PyQrack

Release v0.16.3

Daniel Strano and the Qrack Contributors

Sep 10, 2023

CONTENTS

CHAPTER

ONE

INTRODUCTION

PyQrack is a pure Python *ctypes* wrapper on the C++11 Qrack library, for quantum computing simulation.

An introductory talk to Qrack can be found here Intro to Qrack: a framework for fast quantum simulation by Daniel Strano | Quantum Software Talks.

CHAPTER

TWO

HARDWARE COMPILATION

2.1 Efficient Unitary Clifford+RZ Simulation

QStabilizerHybrid is classically efficient for the gate set "Clifford+RZ," (or "Clifford+T,") except for measurement, since v1.14. The entire unitary portion of circuit simulation, before measurement, has a polynomial-complexity simulation algorithm, in space and time requirements. If measuring across the full width of the simulator, or sampling, measurement (alone) scales exponentially in space requirements proportional to (less than or up to) the number of non-Clifford RZ (or T) gates, and exponentially in time requirements proportional to base logical qubit count in the simulator instance.

No special considerations are necessary to engage this simulation mode: simply restrict your gate set to Clifford+RZ, when using any simulator that properly includes the *QStabilizerHybrid* layer, such as the default optimal simulator stack.

2.2 Output Unitary Clifford+RZ Simulation For Quantum Hardware

It is theoretically possible to use the Clifford+RZ improvements of v1.14 to compile for hardware. (*QUnit* "Schmidt decomposition cannot be used over *QStabilizerHybrid*, for this.)

Since v1.15, it is now possible to output *QStabilizerHybrid* state to file, (not while using *QUnit*). This is done with instance method *out_to_file(filename)*, on a simulator instance. With class method *QrackSimulator.file_to_qiskit_circuit(filename)*, we perform the algorithm that follows, to produce a Qiskit circuit. The files have the following format, by line:

[Logical qubit count]

[Stabilizer qubit count, including ancillae]

[Stabilizer x/z/r generators, one row per line, "tableau" format, repeated for logical qubit count of rows x2]

[Per-qubit MPS buffers, 2x2 complex matrices, row-major order, one matrix per line, repeated for stabilizer qubit count of rows]

For example:

3

3

1 1 0 0 1 0 2

0 1 0 1 0 0 0

0 0 0 0 0 1 0

0 0 0 1 0 1 2

0 0 0 1 1 0 0

1 1 1 0 1 0 0

(1,0) (0,0) (0,0) (1,0)

(1,0) (0,0) (0,0) (1,0)

(0,0) (0.707107,-0.707107) (0,1) (0,0)

is a valid file, with 0 ancillae. It is theoretically relatively easy to prepare this result of unitary circuit simulation on a quantum hardware device: first prepare the stabilizer state, (with purely Clifford gates,) then apply the (potentially non-Clifford) 2x2 matrices over the same sequential qubit index order. This can represent a universal quantum state of the logical qubits.

It might be helpful to look at Qiskit's *Clifford <https://qiskit.org/documentation/stubs/qiskit.quantum_info.Clifford.html>_* and *Clifford.to_circuit() <https://qiskit.org/documentation/stubs/qiskit.quantum_info.Clifford.to_circuit.html#qiskit.quantum_info.Cliffo* to convert this stabilizer state representation to a state preparation circuit.

2.2.1 API Reference

This page contains auto-generated API reference documentation[1].

pyqrack

Subpackages

pyqrack.qrack_system

Submodules

pyqrack.qrack_system.qrack_system

Module Contents

Classes

QrackSystem

class pyqrack.qrack_system.qrack_system.**QrackSystem**

[1] Created with sphinx-autoapi

Package Contents

Classes

QrackSystem

Attributes

Qrack

class pyqrack.qrack_system.**QrackSystem**

pyqrack.qrack_system.**Qrack**

pyqrack.tests

Submodules

pyqrack.tests.test_mirror_circuits

Module Contents

Classes

TestMirrorCircuits

Functions

mc_gate(circ, c, mat, q, p)	add a Multi-controlled mat gate, controlled on qubits c, targeting qubit q
gen_random_1q_gates(n_qubits, gate_count_1qb, depth)	
gen_random_multiq_gates(n_qubits, gate_count_2qb, ...)	
random_bit_string(n_qubits)	
single_qubit_gates(n)	
mirrored_single_qubit_gate(sim, gate, q)	
multi_qubit_gates(circuit, gate, b1, b2, b3)	
mirrored_multi_qubit_gates(sim, gate, b1, b2, b3)	
mirror_circuit(qsim, n_qubits, initial_bitstr, depth, ...)	
trials()	
n_shots()	
n_qubits()	
depth()	
gate_count_1qb()	
gate_count_multiqb()	
gate_count_2qb()	

Attributes

X
Y
Z
SQRT1_2

pyqrack.tests.test_mirror_circuits.**X = [0, 1, 1, 0]**

pyqrack.tests.test_mirror_circuits.**Y**

pyqrack.tests.test_mirror_circuits.**Z**

pyqrack.tests.test_mirror_circuits.**SQRT1_2**

pyqrack.tests.test_mirror_circuits.**mc_gate**(*circ:* pyqrack.QrackCircuit, *c: List[int]*, *mat: List[complex]*, *q: int*, *p: int*)

add a Multi-controlled mat gate, controlled on qubits c, targeting qubit q

pyqrack.tests.test_mirror_circuits.**gen_random_1q_gates**(*n_qubits*, *gate_count_1qb*, *depth*)

pyqrack.tests.test_mirror_circuits.**gen_random_multiq_gates**(*n_qubits*, *gate_count_2qb*, *gate_count_multiqb*, *depth*)

pyqrack.tests.test_mirror_circuits.**random_bit_string**(*n_qubits*)

pyqrack.tests.test_mirror_circuits.**single_qubit_gates**(*n*)

pyqrack.tests.test_mirror_circuits.**mirrored_single_qubit_gate**(*sim:* pyqrack.QrackSimulator, *gate*, *q*)

pyqrack.tests.test_mirror_circuits.**multi_qubit_gates**(*circuit*, *gate*, *b1*, *b2*, *b3*)

pyqrack.tests.test_mirror_circuits.**mirrored_multi_qubit_gates**(*sim:* pyqrack.QrackSimulator, *gate*, *b1*, *b2*, *b3*)

pyqrack.tests.test_mirror_circuits.**mirror_circuit**(*qsim:* pyqrack.QrackSimulator, *n_qubits: int*, *initial_bitstr: int*, *depth: int*, *random_1q_gates: List[List[int]]*, *random_multiq_gates: List[List[Dict[str, int]]]*, *shots: int*)

pyqrack.tests.test_mirror_circuits.**trials**()

pyqrack.tests.test_mirror_circuits.**n_shots**()

pyqrack.tests.test_mirror_circuits.**n_qubits**()

pyqrack.tests.test_mirror_circuits.**depth**()

pyqrack.tests.test_mirror_circuits.**gate_count_1qb**()

pyqrack.tests.test_mirror_circuits.**gate_count_multiqb**()

pyqrack.tests.test_mirror_circuits.**gate_count_2qb**()

class pyqrack.tests.test_mirror_circuits.**TestMirrorCircuits**

test_mirror_circuits(*depth: int*, *trials: int*, *gate_count_1qb: int*, *n_qubits: int*, *gate_count_multiqb: int*, *gate_count_2qb: int*, *n_shots: int*)

pyqrack.util

Submodules

pyqrack.util.convert_qiskit_circuit_to_qasm_experiment

Module Contents

Classes

QrackQasmQobjInstructionConditional

Functions

convert_qiskit_circuit_to_qasm_experiment(experiment)

Attributes

_IS_QISKIT_AVAILABLE
_IS_QISKIT_AVAILABLE

pyqrack.util.convert_qiskit_circuit_to_qasm_experiment.**_IS_QISKIT_AVAILABLE = True**

pyqrack.util.convert_qiskit_circuit_to_qasm_experiment.**_IS_QISKIT_AVAILABLE = False**

class pyqrack.util.convert_qiskit_circuit_to_qasm_experiment.**QrackQasmQobjInstructionConditional**(*mask*, *val*)

pyqrack.util.convert_qiskit_circuit_to_qasm_experiment.**convert_qiskit_circuit_to_qasm_experiment**(*experim*
con-
fig=Non
header=

Package Contents

Functions

convert_qiskit_circuit_to_qasm_experiment

pyqrack.util.**convert_qiskit_circuit_to_qasm_experiment**(*experiment*, *config=None*, *header=None*)

Submodules

pyqrack.neuron_activation_fn

Module Contents

Classes

NeuronActivationFn	Enum where members are also (and must be) ints

class pyqrack.neuron_activation_fn.**NeuronActivationFn**

Bases: enum.IntEnum

Enum where members are also (and must be) ints

Sigmoid = 0

ReLU = 1

GeLU = 2

Generalized_Logistic = 3

LeakyReLU = 4

pyqrack.pauli

Module Contents

Classes

Pauli	Enum where members are also (and must be) ints

class pyqrack.pauli.**Pauli**

Bases: enum.IntEnum

Enum where members are also (and must be) ints

PauliI = 0

PauliX = 1

PauliY = 3

PauliZ = 2

pyqrack.qrack_circuit

Module Contents

Classes

QrackCircuit	Class that exposes the QCircuit class of Qrack

Attributes

_IS_QISKIT_AVAILABLE
_IS_QISKIT_AVAILABLE
_IS_QUIMB_AVAILABLE
_IS_QUIMB_AVAILABLE
_IS_TENSORCIRCUIT_AVAILABLE
_IS_TENSORCIRCUIT_AVAILABLE

pyqrack.qrack_circuit.**_IS_QISKIT_AVAILABLE = True**

pyqrack.qrack_circuit.**_IS_QISKIT_AVAILABLE = False**

pyqrack.qrack_circuit.**_IS_QUIMB_AVAILABLE = True**

pyqrack.qrack_circuit.**_IS_QUIMB_AVAILABLE = False**

pyqrack.qrack_circuit.**_IS_TENSORCIRCUIT_AVAILABLE = True**

pyqrack.qrack_circuit.**_IS_TENSORCIRCUIT_AVAILABLE = False**

class pyqrack.qrack_circuit.**QrackCircuit**(*is_collapse=True*, *clone_cid=-1*, *is_inverse=False*, *past_light_cone=[]*)

Class that exposes the QCircuit class of Qrack

QrackCircuit allows the user to specify a unitary circuit, before running it. Upon running the state, the result is a QrackSimulator state. Currently, measurement is not supported, but measurement can be run on the resultant QrackSimulator.

cid

Qrack ID of this circuit

Type
int

__del__()

_ulonglong_byref(*a*)

_double_byref(*a*)

_complex_byref(*a*)

clone()

Make a new circuit that is an exact clone of this circuit

Raises
RuntimeError – QrackCircuit C++ library raised an exception.

inverse()

Make a new circuit that is the exact inverse of this circuit

Raises
RuntimeError – QrackCircuit C++ library raised an exception.

past_light_cone(*q*)

Make a new circuit with just this circuits' past light cone for certain qubits.

Parameters
q – list of qubit indices to include at beginning of past light cone

Raises
RuntimeError – QrackCircuit C++ library raised an exception.

get_qubit_count()

Get count of qubits in circuit

Raises
RuntimeError – QrackCircuit C++ library raised an exception.

swap(*q1*, *q2*)

Add a 'Swap' gate to the circuit

Parameters

- q1 – qubit index #1
- q2 – qubit index #2

Raises
RuntimeError – QrackCircuit C++ library raised an exception.

mtrx(*m*, *q*)

Operation from matrix.

Applies arbitrary operation defined by the given matrix.

Parameters

- m – row-major complex list representing the operator.
- q – the qubit number on which the gate is applied to.

Raises

- ValueError – 2x2 matrix 'm' in QrackCircuit.mtrx() must contain at least 4 elements.
- RuntimeError – QrackSimulator raised an exception.

ucmtrx(*c*, *m*, *q*, *p*)

Multi-controlled single-target-qubit gate

Specify a controlled gate by its control qubits, its single-qubit matrix "payload," the target qubit, and the permutation of qubits that activates the gate.

Parameters

- **c** – list of controlled qubits
- **m** – row-major complex list representing the operator.
- **q** – target qubit
- **p** – permutation of target qubits

Raises

- **ValueError** – 2x2 matrix 'm' in QrackCircuit.ucmtrx() must contain at least 4 elements.
- **RuntimeError** – QrackSimulator raised an exception.

run(*qsim*)

Run circuit on simulator

Run the encoded circuit on a specific simulator. The result will remain in this simulator.

Parameters
qsim – QrackSimulator on which to run circuit

Raises
RuntimeError – QrackCircuit raised an exception.

out_to_file(*filename*)

Output optimized circuit to file

Outputs the (optimized) circuit to a file named according to the "filename" parameter.

Parameters
filename – Name of file

in_from_file()

Read in optimized circuit from file

Reads in an (optimized) circuit from a file named according to the "filename" parameter.

Parameters
filename – Name of file

file_to_qiskit_circuit()

Convert an output file to a Qiskit circuit

Reads in an (optimized) circuit from a file named according to the "filename" parameter and outputs a Qiskit circuit.

Parameters
filename – Name of file

Raises
RuntimeErorr – Before trying to file_to_qiskit_circuit() with QrackCircuit, you must install Qiskit, numpy, and math!

in_from_qiskit_circuit()

Read a Qiskit circuit into a QrackCircuit

Reads in a circuit from a Qiskit *QuantumCircuit*

Parameters
circ – Qiskit circuit

Raises

RuntimeErorr – Before trying to file_to_qiskit_circuit() with QrackCircuit, you must install Qiskit, numpy, and math!

file_to_quimb_circuit(*circuit_type=QuimbCircuitType.Circuit*, *psi0=None*, *gate_opts=None*, *tags=None*, *psi0_dtype='complex128'*, *psi0_tag='PSI0'*, *bra_site_ind_id='b{}'*)

Convert an output file to a Quimb circuit

Reads in an (optimized) circuit from a file named according to the "filename" parameter and outputs a Quimb circuit.

Parameters

- **filename** – Name of file
- **circuit_type** – "QuimbCircuitType" enum value specifying type of Quimb circuit
- **psi0** – The initial state, assumed to be |00000....0> if not given. The state is always copied and the tag PSI0 added
- **gate_opts** – Default keyword arguments to supply to each gate_TN_1D() call during the circuit
- **tags** – Tag(s) to add to the initial wavefunction tensors (whether these are propagated to the rest of the circuit's tensors
- **psi0_dtype** – Ensure the initial state has this dtype.
- **psi0_tag** – Ensure the initial state has this tag.
- **bra_site_ind_id** – Use this to label 'bra' site indices when creating certain (mostly internal) intermediate tensor networks.

Raises

RuntimeErorr – Before trying to file_to_quimb_circuit() with QrackCircuit, you must install quimb, Qiskit, numpy, and math!

file_to_tensorcircuit(*inputs=None*, *circuit_params=None*, *binding_params=None*)

Convert an output file to a TensorCircuit circuit

Reads in an (optimized) circuit from a file named according to the "filename" parameter and outputs a TensorCircuit circuit.

Parameters

- **filename** – Name of file
- **inputs** – pass-through to tensorcircuit.Circuit.from_qiskit
- **circuit_params** – pass-through to tensorcircuit.Circuit.from_qiskit
- **binding_params** – pass-through to tensorcircuit.Circuit.from_qiskit

Raises

RuntimeErorr – Before trying to file_to_quimb_circuit() with QrackCircuit, you must install TensorCircuit, Qiskit, numpy, and math!

in_from_tensorcircuit(*enable_instruction=False*, *enable_inputs=False*)

Convert a TensorCircuit circuit to a QrackCircuit

Accepts a TensorCircuit circuit and outputs an equivalent QrackCircuit

Parameters

- **tcirc** – TensorCircuit circuit

- **enable_instruction** – whether to also export measurement and reset instructions
- **enable_inputs** – whether to also export the inputs

Raises

RuntimeErorr – Before trying to in_from_tensorcircuit() with QrackCircuit, you must install TensorCircuit, Qiskit, numpy, and math!

pyqrack.qrack_neuron

Module Contents

Classes

QrackNeuron	Class that exposes the QNeuron class of Qrack

class pyqrack.qrack_neuron.**QrackNeuron**(*simulator*, *controls*, *target*, *activation_fn=NeuronActivationFn.Sigmoid*, *alpha=1.0*, *tolerance=sys.float_info.epsilon*, *_init=True*)

Class that exposes the QNeuron class of Qrack

This model of a "quantum neuron" is based on the concept of a "uniformly controlled" rotation of a single output qubit around the Pauli Y axis, and has been developed by others. In our case, the primary relevant gate could also be called a single-qubit-target multiplexer.

(See https://arxiv.org/abs/quant-ph/0407010 for an introduction to "uniformly controlled gates.)

QrackNeuron is meant to be interchangeable with a single classical neuron, as in conventional neural net software. It differs from classical neurons in conventional neural nets, in that the "synaptic cleft" is modelled as a single qubit. Hence, this neuron can train and predict in superposition.

nid

Qrack ID of this neuron

Type
int

simulator

Simulator instance for all synaptic clefts of the neuron

Type
QrackSimulator

controls

Indices of all "control" qubits, for neuron input

Type
list(int)

target

Index of "target" qubit, for neuron output

Type
int

tolerance

Rounding tolerance

Type
double

_get_error()

_throw_if_error()

__del__()

clone()

Clones this neuron.

Create a new, independent neuron instance with identical angles, inputs, output, and tolerance, for the same QrackSimulator.

Raises
RuntimeError – QrackNeuron C++ library raised an exception.

_ulonglong_byref(*a*)

_real1_byref(*a*)

set_angles(*a*)

Directly sets the neuron parameters.

Set all synaptic parameters of the neuron directly, by a list enumerated over the integer permutations of input qubits.

Parameters
a (*list(double)*) – List of input permutation angles

Raises

- **ValueError** – Angles 'a' in QrackNeuron.set_angles() must contain at least (2 ** len(self.controls)) elements.
- **RuntimeError** – QrackSimulator raised an exception.

get_angles()

Directly gets the neuron parameters.

Get all synaptic parameters of the neuron directly, as a list enumerated over the integer permutations of input qubits.

Raises
RuntimeError – QrackNeuron C++ library raised an exception.

set_alpha(*a*)

Set the neuron 'alpha' parameter.

To enable nonlinear activation, *QrackNeuron* has an 'alpha' parameter that is applied as a power to its angles, before learning and prediction. This makes the activation function sharper (or less sharp).

Raises
RuntimeError – QrackNeuron C++ library raised an exception.

`set_activation_fn`(*f*)

Sets the activation function of this QrackNeuron

Nonlinear activation functions can be important to neural net applications, like DNN. The available activation functions are enumerated in *NeuronActivationFn*.

Raises
`RuntimeError` – QrackNeuron C++ library raised an exception.

`predict`(*e=True*, *r=True*)

Predict based on training

"Predict" the anticipated output, based on input and training. By default, "predict()" will initialize the output qubit as by resetting to |0> and then acting a Hadamard gate. From that state, the method amends the output qubit upon the basis of the state of its input qubits, applying a rotation around Pauli Y axis according to the angle learned for the input.

Parameters

- `e` (*bool*) – If False, predict the opposite
- `r` (*bool*) – If True, start by resetting the output to 50/50

Raises
`RuntimeError` – QrackNeuron C++ library raised an exception.

`unpredict`(*e=True*)

Uncompute a prediction

Uncompute a 'prediction' of the anticipated output, based on input and training.

Parameters
`e` (*bool*) – If False, unpredict the opposite

Raises
`RuntimeError` – QrackNeuron C++ library raised an exception.

`learn_cycle`(*e=True*)

Run a learning cycle

A learning cycle consists of predicting a result, saving the classical outcome, and uncomputing the prediction.

Parameters
`e` (*bool*) – If False, predict the opposite

Raises
`RuntimeError` – QrackNeuron C++ library raised an exception.

`learn`(*eta*, *e=True*, *r=True*)

Learn from current qubit state

"Learn" to associate current inputs with output. Based on input qubit states and volatility 'eta,' the input state synaptic parameter is updated to prefer the "e" ("expected") output.

Parameters

- `eta` (*double*) – Training volatility, 0 to 1
- `e` (*bool*) – If False, predict the opposite
- `r` (*bool*) – If True, start by resetting the output to 50/50

Raises

RuntimeError – QrackNeuron C++ library raised an exception.

learn_permutation(*eta*, *e=True*, *r=True*)

Learn from current classical state

Learn to associate current inputs with output, under the assumption that the inputs and outputs are "classical." Based on input qubit states and volatility 'eta,' the input state angle is updated to prefer the "e" ("expected") output.

Parameters

- **eta** (*double*) – Training volatility, 0 to 1
- **e** (*bool*) – If False, predict the opposite
- **r** (*bool*) – If True, start by resetting the output to 50/50

Raises

RuntimeError – QrackNeuron C++ library raised an exception.

pyqrack.qrack_simulator

Module Contents

Classes

QrackSimulator	Interface for all the QRack functionality.

Attributes

_IS_QISKIT_AVAILABLE
_IS_QISKIT_AVAILABLE
_IS_NUMPY_AVAILABLE
_IS_NUMPY_AVAILABLE

pyqrack.qrack_simulator.**_IS_QISKIT_AVAILABLE = True**

pyqrack.qrack_simulator.**_IS_QISKIT_AVAILABLE = False**

pyqrack.qrack_simulator.**_IS_NUMPY_AVAILABLE = True**

pyqrack.qrack_simulator.**_IS_NUMPY_AVAILABLE = False**

class pyqrack.qrack_simulator.**QrackSimulator**(*qubitCount=-1*, *cloneSid=-1*, *isTensorNetwork=True*, *isSchmidtDecomposeMulti=True*, *isSchmidtDecompose=True*, *isStabilizerHybrid=True*, *isBinaryDecisionTree=False*, *isPaged=True*, *isCpuGpuHybrid=True*, *isOpenCL=True*, *isHostPointer=False*, *pyzxCircuit=None*, *qiskitCircuit=None*)

Interface for all the QRack functionality.

qubitCount

Number of qubits that are to be simulated.

Type
int

sid

Corresponding simulator id.

Type
int

_get_error()

_throw_if_error()

__del__()

_int_byref(*a*)

_ulonglong_byref(*a*)

_double_byref(*a*)

_complex_byref(*a*)

_real1_byref(*a*)

_bool_byref(*a*)

_qrack_complex_byref(*a*)

_to_ubyte(*nv*, *v*)

_to_ulonglong(*m*, *v*)

_pairwise(*it*)

seed(*s*)

set_concurrency(*p*)

x(*q*)

Applies X gate.

Applies the Pauli "X" operator to the qubit at position "q." The Pauli "X" operator is equivalent to a logical "NOT."

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

y(*q*)

Applies Y gate.

Applies the Pauli "Y" operator to the qubit at "q." The Pauli "Y" operator is equivalent to a logical "NOT" with permutation phase.

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

z(*q*)

Applies Z gate.

Applies the Pauli "Z" operator to the qubit at "q." The Pauli "Z" operator flips the phase of |1>

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

h(*q*)

Applies H gate.

Applies the Hadarmard operator to the qubit at "q."

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

s(*q*)

Applies S gate.

Applies the 1/4 phase rotation to the qubit at "q."

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

t(*q*)

Applies T gate.

Applies the 1/8 phase rotation to the qubit at "q."

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

adjs(*q*)

Adjoint of S gate

Applies the gate equivalent to the inverse of S gate.

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

`adjt`(*q*)

Adjoint of T gate

Applies the gate equivalent to the inverse of T gate.

Parameters
`q` – the qubit number on which the gate is applied to.

Raises
`RuntimeError` – QrackSimulator raised an exception.

`u`(*q*, *th*, *ph*, *la*)

General unitary gate.

Applies a gate guaranteed to be unitary. Spans all possible single bit unitary gates.

U(theta, phi, lambda) = RZ(phi + pi/2)RX(theta)RZ(lambda - pi/2)

Parameters

- `q` – the qubit number on which the gate is applied to.
- `th` – theta
- `ph` – phi
- `la` – lambda

Raises
`RuntimeError` – QrackSimulator raised an exception.

`mtrx`(*m*, *q*)

Operation from matrix.

Applies arbitrary operation defined by the given matrix.

Parameters

- `m` – row-major complex list representing the operator.
- `q` – the qubit number on which the gate is applied to.

Raises

- `ValueError` – 2x2 matrix 'm' in QrackSimulator.mtrx() must contain at least 4 elements.
- `RuntimeError` – QrackSimulator raised an exception.

`r`(*b*, *ph*, *q*)

Rotation gate.

Rotate the qubit along the given pauli basis by the given angle.

Parameters

- `b` – Pauli basis
- `ph` – rotation angle
- `q` – the qubit number on which the gate is applied to

Raises
`RuntimeError` – QrackSimulator raised an exception.

exp(*b*, *ph*, *q*)

Arbitrary exponentiation

exp(b, theta) = e^{i*theta*[b_0 . b_1 ...]} where . is the tensor product.

Parameters

- **b** – Pauli basis
- **ph** – coefficient of exponentiation
- **q** – the qubit number on which the gate is applied to

Raises

RuntimeError – QrackSimulator raised an exception.

mcx(*c*, *q*)

Multi-controlled X gate

If all controlled qubits are |*1*> then the target qubit is flipped.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

mcy(*c*, *q*)

Multi-controlled Y gate

If all controlled qubits are |*1*> then the Pauli "Y" gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

mcz(*c*, *q*)

Multi-controlled Z gate

If all controlled qubits are |*1*> then the Pauli "Z" gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

mch(*c*, *q*)

Multi-controlled H gate

If all controlled qubits are |*1*> then the Hadarmard gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.

- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

mcs(*c*, *q*)

Multi-controlled S gate

If all controlled qubits are |*1*> then the “S” gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

mct(*c*, *q*)

Multi-controlled T gate

If all controlled qubits are |*1*> then the “T” gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

mcadjs(*c*, *q*)

Multi-controlled adjs gate

If all controlled qubits are |*1*> then the adjs gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

mcadjt(*c*, *q*)

Multi-controlled adjt gate

If all controlled qubits are |*1*> then the adjt gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

mcu(*c*, *q*, *th*, *ph*, *la*)

Multi-controlled arbitraty unitary

If all controlled qubits are |*1*> then the unitary gate described by parameters is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.
- **th** – theta
- **ph** – phi
- **la** – lambda

Raises
RuntimeError – QrackSimulator raised an exception.

mcmtrx(*c*, *m*, *q*)

Multi-controlled arbitrary operator

If all controlled qubits are |*1*> then the arbitrary operation by parameters is applied to the target qubit.

Parameters

- **c** – list of controlled qubits
- **m** – row-major complex list representing the operator.
- **q** – target qubit

Raises

- **ValueError** – 2x2 matrix 'm' in QrackSimulator.mcmtrx() must contain at least 4 elements.
- **RuntimeError** – QrackSimulator raised an exception.

macx(*c*, *q*)

Anti multi-controlled X gate

If all controlled qubits are |*0*> then the target qubit is flipped.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

macy(*c*, *q*)

Anti multi-controlled Y gate

If all controlled qubits are |*0*> then the Pauli "Y" gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

macz(*c*, *q*)

Anti multi-controlled Z gate

If all controlled qubits are |*0*> then the Pauli "Z" gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

mach(*c*, *q*)

Anti multi-controlled H gate

If all controlled qubits are |*0*> then the Hadarmard gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

macs(*c*, *q*)

Anti multi-controlled S gate

If all controlled qubits are |*0*> then the “S” gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

mact(*c*, *q*)

Anti multi-controlled T gate

If all controlled qubits are |*0*> then the “T” gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

macadjs(*c*, *q*)

Anti multi-controlled adjs gate

If all controlled qubits are |*0*> then the adjs gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

macadjt(*c*, *q*)

Anti multi-controlled adjt gate

If all controlled qubits are |0> then the adjt gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

macu(*c*, *q*, *th*, *ph*, *la*)

Anti multi-controlled arbitraty unitary

If all controlled qubits are |0> then the unitary gate described by parameters is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.
- **th** – theta
- **ph** – phi
- **la** – lambda

Raises

RuntimeError – QrackSimulator raised an exception.

macmtrx(*c*, *m*, *q*)

Anti multi-controlled arbitraty operator

If all controlled qubits are |0> then the arbitrary operation by parameters is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **m** – row-major complex matrix which defines the operator.
- **q** – target qubit.

Raises

- **ValueError** – 2x2 matrix 'm' in QrackSimulator.macmtrx() must contain at least 4 elements.
- **RuntimeError** – QrackSimulator raised an exception.

ucmtrx(*c*, *m*, *q*, *p*)

Multi-controlled arbitrary operator with arbitrary controls

If all control qubits match 'p' permutation by bit order, then the arbitrary operation by parameters is applied to the target qubit.

Parameters

- **c** – list of control qubits
- **m** – row-major complex list representing the operator.
- **q** – target qubit

- **p** – permutation of list of control qubits

Raises

- **ValueError** – 2x2 matrix 'm' in QrackSimulator.ucmtrx() must contain at least 4 elements.
- **RuntimeError** – QrackSimulator raised an exception.

multiplex1_mtrx(*c*, *q*, *m*)

Multiplex gate

A multiplex gate with a single target and an arbitrary number of controls.

Parameters

- **c** – list of controlled qubits.
- **m** – row-major complex matrix which defines the operator.
- **q** – target qubit.

Raises

- **ValueError** – Multiplex matrix 'm' in QrackSimulator.multiplex1_mtrx() must contain at least 4 elements.
- **RuntimeError** – QrackSimulator raised an exception.

mx(*q*)

Multi X-gate

Applies the Pauli "X" operator on all qubits.

Parameters
q – list of qubits to apply X on.

Raises
RuntimeError – QrackSimulator raised an exception.

my(*q*)

Multi Y-gate

Applies the Pauli "Y" operator on all qubits.

Parameters
q – list of qubits to apply Y on.

Raises
RuntimeError – QrackSimulator raised an exception.

mz(*q*)

Multi Z-gate

Applies the Pauli "Z" operator on all qubits.

Parameters
q – list of qubits to apply Z on.

Raises
RuntimeError – QrackSimulator raised an exception.

mcr(*b*, *ph*, *c*, *q*)

Multi-controlled arbitrary rotation.

If all controlled qubits are |*1*> then the arbitrary rotation by parameters is applied to the target qubit.

Parameters

- **b** – Pauli basis
- **ph** – coefficient of exponentiation.
- **c** – list of controlled qubits.
- **q** – the qubit number on which the gate is applied to.

Raises

RuntimeError – QrackSimulator raised an exception.

mcexp(*b*, *ph*, *cs*, *q*)

Multi-controlled arbitrary exponentiation

If all controlled qubits are |*1*> then the target qubit is exponentiated an pauli basis basis with coefficient.

Parameters

- **b** – Pauli basis
- **ph** – coefficient of exponentiation.
- **q** – the qubit number on which the gate is applied to.

Raises

RuntimeError – QrackSimulator raised an exception.

swap(*qi1*, *qi2*)

Swap Gate

Swaps the qubits at two given positions.

Parameters

- **qi1** – First position of qubit.
- **qi2** – Second position of qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

iswap(*qi1*, *qi2*)

Swap Gate with phase.

Swaps the qubits at two given positions. If the bits are different then there is additional phase of *i*.

Parameters

- **qi1** – First position of qubit.
- **qi2** – Second position of qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

adjiswap(*qi1*, *qi2*)

Swap Gate with phase.

Swaps the qubits at two given positions. If the bits are different then there is additional phase of *-i*.

Parameters

- **qi1** – First position of qubit.
- **qi2** – Second position of qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

fsim(*th*, *ph*, *qi1*, *qi2*)

Fsim gate.

The 2-qubit "fSim" gate Useful in the simulation of particles with fermionic statistics

Parameters

- **qi1** – First position of qubit.
- **qi2** – Second position of qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

cswap(*c*, *qi1*, *qi2*)

Controlled-swap Gate

Swaps the qubits at two given positions if the control qubits are |*1*>

Parameters

- **c** – list of controlled qubits.
- **qi1** – First position of qubit.
- **qi2** – Second position of qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

acswap(*c*, *qi1*, *qi2*)

Anti controlled-swap Gate

Swaps the qubits at two given positions if the control qubits are |*0*>

Parameters

- **c** – list of controlled qubits.
- **qi1** – First position of qubit.
- **qi2** – Second position of qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

m(*q*)

Measurement gate

Measures the qubit at "q" and returns Boolean value. This operator is not unitary & is probabilistic in nature.

Parameters
q – qubit to measure

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Measurement result.

`force_m`(*q*, *r*)

Force-Measurement gate

Acts as if the measurement is applied and the result obtained is *r*

Parameters

- **q** – qubit to measure
- **r** – the required result

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Measurement result.

`m_all`()

Measure-all gate

Measures measures all qubits. This operator is not unitary & is probabilistic in nature.

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Measurement result of all qubits.

`measure_pauli`(*b*, *q*)

Pauli Measurement gate

Measures the qubit at "q" with the given pauli basis. This operator is not unitary & is probabilistic in nature.

Parameters

- **b** – Pauli basis
- **q** – qubit to measure

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Measurement result.

`measure_shots`(*q*, *s*)

Multi-shot measurement operator

Measures the qubit at "q" with the given pauli basis. This operator is not unitary & is probabilistic in nature.

Parameters

- **q** – list of qubits to measure
- **s** – number of shots

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
list of measurement result.

`reset_all`()

Reset gate

Resets all qubits to |*0*>

Raises
`RuntimeError` – QrackSimulator raised an exception.

`_split_longs`(*a*)

Split operation

Splits the given integer into 64 bit numbers.

Parameters
`a` – number to split

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
list of split numbers.

`_split_longs_2`(*a*, *m*)

Split simultanoues operation

Splits 2 integers into same number of 64 bit numbers.

Parameters

- `a` – first number to split
- `m` – second number to split

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
pair of lists of split numbers.

`add`(*a*, *q*)

Add integer to qubit

Adds the given integer to the given set of qubits.

Parameters

- `a` – first number to split
- `q` – list of qubits to add the number

Raises
`RuntimeError` – QrackSimulator raised an exception.

`sub`(*a*, *q*)

Subtract integer to qubit

Subtracts the given integer to the given set of qubits.

Parameters

- `a` – first number to split
- `q` – list of qubits to subtract the number

Raises
`RuntimeError` – QrackSimulator raised an exception.

`adds`(*a*, *s*, *q*)

Signed Addition integer to qubit

Signed Addition of the given integer to the given set of qubits, if there is an overflow the resultant will become negative.

Parameters

- **a** – number to add
- **s** – qubit to store overflow
- **q** – list of qubits to add the number

Raises

RuntimeError – QrackSimulator raised an exception.

`subs`(*a*, *s*, *q*)

Subtract integer to qubit

Subtracts the given integer to the given set of qubits, if there is an overflow the resultant will become negative.

Parameters

- **a** – number to subtract
- **s** – qubit to store overflow
- **q** – list of qubits to subtract the number

Raises

RuntimeError – QrackSimulator raised an exception.

`mul`(*a*, *q*, *o*)

Multiplies integer to qubit

Multiplies the given integer to the given set of qubits. Carry register is required for maintaining the unitary nature of operation and must be as long as the input qubit register.

Parameters

- **a** – number to multiply
- **q** – list of qubits to multiply the number
- **o** – carry register

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot mul()! (Turn off just this option, in the constructor.)

`div`(*a*, *q*, *o*)

Divides qubit by integer

'Divides' the given qubits by the integer. (This is rather the adjoint of mul().) Carry register is required for maintaining the unitary nature of operation.

Parameters

- **a** – integer to divide by
- **q** – qubits to divide

- **o** – carry register

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot div()! (Turn off just this option, in the constructor.)

muln(*a*, *m*, *q*, *o*)

Modulo Multiplication

Modulo Multiplication of the given integer to the given set of qubits Out-of-place register is required to store the resultant.

Parameters

- **a** – number to multiply
- **m** – modulo number
- **q** – list of qubits to multiply the number
- **o** – carry register

Raises

RuntimeError – QrackSimulator raised an exception.

divn(*a*, *m*, *q*, *o*)

Modulo Division

'Modulo Division' of the given set of qubits by the given integer (This is rather the adjoint of muln().) Out-of-place register is required to retrieve the resultant.

Parameters

- **a** – integer by which qubit will be divided
- **m** – modulo integer
- **q** – qubits to divide
- **o** – carry register

Raises

RuntimeError – QrackSimulator raised an exception.

pown(*a*, *m*, *q*, *o*)

Modulo Power

Raises the qubit to the power *a* to which *mod m* is applied to. Out-of-place register is required to store the resultant.

Parameters

- **a** – number in power
- **m** – modulo number
- **q** – list of qubits to exponentiate
- **o** – out-of-place register

Raises

- **RuntimeError** – QrackSimulator raised an exception.

- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot pown()! (Turn off just this option, in the constructor.)

mcadd(*a*, *c*, *q*)

Controlled-add

Adds the given integer to the given set of qubits if all controlled qubits are |*1*>.

Parameters

- **a** – number to add.
- **c** – list of controlled qubits.
- **q** – list of qubits to add the number

Raises

RuntimeError – QrackSimulator raised an exception.

mcsub(*a*, *c*, *q*)

Controlled-subtract

Subtracts the given integer to the given set of qubits if all controlled qubits are |*1*>.

Parameters

- **a** – number to subtract.
- **c** – list of controlled qubits.
- **q** – list of qubits to add the number

Raises

RuntimeError – QrackSimulator raised an exception.

mcmul(*a*, *c*, *q*, *o*)

Controlled-multiply

Multiplies the given integer to the given set of qubits if all controlled qubits are |*1*>. Carry register is required for maintaining the unitary nature of operation.

Parameters

- **a** – number to multiply
- **c** – list of controlled qubits.
- **q** – list of qubits to add the number
- **o** – carry register

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot mcmul()! (Turn off just this option, in the constructor.)

mcdiv(*a*, *c*, *q*, *o*)

Controlled-divide.

'Divides' the given qubits by the integer if all controlled qubits are |*1*>. (This is rather the adjoint of mcmul().) Carry register is required for maintaining the unitary nature of operation.

Parameters

- **a** – number to divide by

- **c** – list of controlled qubits.
- **q** – qubits to divide
- **o** – carry register

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot mcdiv()! (Turn off just this option, in the constructor.)

mcmuln(*a*, *c*, *m*, *q*, *o*)

Controlled-modulo multiplication

Modulo multiplication of the given integer to the given set of qubits if all controlled qubits are |*1*>. Out-of-place register is required to store the resultant.

Parameters

- **a** – number to multiply
- **c** – list of controlled qubits.
- **m** – modulo number
- **q** – list of qubits to add the number
- **o** – out-of-place output register

Raises

RuntimeError – QrackSimulator raised an exception.

mcdivn(*a*, *c*, *m*, *q*, *o*)

Controlled-divide.

Modulo division of the given qubits by the given number if all controlled qubits are |*1*>. (This is rather the adjoint of mcmuln().) Out-of-place register is required to retrieve the resultant.

Parameters

- **a** – number to divide by
- **c** – list of controlled qubits.
- **m** – modulo number
- **q** – qubits to divide
- **o** – carry register

Raises

RuntimeError – QrackSimulator raised an exception.

mcpown(*a*, *c*, *m*, *q*, *o*)

Controlled-modulo Power

Raises the qubit to the power *a* to which *mod m* is applied to if all the controlled qubits are set to |*1*>. Out-of-place register is required to store the resultant.

Parameters

- **a** – number in power
- **c** – control qubits
- **m** – modulo number

- **q** – list of qubits to exponentiate
- **o** – out-of-place register

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot mcpown()! (Turn off just this option, in the constructor.)

lda(*qi*, *qv*, *t*)

Load Accumalator

Quantum counterpart for LDA from MOS-6502 assembly. *t* must be of the length *2 ** len(qi)*. It loads each list entry index of t into the qi register and each list entry value into the qv register.

Parameters

- **qi** – qubit register for index
- **qv** – qubit register for value
- **t** – list of values

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot lda()! (Turn off just this option, in the constructor.)

adc(*s*, *qi*, *qv*, *t*)

Add with Carry

Quantum counterpart for ADC from MOS-6502 assembly. *t* must be of the length *2 ** len(qi)*.

Parameters

- **qi** – qubit register for index
- **qv** – qubit register for value
- **t** – list of values

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot adc()! (Turn off just this option, in the constructor.)

sbc(*s*, *qi*, *qv*, *t*)

Subtract with Carry

Quantum counterpart for SBC from MOS-6502 assembly. *t* must be of the length *2 ** len(qi)*

Parameters

- **qi** – qubit register for index
- **qv** – qubit register for value
- **t** – list of values

Raises

- **RuntimeError** – QrackSimulator raised an exception.

- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot sbc()! (Turn off just this option, in the constructor.)

hash(*q*, *t*)

Hash function

Replicates the behaviour of LDA without the index register. For the operation to be unitary, the entries present in *t* must be unique, and the length of *t* must be *2 ** len(qi)*.

Parameters

- **q** – qubit register for value
- **t** – list of values

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot hash()! (Turn off just this option, in the constructor.)

qand(*qi1*, *qi2*, *qo*)

Logical AND

Logical AND of 2 qubits whose result is stored in the target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

qor(*qi1*, *qi2*, *qo*)

Logical OR

Logical OR of 2 qubits whose result is stored in the target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

qxor(*qi1*, *qi2*, *qo*)

Logical XOR

Logical exlusive-OR of 2 qubits whose result is stored in the target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises
RuntimeError – QrackSimulator raised an exception.

qnand(*qi1*, *qi2*, *qo*)

Logical NAND

Logical NAND of 2 qubits whose result is stored in the target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises
RuntimeError – QrackSimulator raised an exception.

qnor(*qi1*, *qi2*, *qo*)

Logical NOR

Logical NOR of 2 qubits whose result is stored in the target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises
RuntimeError – QrackSimulator raised an exception.

qxnor(*qi1*, *qi2*, *qo*)

Logical XOR

Logical exlusive-NOR of 2 qubits whose result is stored in the target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises
RuntimeError – QrackSimulator raised an exception.

cland(*ci*, *qi*, *qo*)

Classical AND

Logical AND with one qubit and one classical bit whose result is stored in target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises
RuntimeError – QrackSimulator raised an exception.

`clor`(*ci*, *qi*, *qo*)

Classical OR

Logical OR with one qubit and one classical bit whose result is stored in target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

`clxor`(*ci*, *qi*, *qo*)

Classical XOR

Logical exlusive-OR with one qubit and one classical bit whose result is stored in target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

`clnand`(*ci*, *qi*, *qo*)

Classical NAND

Logical NAND with one qubit and one classical bit whose result is stored in target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

`clnor`(*ci*, *qi*, *qo*)

Classical NOR

Logical NOR with one qubit and one classical bit whose result is stored in target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

`clxnor`(*ci*, *qi*, *qo*)

Classical XNOR

Logical exlusive-NOR with one qubit and one classical bit whose result is stored in target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

`qft`(*qs*)

Quantum Fourier Transform

Applies Quantum Fourier Transform on the list of qubits provided.

Parameters

qs – list of qubits

Raises

RuntimeError – QrackSimulator raised an exception.

`iqft`(*qs*)

Inverse-quantum Fourier Transform

Applies Inverse-quantum Fourier Transform on the list of qubits provided.

Parameters

qs – list of qubits

Raises

RuntimeError – QrackSimulator raised an exception.

`allocate_qubit`(*qid*)

Allocate Qubit

Allocate 1 new qubit with the given qubit ID.

Parameters

qid – qubit id

Raises

RuntimeError – QrackSimulator raised an exception.

`release`(*q*)

Release Qubit

Release qubit given by the given qubit ID.

Parameters

q – qubit id

Raises

RuntimeError – QrackSimulator raised an exception.

Returns

If the qubit was in |*0*> state with small tolerance.

num_qubits()

Get Qubit count

Returns the qubit count of the simulator.

Parameters
q – qubit id

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Qubit count of the simulator

compose(*other*, *q*)

Compose qubits

Compose quantum description of given qubit with the current system.

Parameters
q – qubit id

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot compose()! (Turn off just this option, in the constructor.)

decompose(*q*)

Decompose system

Decompose the given qubit out of the system. Warning: The qubit subsystem state must be separable, or the behavior of this method is undefined.

Parameters
q – qubit id

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot decompose()! (Turn off just this option, in the constructor.)

Returns
State of the systems.

dispose(*q*)

Dispose qubits

Minimally decompose a set of contiguous bits from the separably composed unit, and discard the separable bits. Warning: The qubit subsystem state must be separable, or the behavior of this method is undefined.

Parameters
q – qubit

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot dispose()! (Turn off just this option, in the constructor.)

Returns
State of the systems.

`dump_ids()`

Dump all IDs

Dump all IDs from the selected simulator ID into the callback.

Returns
List of ids

`dump_ids_callback()`

C callback function

`dump()`

Dump state vector

Dump state vector from the selected simulator ID into the callback.

Returns
State vector list

`dump_callback(i)`

C callback function

`in_ket(ket)`

Set state vector

Set state vector for the selected simulator ID. Warning: State vector is not always the internal representation leading to sub-optimal performance of the method.

Parameters
`ket` – the state vector to which simulator will be set

Raises
`RuntimeError` – QrackSimulator raised an exception.

`out_ket()`

Set state vector

Returns the raw state vector of the simulator. Warning: State vector is not always the internal representation leading to sub-optimal performance of the method.

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
list representing the state vector.

`prob(q)`

Probability of |*1*>

Get the probability that a qubit is in the |*1*> state.

Parameters
`q` – qubit id

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
probability of qubit being in |*1*>

`prob_rdm`(*q*)

Probability of |*1*>, (tracing out the reduced density matrix without stabilizer ancillary qubits)

Get the probability that a qubit is in the |*1*> state.

Parameters
`q` – qubit id

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
probability of qubit being in |*1*>

`prob_perm`(*q*, *c*)

Probability of permutation

Get the probability that the qubit IDs in "q" have the truth values in "c", directly corresponding by list index.

Parameters

- `q` – list of qubit ids
- `c` – list of qubit truth values bools

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
probability that each qubit in "q[i]" has corresponding truth value in "c[i]", at once

`prob_perm_rdm`(*q*, *c*, *r=True*)

Probability of permutation, (tracing out the reduced density matrix without stabilizer ancillary qubits)

Get the probability that the qubit IDs in "q" have the truth values in "c", directly corresponding by list index.

Parameters

- `q` – list of qubit ids
- `c` – list of qubit truth values bools
- `r` – round Rz gates down from T^(1/2)

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
probability that each qubit in "q[i]" has corresponding truth value in "c[i]", at once

`permutation_expectation`(*q*)

Permutation expectation value

Get the permutation expectation value, based upon the order of input qubits.

Parameters
`q` – qubits, from low to high

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
Expectation value

permutation_expectation_rdm(*q*, *r=True*)

Permutation expectation value, (tracing out the reduced density matrix without stabilizer ancillary qubits)

Get the permutation expectation value, based upon the order of input qubits.

Parameters

- **q** – qubits, from low to high
- **r** – round Rz gates down from T^(1/2)

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Expectation value

factorized_expectation(*q*, *c*)

Factorized expectation value

Get the factorized expectation value, where each entry in "c" is an expectation value for corresponding "q" being false, then true, repeated for each in "q".

Parameters

- **q** – qubits, from low to high
- **c** – qubit falsey/truthy values, from low to high

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Expectation value

factorized_expectation_rdm(*q*, *c*, *r=True*)

Factorized expectation value, (tracing out the reduced density matrix without stabilizer ancillary qubits)

Get the factorized expectation value, where each entry in "c" is an expectation value for corresponding "q" being false, then true, repeated for each in "q".

Parameters

- **q** – qubits, from low to high
- **c** – qubit falsey/truthy values, from low to high
- **r** – round Rz gates down from T^(1/2)

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Expectation value

factorized_expectation_fp(*q*, *c*)

Factorized expectation value (floating-point)

Get the factorized expectation value, where each entry in "c" is an expectation value for corresponding "q" being false, then true, repeated for each in "q".

Parameters

- **q** – qubits, from low to high
- **c** – qubit falsey/truthy values, from low to high

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Expectation value

factorized_expectation_fp_rdm(*q*, *c*, *r=True*)

Factorized expectation value, (tracing out the reduced density matrix without stabilizer ancillary qubits)

Get the factorized expectation value, where each entry in "c" is an expectation value for corresponding "q" being false, then true, repeated for each in "q".

Parameters

- **q** – qubits, from low to high
- **c** – qubit falsey/truthy values, from low to high
- **r** – round Rz gates down from T^(1/2)

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Expectation value

joint_ensemble_probability(*b*, *q*)

Ensemble probability

Find the joint probability for all specified qubits under the respective Pauli basis transformations.

Parameters

- **b** – pauli basis
- **q** – specified qubits

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Expectation value

phase_parity(*la*, *q*)

Phase to odd parity

Applies *e^(i*la)* phase factor to all combinations of bits with odd parity, based upon permutations of qubits.

Parameters

- **la** – phase
- **q** – specified qubits

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot phase_parity()! (Turn off just this option, in the constructor.)

try_separate_1qb(*qi1*)

Manual seperation

Exposes manual control for schmidt decomposition which attempts to decompose the qubit with possible performance improvement

Parameters
qi1 – qubit to be decomposed

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
State of the qubit.

try_separate_2qb(*qi1*, *qi2*)

Manual two-qubits seperation

two-qubits counterpart of *try_separate_1qb*.

Parameters

- **qi1** – first qubit to be decomposed
- **qi2** – second qubit to be decomposed

Raises
Runtimeerror – QrackSimulator raised an exception.

Returns
State of both the qubits.

try_separate_tolerance(*qs*, *t*)

Manual multi-qubits seperation

Multi-qubits counterpart of *try_separate_1qb*.

Parameters

- **qs** – list of qubits to be decomposed
- **t** – allowed tolerance

Raises
Runtimeerror – QrackSimulator raised an exception.

Returns
State of all the qubits.

get_unitary_fidelity()

Get fidelity estimate

When using "Schmidt decomposition rounding parameter" ("SDRP") approximate simulation, QrackSimulator() can make an excellent estimate of its overall fidelity at any time, tested against a nearest-neighbor variant of quantum volume circuits.

Resetting the fidelity calculation to 1.0 happens automatically when calling *mall* are can be done manually with *reset_unitary_fidelity()*.

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Fidelity estimate

reset_unitary_fidelity()

Reset fidelity estimate

When using "Schmidt decomposition rounding parameter" ("SDRP") approximate simulation, QrackSimulator() can make an excellent estimate of its overall fidelity at any time, tested against a nearest-neighbor variant of quantum volume circuits.

Resetting the fidelity calculation to 1.0 happens automatically when calling *m_all* or can be done manually with *reset_unitary_fidelity()*.

Raises
RuntimeError – QrackSimulator raised an exception.

set_sdrp(*sdrp*)

Set "Schmidt decomposition rounding parameter"

When using "Schmidt decomposition rounding parameter" ("SDRP") approximate simulation, QrackSimulator() can make an excellent estimate of its overall fidelity at any time, tested against a nearest-neighbor variant of quantum volume circuits.

Resetting the fidelity calculation to 1.0 happens automatically when calling *m_all* or can be done manually with *reset_unitary_fidelity()*.

Raises
RuntimeError – QrackSimulator raised an exception.

set_reactive_separate(*irs*)

Set reactive separation option

If reactive separation is available, then this method turns it off/on. Note that reactive separation is on by default.

Parameters
irs – is aggresively separable

Raises
RuntimeError – QrackSimulator raised an exception.

set_t_injection(*iti*)

Set t-injection option

If t-injection is available, then this method turns it off/on. Note that t-injection is on by default.

Parameters
iti – use "reverse t-injection gadget"

Raises
RuntimeError – QrackSimulator raised an exception.

out_to_file(*filename*)

Output state to file (stabilizer only!)

Outputs the hybrid stabilizer state to file.

Parameters
filename – Name of file

in_from_file(*is_binary_decision_tree=False*, *is_paged=True*, *is_cpu_gpu_hybrid=True*, *is_opencl=True*, *is_host_pointer=False*)

Input state from file (stabilizer only!)

Reads in a hybrid stabilizer state from file.

Parameters
filename – Name of file

file_to_qiskit_circuit(*is_hardware_encoded=False*)

Convert an output state file to a Qiskit circuit

Reads in an (optimized) circuit from a file named according to the "filename" parameter and outputs a Qiskit circuit.

Parameters
filename – Name of file

Raises
RuntimeErorr – Before trying to file_to_qiskit_circuit() with QrackCircuit, you must install Qiskit, numpy, and math!

file_to_optimized_qiskit_circuit()

Convert an output state file to a Qiskit circuit

Reads in a circuit from a file named according to the "filename" parameter and outputs a 'hyper-optimized' Qiskit circuit that favors maximum reduction in gate count and depth at the potential expense of additional non-Clifford gates. (Ancilla qubits are left included in the output, though they probably have no gates.)

Parameters
filename – Name of file

Raises
RuntimeErorr – Before trying to file_to_qiskit_circuit() with QrackCircuit, you must install Qiskit, numpy, and math!

_apply_pyzx_op(*gate*)

run_pyzx_gates(*gates*)

PYZX Gates

Converts PYZX gates to *QRackSimulator* and immediately executes them.

Parameters
gates – list of PYZX gates

Raises
RuntimeError – QrackSimulator raised an exception.

_apply_op(*operation*)

_add_sample_measure(*sample_qubits*, *sample_clbits*, *num_samples*)

Generate data samples from current statevector.

Taken almost straight from the terra source code.

Parameters

- **measure_params** (*list*) – List of (qubit, clbit) values for measure instructions to sample.
- **num_samples** (*int*) – The number of data samples to generate.

Returns
A list of data values in hex format.

Return type
list

run_qiskit_circuit(*experiment*, *shots=1*)

pyqrack.quimb_circuit_type

Module Contents

Classes

QuimbCircuitType	Enum where members are also (and must be) ints

class pyqrack.quimb_circuit_type.**QuimbCircuitType**

Bases: enum.IntEnum

Enum where members are also (and must be) ints

Circuit = 0

CircuitDense = 1

CircuitMPS = 3

Package Contents

Classes

QrackSystem	
QrackSimulator	Interface for all the QRack functionality.
QrackNeuron	Class that exposes the QNeuron class of Qrack
QrackCircuit	Class that exposes the QCircuit class of Qrack
Pauli	Enum where members are also (and must be) ints
NeuronActivationFn	Enum where members are also (and must be) ints
QuimbCircuitType	Enum where members are also (and must be) ints

Attributes

Qrack	

class pyqrack.**QrackSystem**

pyqrack.**Qrack**

class pyqrack.**QrackSimulator**(*qubitCount=-1*, *cloneSid=-1*, *isTensorNetwork=True*, *isSchmidtDecomposeMulti=True*, *isSchmidtDecompose=True*, *isStabilizerHybrid=True*, *isBinaryDecisionTree=False*, *isPaged=True*, *isCpuGpuHybrid=True*, *isOpenCL=True*, *isHostPointer=False*, *pyzxCircuit=None*, *qiskitCircuit=None*)

Interface for all the QRack functionality.

qubitCount

Number of qubits that are to be simulated.

Type
int

sid

Corresponding simulator id.

Type
int

_get_error()

_throw_if_error()

__del__()

_int_byref(*a*)

_ulonglong_byref(*a*)

_double_byref(*a*)

_complex_byref(*a*)

_real1_byref(*a*)

_bool_byref(*a*)

_qrack_complex_byref(*a*)

_to_ubyte(*nv*, *v*)

_to_ulonglong(*m*, *v*)

_pairwise(*it*)

seed(*s*)

set_concurrency(*p*)

x(*q*)

Applies X gate.

Applies the Pauli "X" operator to the qubit at position "q." The Pauli "X" operator is equivalent to a logical "NOT."

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

y(*q*)

Applies Y gate.

Applies the Pauli "Y" operator to the qubit at "q." The Pauli "Y" operator is equivalent to a logical "NOT" with permutation phase.

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

z(*q*)

Applies Z gate.

Applies the Pauli "Z" operator to the qubit at "q." The Pauli "Z" operator flips the phase of |1>

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

h(*q*)

Applies H gate.

Applies the Hadarmard operator to the qubit at "q."

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

s(*q*)

Applies S gate.

Applies the 1/4 phase rotation to the qubit at "q."

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

t(*q*)

Applies T gate.

Applies the 1/8 phase rotation to the qubit at "q."

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

adjs(*q*)

Adjoint of S gate

Applies the gate equivalent to the inverse of S gate.

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

adjt(*q*)

Adjoint of T gate

Applies the gate equivalent to the inverse of T gate.

Parameters
q – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

u(*q*, *th*, *ph*, *la*)

General unitary gate.

Applies a gate guaranteed to be unitary. Spans all possible single bit unitary gates.

U(theta, phi, lambda) = RZ(phi + pi/2)RX(theta)RZ(lambda - pi/2)

Parameters

- **q** – the qubit number on which the gate is applied to.
- **th** – theta
- **ph** – phi
- **la** – lambda

Raises
RuntimeError – QrackSimulator raised an exception.

mtrx(*m*, *q*)

Operation from matrix.

Applies arbitrary operation defined by the given matrix.

Parameters

- **m** – row-major complex list representing the operator.
- **q** – the qubit number on which the gate is applied to.

Raises

- **ValueError** – 2x2 matrix 'm' in QrackSimulator.mtrx() must contain at least 4 elements.
- **RuntimeError** – QrackSimulator raised an exception.

r(*b*, *ph*, *q*)

Rotation gate.

Rotate the qubit along the given pauli basis by the given angle.

Parameters

- **b** – Pauli basis
- **ph** – rotation angle
- **q** – the qubit number on which the gate is applied to

Raises
RuntimeError – QrackSimulator raised an exception.

exp(*b*, *ph*, *q*)

Arbitrary exponentiation

exp(b, theta) = e^{i*theta*[b_0 . b_1 ...]} where . is the tensor product.

Parameters

- **b** – Pauli basis

- **ph** – coefficient of exponentiation
- **q** – the qubit number on which the gate is applied to

Raises
RuntimeError – QrackSimulator raised an exception.

mcx(*c*, *q*)

Multi-controlled X gate

If all controlled qubits are |*1*> then the target qubit is flipped.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

mcy(*c*, *q*)

Multi-controlled Y gate

If all controlled qubits are |*1*> then the Pauli "Y" gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

mcz(*c*, *q*)

Multi-controlled Z gate

If all controlled qubits are |*1*> then the Pauli "Z" gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

mch(*c*, *q*)

Multi-controlled H gate

If all controlled qubits are |*1*> then the Hadarmard gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

`mcs`(*c*, *q*)

Multi-controlled S gate

If all controlled qubits are |*1*> then the “S” gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

`mct`(*c*, *q*)

Multi-controlled T gate

If all controlled qubits are |*1*> then the “T” gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

`mcadjs`(*c*, *q*)

Multi-controlled adjs gate

If all controlled qubits are |*1*> then the adjs gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

`mcadjt`(*c*, *q*)

Multi-controlled adjt gate

If all controlled qubits are |*1*> then the adjt gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

`mcu`(*c*, *q*, *th*, *ph*, *la*)

Multi-controlled arbitraty unitary

If all controlled qubits are |*1*> then the unitary gate described by parameters is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

- **th** – theta
- **ph** – phi
- **la** – lambda

Raises
RuntimeError – QrackSimulator raised an exception.

mcmtrx(*c*, *m*, *q*)

Multi-controlled arbitrary operator

If all controlled qubits are |*1*> then the arbitrary operation by parameters is applied to the target qubit.

Parameters

- **c** – list of controlled qubits
- **m** – row-major complex list representing the operator.
- **q** – target qubit

Raises

- **ValueError** – 2x2 matrix 'm' in QrackSimulator.mcmtrx() must contain at least 4 elements.
- **RuntimeError** – QrackSimulator raised an exception.

macx(*c*, *q*)

Anti multi-controlled X gate

If all controlled qubits are |*0*> then the target qubit is flipped.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

macy(*c*, *q*)

Anti multi-controlled Y gate

If all controlled qubits are |*0*> then the Pauli "Y" gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

macz(*c*, *q*)

Anti multi-controlled Z gate

If all controlled qubits are |*0*> then the Pauli "Z" gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

mach(*c*, *q*)

Anti multi-controlled H gate

If all controlled qubits are |0> then the Hadarmard gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

macs(*c*, *q*)

Anti multi-controlled S gate

If all controlled qubits are |0> then the "S" gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

mact(*c*, *q*)

Anti multi-controlled T gate

If all controlled qubits are |0> then the "T" gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

macadjs(*c*, *q*)

Anti multi-controlled adjs gate

If all controlled qubits are |0> then the adjs gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

macadjt(*c*, *q*)

Anti multi-controlled adjt gate

If all controlled qubits are |0> then the adjt gate is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.

- **q** – target qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

macu(*c*, *q*, *th*, *ph*, *la*)

Anti multi-controlled arbitraty unitary

If all controlled qubits are |0> then the unitary gate described by parameters is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **q** – target qubit.
- **th** – theta
- **ph** – phi
- **la** – lambda

Raises
RuntimeError – QrackSimulator raised an exception.

macmtrx(*c*, *m*, *q*)

Anti multi-controlled arbitraty operator

If all controlled qubits are |0> then the arbitrary operation by parameters is applied to the target qubit.

Parameters

- **c** – list of controlled qubits.
- **m** – row-major complex matrix which defines the operator.
- **q** – target qubit.

Raises

- **ValueError** – 2x2 matrix ‘m’ in QrackSimulator.macmtrx() must contain at least 4 elements.
- **RuntimeError** – QrackSimulator raised an exception.

ucmtrx(*c*, *m*, *q*, *p*)

Multi-controlled arbitrary operator with arbitrary controls

If all control qubits match ‘p’ permutation by bit order, then the arbitrary operation by parameters is applied to the target qubit.

Parameters

- **c** – list of control qubits
- **m** – row-major complex list representing the operator.
- **q** – target qubit
- **p** – permutation of list of control qubits

Raises

- **ValueError** – 2x2 matrix ‘m’ in QrackSimulator.ucmtrx() must contain at least 4 elements.
- **RuntimeError** – QrackSimulator raised an exception.

multiplex1_mtrx(*c*, *q*, *m*)

Multiplex gate

A multiplex gate with a single target and an arbitrary number of controls.

Parameters

- **c** – list of controlled qubits.
- **m** – row-major complex matrix which defines the operator.
- **q** – target qubit.

Raises

- **ValueError** – Multiplex matrix 'm' in QrackSimulator.multiplex1_mtrx() must contain at least 4 elements.
- **RuntimeError** – QrackSimulator raised an exception.

mx(*q*)

Multi X-gate

Applies the Pauli "X" operator on all qubits.

Parameters
q – list of qubits to apply X on.

Raises
RuntimeError – QrackSimulator raised an exception.

my(*q*)

Multi Y-gate

Applies the Pauli "Y" operator on all qubits.

Parameters
q – list of qubits to apply Y on.

Raises
RuntimeError – QrackSimulator raised an exception.

mz(*q*)

Multi Z-gate

Applies the Pauli "Z" operator on all qubits.

Parameters
q – list of qubits to apply Z on.

Raises
RuntimeError – QrackSimulator raised an exception.

mcr(*b*, *ph*, *c*, *q*)

Multi-controlled arbitrary rotation.

If all controlled qubits are |*1*> then the arbitrary rotation by parameters is applied to the target qubit.

Parameters

- **b** – Pauli basis
- **ph** – coefficient of exponentiation.
- **c** – list of controlled qubits.

- **q** – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

mcexp(*b*, *ph*, *cs*, *q*)

Multi-controlled arbitrary exponentiation

If all controlled qubits are |*1*> then the target qubit is exponentiated an pauli basis basis with coefficient.

Parameters

- **b** – Pauli basis
- **ph** – coefficient of exponentiation.
- **q** – the qubit number on which the gate is applied to.

Raises
RuntimeError – QrackSimulator raised an exception.

swap(*qi1*, *qi2*)

Swap Gate

Swaps the qubits at two given positions.

Parameters

- **qi1** – First position of qubit.
- **qi2** – Second position of qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

iswap(*qi1*, *qi2*)

Swap Gate with phase.

Swaps the qubits at two given positions. If the bits are different then there is additional phase of *i*.

Parameters

- **qi1** – First position of qubit.
- **qi2** – Second position of qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

adjiswap(*qi1*, *qi2*)

Swap Gate with phase.

Swaps the qubits at two given positions. If the bits are different then there is additional phase of *-i*.

Parameters

- **qi1** – First position of qubit.
- **qi2** – Second position of qubit.

Raises
RuntimeError – QrackSimulator raised an exception.

`fsim`(*th*, *ph*, *qi1*, *qi2*)

Fsim gate.

The 2-qubit "fSim" gate Useful in the simulation of particles with fermionic statistics

Parameters

- **qi1** – First position of qubit.
- **qi2** – Second position of qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

`cswap`(*c*, *qi1*, *qi2*)

Controlled-swap Gate

Swaps the qubits at two given positions if the control qubits are *|1>*

Parameters

- **c** – list of controlled qubits.
- **qi1** – First position of qubit.
- **qi2** – Second position of qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

`acswap`(*c*, *qi1*, *qi2*)

Anti controlled-swap Gate

Swaps the qubits at two given positions if the control qubits are *|0>*

Parameters

- **c** – list of controlled qubits.
- **qi1** – First position of qubit.
- **qi2** – Second position of qubit.

Raises

RuntimeError – QrackSimulator raised an exception.

`m`(*q*)

Measurement gate

Measures the qubit at "q" and returns Boolean value. This operator is not unitary & is probabilistic in nature.

Parameters

q – qubit to measure

Raises

RuntimeError – QrackSimulator raised an exception.

Returns

Measurement result.

`force_m`(*q*, *r*)

Force-Measurement gate

Acts as if the measurement is applied and the result obtained is *r*

Parameters

- **q** – qubit to measure
- **r** – the required result

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Measurement result.

m_all()

Measure-all gate

Measures measures all qubits. This operator is not unitary & is probabilistic in nature.

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Measurement result of all qubits.

measure_pauli(*b*, *q*)

Pauli Measurement gate

Measures the qubit at "q" with the given pauli basis. This operator is not unitary & is probabilistic in nature.

Parameters

- **b** – Pauli basis
- **q** – qubit to measure

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Measurement result.

measure_shots(*q*, *s*)

Multi-shot measurement operator

Measures the qubit at "q" with the given pauli basis. This operator is not unitary & is probabilistic in nature.

Parameters

- **q** – list of qubits to measure
- **s** – number of shots

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
list of measurement result.

reset_all()

Reset gate

Resets all qubits to |*0*>

Raises
RuntimeError – QrackSimulator raised an exception.

`_split_longs`(*a*)

Split operation

Splits the given integer into 64 bit numbers.

Parameters
`a` – number to split

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
list of split numbers.

`_split_longs_2`(*a*, *m*)

Split simultanoues operation

Splits 2 integers into same number of 64 bit numbers.

Parameters

- `a` – first number to split
- `m` – second number to split

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
pair of lists of split numbers.

`add`(*a*, *q*)

Add integer to qubit

Adds the given integer to the given set of qubits.

Parameters

- `a` – first number to split
- `q` – list of qubits to add the number

Raises
`RuntimeError` – QrackSimulator raised an exception.

`sub`(*a*, *q*)

Subtract integer to qubit

Subtracts the given integer to the given set of qubits.

Parameters

- `a` – first number to split
- `q` – list of qubits to subtract the number

Raises
`RuntimeError` – QrackSimulator raised an exception.

`adds`(*a*, *s*, *q*)

Signed Addition integer to qubit

Signed Addition of the given integer to the given set of qubits, if there is an overflow the resultant will become negative.

Parameters

- **a** – number to add
- **s** – qubit to store overflow
- **q** – list of qubits to add the number

Raises
RuntimeError – QrackSimulator raised an exception.

subs(*a*, *s*, *q*)

Subtract integer to qubit

Subtracts the given integer to the given set of qubits, if there is an overflow the resultant will become negative.

Parameters

- **a** – number to subtract
- **s** – qubit to store overflow
- **q** – list of qubits to subtract the number

Raises
RuntimeError – QrackSimulator raised an exception.

mul(*a*, *q*, *o*)

Multiplies integer to qubit

Multiplies the given integer to the given set of qubits. Carry register is required for maintaining the unitary nature of operation and must be as long as the input qubit register.

Parameters

- **a** – number to multiply
- **q** – list of qubits to multiply the number
- **o** – carry register

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot mul()! (Turn off just this option, in the constructor.)

div(*a*, *q*, *o*)

Divides qubit by integer

'Divides' the given qubits by the integer. (This is rather the adjoint of mul().) Carry register is required for maintaining the unitary nature of operation.

Parameters

- **a** – integer to divide by
- **q** – qubits to divide
- **o** – carry register

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot div()! (Turn off just this option, in the constructor.)

`muln`(*a*, *m*, *q*, *o*)

Modulo Multiplication

Modulo Multiplication of the given integer to the given set of qubits Out-of-place register is required to store the resultant.

Parameters

- **a** – number to multiply
- **m** – modulo number
- **q** – list of qubits to multiply the number
- **o** – carry register

Raises

RuntimeError – QrackSimulator raised an exception.

`divn`(*a*, *m*, *q*, *o*)

Modulo Division

'Modulo Division' of the given set of qubits by the given integer (This is rather the adjoint of muln().) Out-of-place register is required to retrieve the resultant.

Parameters

- **a** – integer by which qubit will be divided
- **m** – modulo integer
- **q** – qubits to divide
- **o** – carry register

Raises

RuntimeError – QrackSimulator raised an exception.

`pown`(*a*, *m*, *q*, *o*)

Modulo Power

Raises the qubit to the power *a* to which *mod m* is applied to. Out-of-place register is required to store the resultant.

Parameters

- **a** – number in power
- **m** – modulo number
- **q** – list of qubits to exponentiate
- **o** – out-of-place register

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot pown()! (Turn off just this option, in the constructor.)

`mcadd`(*a*, *c*, *q*)

Controlled-add

Adds the given integer to the given set of qubits if all controlled qubits are |*1*>.

Parameters

- **a** – number to add.
- **c** – list of controlled qubits.
- **q** – list of qubits to add the number

Raises
RuntimeError – QrackSimulator raised an exception.

mcsub(*a*, *c*, *q*)

Controlled-subtract

Subtracts the given integer to the given set of qubits if all controlled qubits are |*1*>.

Parameters

- **a** – number to subtract.
- **c** – list of controlled qubits.
- **q** – list of qubits to add the number

Raises
RuntimeError – QrackSimulator raised an exception.

mcmul(*a*, *c*, *q*, *o*)

Controlled-multiply

Multiplies the given integer to the given set of qubits if all controlled qubits are |*1*>. Carry register is required for maintaining the unitary nature of operation.

Parameters

- **a** – number to multiply
- **c** – list of controlled qubits.
- **q** – list of qubits to add the number
- **o** – carry register

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot mcmul()! (Turn off just this option, in the constructor.)

mcdiv(*a*, *c*, *q*, *o*)

Controlled-divide.

'Divides' the given qubits by the integer if all controlled qubits are |*1*>. (This is rather the adjoint of mcmul().) Carry register is required for maintaining the unitary nature of operation.

Parameters

- **a** – number to divide by
- **c** – list of controlled qubits.
- **q** – qubits to divide
- **o** – carry register

Raises

- **RuntimeError** – QrackSimulator raised an exception.

- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot mcdiv()! (Turn off just this option, in the constructor.)

mcmuln(*a*, *c*, *m*, *q*, *o*)

Controlled-modulo multiplication

Modulo multiplication of the given integer to the given set of qubits if all controlled qubits are |*1*>. Out-of-place register is required to store the resultant.

Parameters

- **a** – number to multiply
- **c** – list of controlled qubits.
- **m** – modulo number
- **q** – list of qubits to add the number
- **o** – out-of-place output register

Raises

RuntimeError – QrackSimulator raised an exception.

mcdivn(*a*, *c*, *m*, *q*, *o*)

Controlled-divide.

Modulo division of the given qubits by the given number if all controlled qubits are |*1*>. (This is rather the adjoint of mcmuln().) Out-of-place register is required to retrieve the resultant.

Parameters

- **a** – number to divide by
- **c** – list of controlled qubits.
- **m** – modulo number
- **q** – qubits to divide
- **o** – carry register

Raises

RuntimeError – QrackSimulator raised an exception.

mcpown(*a*, *c*, *m*, *q*, *o*)

Controlled-modulo Power

Raises the qubit to the power *a* to which *mod m* is applied to if all the controlled qubits are set to |*1*>. Out-of-place register is required to store the resultant.

Parameters

- **a** – number in power
- **c** – control qubits
- **m** – modulo number
- **q** – list of qubits to exponentiate
- **o** – out-of-place register

Raises

- **RuntimeError** – QrackSimulator raised an exception.

- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot mcpown()! (Turn off just this option, in the constructor.)

lda(*qi*, *qv*, *t*)

Load Accumalator

Quantum counterpart for LDA from MOS-6502 assembly. *t* must be of the length *2 ** len(qi)*. It loads each list entry index of t into the qi register and each list entry value into the qv register.

Parameters

- **qi** – qubit register for index
- **qv** – qubit register for value
- **t** – list of values

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot lda()! (Turn off just this option, in the constructor.)

adc(*s*, *qi*, *qv*, *t*)

Add with Carry

Quantum counterpart for ADC from MOS-6502 assembly. *t* must be of the length *2 ** len(qi)*.

Parameters

- **qi** – qubit register for index
- **qv** – qubit register for value
- **t** – list of values

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot adc()! (Turn off just this option, in the constructor.)

sbc(*s*, *qi*, *qv*, *t*)

Subtract with Carry

Quantum counterpart for SBC from MOS-6502 assembly. *t* must be of the length *2 ** len(qi)*

Parameters

- **qi** – qubit register for index
- **qv** – qubit register for value
- **t** – list of values

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot sbc()! (Turn off just this option, in the constructor.)

hash(*q*, *t*)

Hash function

Replicates the behaviour of LDA without the index register. For the operation to be unitary, the entries present in *t* must be unique, and the length of *t* must be *2 ** len(qi)*.

Parameters

- **q** – qubit register for value
- **t** – list of values

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot hash()! (Turn off just this option, in the constructor.)

qand(*qi1*, *qi2*, *qo*)

Logical AND

Logical AND of 2 qubits whose result is stored in the target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

qor(*qi1*, *qi2*, *qo*)

Logical OR

Logical OR of 2 qubits whose result is stored in the target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

qxor(*qi1*, *qi2*, *qo*)

Logical XOR

Logical exlusive-OR of 2 qubits whose result is stored in the target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

qnand(*qi1*, *qi2*, *qo*)

Logical NAND

Logical NAND of 2 qubits whose result is stored in the target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

qnor(*qi1*, *qi2*, *qo*)

Logical NOR

Logical NOR of 2 qubits whose result is stored in the target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

qxnor(*qi1*, *qi2*, *qo*)

Logical XOR

Logical exlusive-NOR of 2 qubits whose result is stored in the target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

cland(*ci*, *qi*, *qo*)

Classical AND

Logical AND with one qubit and one classical bit whose result is stored in target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

clor(*ci*, *qi*, *qo*)

Classical OR

Logical OR with one qubit and one classical bit whose result is stored in target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

clxor(*ci*, *qi*, *qo*)

Classical XOR

Logical exlusive-OR with one qubit and one classical bit whose result is stored in target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

clnand(*ci*, *qi*, *qo*)

Classical NAND

Logical NAND with one qubit and one classical bit whose result is stored in target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

clnor(*ci*, *qi*, *qo*)

Classical NOR

Logical NOR with one qubit and one classical bit whose result is stored in target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

`clxnor`(*ci*, *qi*, *qo*)

Classical XNOR

Logical exlusive-NOR with one qubit and one classical bit whose result is stored in target qubit.

Parameters

- **qi1** – qubit 1
- **qi2** – qubit 2
- **qo** – target qubit

Raises

RuntimeError – QrackSimulator raised an exception.

`qft`(*qs*)

Quantum Fourier Transform

Applies Quantum Fourier Transform on the list of qubits provided.

Parameters

qs – list of qubits

Raises

RuntimeError – QrackSimulator raised an exception.

`iqft`(*qs*)

Inverse-quantum Fourier Transform

Applies Inverse-quantum Fourier Transform on the list of qubits provided.

Parameters

qs – list of qubits

Raises

RuntimeError – QrackSimulator raised an exception.

`allocate_qubit`(*qid*)

Allocate Qubit

Allocate 1 new qubit with the given qubit ID.

Parameters

qid – qubit id

Raises

RuntimeError – QrackSimulator raised an exception.

`release`(*q*)

Release Qubit

Release qubit given by the given qubit ID.

Parameters

q – qubit id

Raises

RuntimeError – QrackSimulator raised an exception.

Returns

If the qubit was in |*0*> state with small tolerance.

`num_qubits()`

Get Qubit count

Returns the qubit count of the simulator.

Parameters
`q` – qubit id

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
Qubit count of the simulator

`compose(other, q)`

Compose qubits

Compose quantum description of given qubit with the current system.

Parameters
`q` – qubit id

Raises

- `RuntimeError` – QrackSimulator raised an exception.
- `RuntimeError` – QrackSimulator with isTensorNetwork=True option cannot compose()! (Turn off just this option, in the constructor.)

`decompose(q)`

Decompose system

Decompose the given qubit out of the system. Warning: The qubit subsystem state must be separable, or the behavior of this method is undefined.

Parameters
`q` – qubit id

Raises

- `RuntimeError` – QrackSimulator raised an exception.
- `RuntimeError` – QrackSimulator with isTensorNetwork=True option cannot decompose()! (Turn off just this option, in the constructor.)

Returns
State of the systems.

`dispose(q)`

Dispose qubits

Minimally decompose a set of contiguous bits from the separably composed unit, and discard the separable bits. Warning: The qubit subsystem state must be separable, or the behavior of this method is undefined.

Parameters
`q` – qubit

Raises

- `RuntimeError` – QrackSimulator raised an exception.
- `RuntimeError` – QrackSimulator with isTensorNetwork=True option cannot dispose()! (Turn off just this option, in the constructor.)

Returns
State of the systems.

`dump_ids()`

Dump all IDs

Dump all IDs from the selected simulator ID into the callback.

Returns
List of ids

`dump_ids_callback()`

C callback function

`dump()`

Dump state vector

Dump state vector from the selected simulator ID into the callback.

Returns
State vector list

`dump_callback(i)`

C callback function

`in_ket(ket)`

Set state vector

Set state vector for the selected simulator ID. Warning: State vector is not always the internal representation leading to sub-optimal performance of the method.

Parameters
`ket` – the state vector to which simulator will be set

Raises
`RuntimeError` – QrackSimulator raised an exception.

`out_ket()`

Set state vector

Returns the raw state vector of the simulator. Warning: State vector is not always the internal representation leading to sub-optimal performance of the method.

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
list representing the state vector.

`prob(q)`

Probability of |*1*>

Get the probability that a qubit is in the |*1*> state.

Parameters
`q` – qubit id

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
probability of qubit being in |*1*>

`prob_rdm`(*q*)

Probability of |*1*>, (tracing out the reduced density matrix without stabilizer ancillary qubits)

Get the probability that a qubit is in the |*1*> state.

Parameters
`q` – qubit id

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
probability of qubit being in |*1*>

`prob_perm`(*q*, *c*)

Probability of permutation

Get the probability that the qubit IDs in "q" have the truth values in "c", directly corresponding by list index.

Parameters

- `q` – list of qubit ids
- `c` – list of qubit truth values bools

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
probability that each qubit in "q[i]" has corresponding truth value in "c[i]", at once

`prob_perm_rdm`(*q*, *c*, *r=True*)

Probability of permutation, (tracing out the reduced density matrix without stabilizer ancillary qubits)

Get the probability that the qubit IDs in "q" have the truth values in "c", directly corresponding by list index.

Parameters

- `q` – list of qubit ids
- `c` – list of qubit truth values bools
- `r` – round Rz gates down from T^(1/2)

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
probability that each qubit in "q[i]" has corresponding truth value in "c[i]", at once

`permutation_expectation`(*q*)

Permutation expectation value

Get the permutation expectation value, based upon the order of input qubits.

Parameters
`q` – qubits, from low to high

Raises
`RuntimeError` – QrackSimulator raised an exception.

Returns
Expectation value

permutation_expectation_rdm(*q*, *r=True*)

Permutation expectation value, (tracing out the reduced density matrix without stabilizer ancillary qubits)

Get the permutation expectation value, based upon the order of input qubits.

Parameters

- **q** – qubits, from low to high
- **r** – round Rz gates down from T^(1/2)

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Expectation value

factorized_expectation(*q*, *c*)

Factorized expectation value

Get the factorized expectation value, where each entry in "c" is an expectation value for corresponding "q" being false, then true, repeated for each in "q".

Parameters

- **q** – qubits, from low to high
- **c** – qubit falsey/truthy values, from low to high

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Expectation value

factorized_expectation_rdm(*q*, *c*, *r=True*)

Factorized expectation value, (tracing out the reduced density matrix without stabilizer ancillary qubits)

Get the factorized expectation value, where each entry in "c" is an expectation value for corresponding "q" being false, then true, repeated for each in "q".

Parameters

- **q** – qubits, from low to high
- **c** – qubit falsey/truthy values, from low to high
- **r** – round Rz gates down from T^(1/2)

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Expectation value

factorized_expectation_fp(*q*, *c*)

Factorized expectation value (floating-point)

Get the factorized expectation value, where each entry in "c" is an expectation value for corresponding "q" being false, then true, repeated for each in "q".

Parameters

- **q** – qubits, from low to high
- **c** – qubit falsey/truthy values, from low to high

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Expectation value

factorized_expectation_fp_rdm(*q*, *c*, *r=True*)

Factorized expectation value, (tracing out the reduced density matrix without stabilizer ancillary qubits)

Get the factorized expectation value, where each entry in "c" is an expectation value for corresponding "q" being false, then true, repeated for each in "q".

Parameters

- **q** – qubits, from low to high
- **c** – qubit falsey/truthy values, from low to high
- **r** – round Rz gates down from T^(1/2)

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Expectation value

joint_ensemble_probability(*b*, *q*)

Ensemble probability

Find the joint probability for all specified qubits under the respective Pauli basis transformations.

Parameters

- **b** – pauli basis
- **q** – specified qubits

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Expectation value

phase_parity(*la*, *q*)

Phase to odd parity

Applies *e^(i*la)* phase factor to all combinations of bits with odd parity, based upon permutations of qubits.

Parameters

- **la** – phase
- **q** – specified qubits

Raises

- **RuntimeError** – QrackSimulator raised an exception.
- **RuntimeError** – QrackSimulator with isTensorNetwork=True option cannot phase_parity()! (Turn off just this option, in the constructor.)

try_separate_1qb(*qi1*)

Manual seperation

Exposes manual control for schmidt decomposition which attempts to decompose the qubit with possible performance improvement

Parameters
qi1 – qubit to be decomposed

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
State of the qubit.

try_separate_2qb(*qi1*, *qi2*)

Manual two-qubits seperation

two-qubits counterpart of *try_separate_1qb*.

Parameters

- **qi1** – first qubit to be decomposed
- **qi2** – second qubit to be decomposed

Raises
Runtimeerror – QrackSimulator raised an exception.

Returns
State of both the qubits.

try_separate_tolerance(*qs*, *t*)

Manual multi-qubits seperation

Multi-qubits counterpart of *try_separate_1qb*.

Parameters

- **qs** – list of qubits to be decomposed
- **t** – allowed tolerance

Raises
Runtimeerror – QrackSimulator raised an exception.

Returns
State of all the qubits.

get_unitary_fidelity()

Get fidelity estimate

When using "Schmidt decomposition rounding parameter" ("SDRP") approximate simulation, QrackSimulator() can make an excellent estimate of its overall fidelity at any time, tested against a nearest-neighbor variant of quantum volume circuits.

Resetting the fidelity calculation to 1.0 happens automatically when calling *mall* are can be done manually with *reset_unitary_fidelity()*.

Raises
RuntimeError – QrackSimulator raised an exception.

Returns
Fidelity estimate

reset_unitary_fidelity()

Reset fidelity estimate

When using "Schmidt decomposition rounding parameter" ("SDRP") approximate simulation, QrackSimulator() can make an excellent estimate of its overall fidelity at any time, tested against a nearest-neighbor variant of quantum volume circuits.

Resetting the fidelity calculation to 1.0 happens automatically when calling *m_all* or can be done manually with *reset_unitary_fidelity()*.

Raises
RuntimeError – QrackSimulator raised an exception.

set_sdrp(*sdrp*)

Set "Schmidt decomposition rounding parameter"

When using "Schmidt decomposition rounding parameter" ("SDRP") approximate simulation, QrackSimulator() can make an excellent estimate of its overall fidelity at any time, tested against a nearest-neighbor variant of quantum volume circuits.

Resetting the fidelity calculation to 1.0 happens automatically when calling *m_all* or can be done manually with *reset_unitary_fidelity()*.

Raises
RuntimeError – QrackSimulator raised an exception.

set_reactive_separate(*irs*)

Set reactive separation option

If reactive separation is available, then this method turns it off/on. Note that reactive separation is on by default.

Parameters
irs – is aggresively separable

Raises
RuntimeError – QrackSimulator raised an exception.

set_t_injection(*iti*)

Set t-injection option

If t-injection is available, then this method turns it off/on. Note that t-injection is on by default.

Parameters
iti – use "reverse t-injection gadget"

Raises
RuntimeError – QrackSimulator raised an exception.

out_to_file(*filename*)

Output state to file (stabilizer only!)

Outputs the hybrid stabilizer state to file.

Parameters
filename – Name of file

in_from_file(*is_binary_decision_tree=False*, *is_paged=True*, *is_cpu_gpu_hybrid=True*, *is_opencl=True*, *is_host_pointer=False*)

Input state from file (stabilizer only!)

Reads in a hybrid stabilizer state from file.

Parameters
filename – Name of file

file_to_qiskit_circuit(*is_hardware_encoded=False*)

Convert an output state file to a Qiskit circuit

Reads in an (optimized) circuit from a file named according to the "filename" parameter and outputs a Qiskit circuit.

Parameters
filename – Name of file

Raises
RuntimeErorr – Before trying to file_to_qiskit_circuit() with QrackCircuit, you must install Qiskit, numpy, and math!

file_to_optimized_qiskit_circuit()

Convert an output state file to a Qiskit circuit

Reads in a circuit from a file named according to the "filename" parameter and outputs a 'hyper-optimized' Qiskit circuit that favors maximum reduction in gate count and depth at the potential expense of additional non-Clifford gates. (Ancilla qubits are left included in the output, though they probably have no gates.)

Parameters
filename – Name of file

Raises
RuntimeErorr – Before trying to file_to_qiskit_circuit() with QrackCircuit, you must install Qiskit, numpy, and math!

_apply_pyzx_op(*gate*)

run_pyzx_gates(*gates*)

PYZX Gates

Converts PYZX gates to *QRackSimulator* and immediately executes them.

Parameters
gates – list of PYZX gates

Raises
RuntimeError – QrackSimulator raised an exception.

_apply_op(*operation*)

_add_sample_measure(*sample_qubits*, *sample_clbits*, *num_samples*)

Generate data samples from current statevector.

Taken almost straight from the terra source code.

Parameters

- **measure_params** (*list*) – List of (qubit, clbit) values for measure instructions to sample.
- **num_samples** (*int*) – The number of data samples to generate.

Returns
A list of data values in hex format.

Return type
list

run_qiskit_circuit(*experiment*, *shots=1*)

class pyqrack.**QrackNeuron**(*simulator*, *controls*, *target*, *activation_fn=NeuronActivationFn.Sigmoid*, *alpha=1.0*, *tolerance=sys.float_info.epsilon*, *_init=True*)

Class that exposes the QNeuron class of Qrack

This model of a "quantum neuron" is based on the concept of a "uniformly controlled" rotation of a single output qubit around the Pauli Y axis, and has been developed by others. In our case, the primary relevant gate could also be called a single-qubit-target multiplexer.

(See https://arxiv.org/abs/quant-ph/0407010 for an introduction to "uniformly controlled gates.)

QrackNeuron is meant to be interchangeable with a single classical neuron, as in conventional neural net software. It differs from classical neurons in conventional neural nets, in that the "synaptic cleft" is modelled as a single qubit. Hence, this neuron can train and predict in superposition.

nid

Qrack ID of this neuron

Type
int

simulator

Simulator instance for all synaptic clefts of the neuron

Type
QrackSimulator

controls

Indices of all "control" qubits, for neuron input

Type
list(int)

target

Index of "target" qubit, for neuron output

Type
int

tolerance

Rounding tolerance

Type
double

_get_error()

_throw_if_error()

__del__()

clone()

Clones this neuron.

Create a new, independent neuron instance with identical angles, inputs, output, and tolerance, for the same QrackSimulator.

Raises
RuntimeError – QrackNeuron C++ library raised an exception.

_ulonglong_byref(*a*)

_real1_byref(*a*)

set_angles(*a*)

Directly sets the neuron parameters.

Set all synaptic parameters of the neuron directly, by a list enumerated over the integer permutations of input qubits.

Parameters
a (*list(double)*) – List of input permutation angles

Raises

- **ValueError** – Angles 'a' in QrackNeuron.set_angles() must contain at least (2 ** len(self.controls)) elements.
- **RuntimeError** – QrackSimulator raised an exception.

get_angles()

Directly gets the neuron parameters.

Get all synaptic parameters of the neuron directly, as a list enumerated over the integer permutations of input qubits.

Raises
RuntimeError – QrackNeuron C++ library raised an exception.

set_alpha(*a*)

Set the neuron 'alpha' parameter.

To enable nonlinear activation, *QrackNeuron* has an 'alpha' parameter that is applied as a power to its angles, before learning and prediction. This makes the activation function sharper (or less sharp).

Raises
RuntimeError – QrackNeuron C++ library raised an exception.

set_activation_fn(*f*)

Sets the activation function of this QrackNeuron

Nonlinear activation functions can be important to neural net applications, like DNN. The available activation functions are enumerated in *NeuronActivationFn*.

Raises
RuntimeError – QrackNeuron C++ library raised an exception.

predict(*e=True*, *r=True*)

Predict based on training

"Predict" the anticipated output, based on input and training. By default, "predict()" will initialize the output qubit as by resetting to |0> and then acting a Hadamard gate. From that state, the method amends the output qubit upon the basis of the state of its input qubits, applying a rotation around Pauli Y axis according to the angle learned for the input.

Parameters

- **e** (*bool*) – If False, predict the opposite
- **r** (*bool*) – If True, start by resetting the output to 50/50

Raises
RuntimeError – QrackNeuron C++ library raised an exception.

unpredict(*e=True*)

Uncompute a prediction

Uncompute a 'prediction' of the anticipated output, based on input and training.

Parameters
e (*bool*) – If False, unpredict the opposite

Raises
RuntimeError – QrackNeuron C++ library raised an exception.

learn_cycle(*e=True*)

Run a learning cycle

A learning cycle consists of predicting a result, saving the classical outcome, and uncomputing the prediction.

Parameters
e (*bool*) – If False, predict the opposite

Raises
RuntimeError – QrackNeuron C++ library raised an exception.

learn(*eta*, *e=True*, *r=True*)

Learn from current qubit state

"Learn" to associate current inputs with output. Based on input qubit states and volatility 'eta,' the input state synaptic parameter is updated to prefer the "e" ("expected") output.

Parameters

- **eta** (*double*) – Training volatility, 0 to 1
- **e** (*bool*) – If False, predict the opposite
- **r** (*bool*) – If True, start by resetting the output to 50/50

Raises
RuntimeError – QrackNeuron C++ library raised an exception.

learn_permutation(*eta*, *e=True*, *r=True*)

Learn from current classical state

Learn to associate current inputs with output, under the assumption that the inputs and outputs are "classical." Based on input qubit states and volatility 'eta,' the input state angle is updated to prefer the "e" ("expected") output.

Parameters

- **eta** (*double*) – Training volatility, 0 to 1
- **e** (*bool*) – If False, predict the opposite
- **r** (*bool*) – If True, start by resetting the output to 50/50

Raises
RuntimeError – QrackNeuron C++ library raised an exception.

class pyqrack.**QrackCircuit**(*is_collapse=True*, *clone_cid=-1*, *is_inverse=False*, *past_light_cone=[]*)

Class that exposes the QCircuit class of Qrack

QrackCircuit allows the user to specify a unitary circuit, before running it. Upon running the state, the result is a QrackSimulator state. Currently, measurement is not supported, but measurement can be run on the resultant QrackSimulator.

cid

Qrack ID of this circuit

Type
int

`__del__`()

`_ulonglong_byref`(*a*)

`_double_byref`(*a*)

`_complex_byref`(*a*)

`clone`()

Make a new circuit that is an exact clone of this circuit

Raises
`RuntimeError` – QrackCircuit C++ library raised an exception.

`inverse`()

Make a new circuit that is the exact inverse of this circuit

Raises
`RuntimeError` – QrackCircuit C++ library raised an exception.

`past_light_cone`(*q*)

Make a new circuit with just this circuits' past light cone for certain qubits.

Parameters
`q` – list of qubit indices to include at beginning of past light cone

Raises
`RuntimeError` – QrackCircuit C++ library raised an exception.

`get_qubit_count`()

Get count of qubits in circuit

Raises
`RuntimeError` – QrackCircuit C++ library raised an exception.

`swap`(*q1*, *q2*)

Add a 'Swap' gate to the circuit

Parameters

- `q1` – qubit index #1
- `q2` – qubit index #2

Raises
`RuntimeError` – QrackCircuit C++ library raised an exception.

`mtrx`(*m*, *q*)

Operation from matrix.

Applies arbitrary operation defined by the given matrix.

Parameters

- `m` – row-major complex list representing the operator.
- `q` – the qubit number on which the gate is applied to.

Raises

- `ValueError` – 2x2 matrix 'm' in QrackCircuit.mtrx() must contain at least 4 elements.

- **RuntimeError** – QrackSimulator raised an exception.

ucmtrx(*c*, *m*, *q*, *p*)

Multi-controlled single-target-qubit gate

Specify a controlled gate by its control qubits, its single-qubit matrix "payload," the target qubit, and the permutation of qubits that activates the gate.

Parameters

- **c** – list of controlled qubits
- **m** – row-major complex list representing the operator.
- **q** – target qubit
- **p** – permutation of target qubits

Raises

- **ValueError** – 2x2 matrix 'm' in QrackCircuit.ucmtrx() must contain at least 4 elements.
- **RuntimeError** – QrackSimulator raised an exception.

run(*qsim*)

Run circuit on simulator

Run the encoded circuit on a specific simulator. The result will remain in this simulator.

Parameters
qsim – QrackSimulator on which to run circuit

Raises
RuntimeError – QrackCircuit raised an exception.

out_to_file(*filename*)

Output optimized circuit to file

Outputs the (optimized) circuit to a file named according to the "filename" parameter.

Parameters
filename – Name of file

in_from_file()

Read in optimized circuit from file

Reads in an (optimized) circuit from a file named according to the "filename" parameter.

Parameters
filename – Name of file

file_to_qiskit_circuit()

Convert an output file to a Qiskit circuit

Reads in an (optimized) circuit from a file named according to the "filename" parameter and outputs a Qiskit circuit.

Parameters
filename – Name of file

Raises
RuntimeErorr – Before trying to file_to_qiskit_circuit() with QrackCircuit, you must install Qiskit, numpy, and math!

in_from_qiskit_circuit()

Read a Qiskit circuit into a QrackCircuit

Reads in a circuit from a Qiskit *QuantumCircuit*

Parameters
circ – Qiskit circuit

Raises
RuntimeErorr – Before trying to file_to_qiskit_circuit() with QrackCircuit, you must install Qiskit, numpy, and math!

file_to_quimb_circuit(*circuit_type=QuimbCircuitType.Circuit*, *psi0=None*, *gate_opts=None*, *tags=None*, *psi0_dtype='complex128'*, *psi0_tag='PSI0'*, *bra_site_ind_id='b{}'*)

Convert an output file to a Quimb circuit

Reads in an (optimized) circuit from a file named according to the "filename" parameter and outputs a Quimb circuit.

Parameters

- **filename** – Name of file
- **circuit_type** – "QuimbCircuitType" enum value specifying type of Quimb circuit
- **psi0** – The initial state, assumed to be |00000....0> if not given. The state is always copied and the tag PSI0 added
- **gate_opts** – Default keyword arguments to supply to each gate_TN_1D() call during the circuit
- **tags** – Tag(s) to add to the initial wavefunction tensors (whether these are propagated to the rest of the circuit's tensors
- **psi0_dtype** – Ensure the initial state has this dtype.
- **psi0_tag** – Ensure the initial state has this tag.
- **bra_site_ind_id** – Use this to label 'bra' site indices when creating certain (mostly internal) intermediate tensor networks.

Raises
RuntimeErorr – Before trying to file_to_quimb_circuit() with QrackCircuit, you must install quimb, Qiskit, numpy, and math!

file_to_tensorcircuit(*inputs=None*, *circuit_params=None*, *binding_params=None*)

Convert an output file to a TensorCircuit circuit

Reads in an (optimized) circuit from a file named according to the "filename" parameter and outputs a TensorCircuit circuit.

Parameters

- **filename** – Name of file
- **inputs** – pass-through to tensorcircuit.Circuit.from_qiskit
- **circuit_params** – pass-through to tensorcircuit.Circuit.from_qiskit
- **binding_params** – pass-through to tensorcircuit.Circuit.from_qiskit

Raises
RuntimeErorr – Before trying to file_to_quimb_circuit() with QrackCircuit, you must install TensorCircuit, Qiskit, numpy, and math!

in_from_tensorcircuit(*enable_instruction=False*, *enable_inputs=False*)

Convert a TensorCircuit circuit to a QrackCircuit

Accepts a TensorCircuit circuit and outputs an equivalent QrackCircuit

Parameters

- **tcirc** – TensorCircuit circuit
- **enable_instruction** – whether to also export measurement and reset instructions
- **enable_inputs** – whether to also export the inputs

Raises

RuntimeErorr – Before trying to in_from_tensorcircuit() with QrackCircuit, you must install TensorCircuit, Qiskit, numpy, and math!

class pyqrack.**Pauli**

Bases: enum.IntEnum

Enum where members are also (and must be) ints

PauliI = 0

PauliX = 1

PauliY = 3

PauliZ = 2

class pyqrack.**NeuronActivationFn**

Bases: enum.IntEnum

Enum where members are also (and must be) ints

Sigmoid = 0

ReLU = 1

GeLU = 2

Generalized_Logistic = 3

LeakyReLU = 4

class pyqrack.**QuimbCircuitType**

Bases: enum.IntEnum

Enum where members are also (and must be) ints

Circuit = 0

CircuitDense = 1

CircuitMPS = 3

PYTHON MODULE INDEX

p

INDEX

Symbols

A

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

X

Y

Z