.Qsim method*), [161](#), [284](#)

`z()` (*qoqo.operations.SpinInteraction method*), [165](#), [309](#)