MathWorks®

Getting Started Guide for Quantum Computing with MATLAB

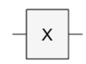
MATLAB Support Package for Quantum Computing lets you build, simulate, and run quantum algorithms.

Overview and download >>

Build >>

Single Qubit Gate

Apply a single qubit gate to a quantum circuit. The quantity in parentheses () is the qubit index on which the gate is applied.

	Quantum Circuit Object	Plot	Matrix
Syntax	`>> qc = quantumCircuit (xGate(1))`	`>> plot (qc)`	`>> getMatrix(qc)`
Output	`NumQubits:1` `Gates:[1×1 quantum.gate.SimpleGate]`		$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$

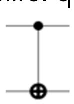
Single Qubit Gate	Description
`xGate(1)`	xGate = π rotation around X-axis to qubit 1
`yGate(1)`	yGate = π rotation around Y-axis to qubit 1
`zGate(1)`	zGate = π rotation around Z-axis to qubit 1
`sGate(1)`	sGate = π/2 positive rotation around Z-axis
`tGate(1)`	tGate = π/2 positive rotation around Z-axis
`tiGate(1)`	tiGate = π/2 negative rotation around Z-axis
`hGate(1)`	hGate = Hadamard gate
`idGate(1)`	idGate = Identity gate (does nothing)

The rotation gates are single qubit gates that take two arguments. The first is the qubit index on which the gate is applied, and second is the rotation angle or phase (θ) in radian.

Parameterized Single Qubit Gate	Description
`rxGate(1,pi/2)`	rxGate = X-axis rotation gate
`ryGate(1,pi/2)`	ryGate = Y-axis rotation gate
`rzGate(1,pi/2)`	rzGate = Z-axis rotation gate
`r1Gate(1,pi/2)`	r1Gate = Z-axis rotation gate with global phase

Gates with One Control Qubit

Apply a gate with one control qubit to a quantum circuit. These gates have two arguments in parentheses (); the first is the control qubit and the second is the target qubit. If the control qubit is in the |0> state, then the gate does nothing. If the control qubit is in the |1> state, then the specified gate acts on the target qubit. >> `cxGate(control,target)`



Qubit index 1: control qubit
Qubit index 2: target qubit

The controlled rotation gates are two qubit gates that take three arguments. The first is the control qubit, the second is the target qubit, and third is the rotation angle or phase (θ) in radian.

Two Qubit Gate	Description
`cxGate(1,2)`	cxGate or cnotGate = π rotation around X-axis to qubit 2 if qubit 1 is in |1> state
`cyGate(1,2)`	cyGate = π rotation around Y-axis to qubit 2 if qubit 1 is in |1> state
`czGate(1,2)`	czGate = π rotation around Z-axis to qubit 2 if qubit 1 is in |1> state
`chGate(1,2)`	czGate = Controlled Hadamard gate

Parameterized Two Qubit Gate	Description
`crxGate(1,2,pi/2)`	crxGate = Controlled X-axis rotation gate
`cryGate(1,2,pi/2)`	cryGate = Controlled y-axis rotation gate
`crzGate(1,2,pi/2)`	crzGate = Controlled z-axis rotation gate
`cr1Gate(1,2,pi/2)`	cr1Gate = Controlled z-axis rotation gate with global phase

MathWorks®
Accelerating the pace of engineering and science

Special Gates

Gate	Description
`compositeGate (qc, [1,2])`	Constructs a composite gate from an inner quantum circuit and returns a CompositeGate object. The qc is the inner quantum circuit in the example.
`ccxGate(1,2,3)`	Controlled controlled X gate (CCNOT or Toffoli gate)
`mcxGate(1:3,4,5)`	Multi-controlled X gate The first argument is three control qubits (1,2,3), the second argument is one target qubit (4), and the third argument is one ancilla qubits (5). This gate operates on a single target qubit based on the states of the control qubits, with a number of ancilla qubits that determines the number of simple gates
`qftGate(1:3)`	Quantum Fourier transform (QFT) gate

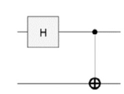
In the gates below, the first and second arguments are the qubit indices, and the third is phase (θ), except for the **`swapGate`** which has two arguments. Swapping the two target qubits for all the gates below does not change the gate operation.

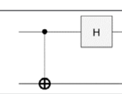
Gate	Description
`rxxGate(1,2,pi/2)`	Ising XX coupling gate
`ryyGate(1,2,pi/2)`	Ising YY coupling gate
`rzzGate(1,2,pi/2)`	Ising ZZ coupling gate
`swapGate(1,2)`	Swaps the values of the qubits.

Simulate >>

Quantum Circuit Operations

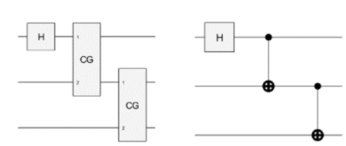
Operation	Example	Output

`plot` = Draw a quantum circuit	`>> gates = [hGate(1); cxGate(1,2)];`
	`>> bell = quantumCircuit(gates, name="bell"); % circuit`
	`>> plot(bell) %plot`
		

`inv` = Inverse of a quantum circuit or gate	`>> bell_inverted = inv(bell);` `>> plot(bell_inverted)`	
`getMatrix` = Matrix representation of a quantum circuit or gate	`>> getMatrix(bell)`	$\begin{bmatrix} 0.707 & 0 & 0.707 & 0 \\ 0 & 0.707 & 0 & 0.707 \\ 0 & 0.707 & 0 & -0.707 \\ 0.707 & 0 & -0.707 & 0 \end{bmatrix}$

`generateQASM` = Generate QASM code	`>> generateQASM(bell)`	`"OPENQASM 3.0;`
		`include "stdgates.inc";`
		`qubit[2] q; bit[2] c; h`
		`q[0]; cx q[0],q[1];`
		`c = measure q;"`

`simulate(circuit, inputState),` Simulate circuit and specify the initial quantum state of the circuit and return a QuantumState object	`>> simulate(bell)`	`BasisStates: [4×1 string]`
		`Amplitudes: [4×1 double]`
		`NumQubits: 2`

`unpack` = Unpack composite gates inside a quantum circuit	`>> cnot_custom = [cxGate(1,2)];`
	`>> qc_inner =`
	`quantumCircuit(cnot_custom);`
	`>> gates_appended = [hGate(1)`
	`compositeGate(qc_inner,[1 2])`
	`compositeGate(qc_inner,[2 3])];`
	`>> circ=`
	`quantumCircuit(gates_appended);`
	`>> plot(circ)%packed composite gate`
	`>> plot(unpack(circ, "recursive"))`
		

MathWorks®
Accelerating the pace of engineering and science

Simulated Quantum Circuit Operations		
Operation	**Example**	**Output**

Initialize the quantum state of the circuit where the qubit 1 (top qubit in circuit plot) is set to |1> and second is to |0>	`S = simulate(bell,"10")`	`QuantumState with properties:`
		`BasisStates: [4×1 string]`
		`Amplitudes: [4×1 double]`
		`NumQubits: 2`

Show the output basis states and their amplitudes	`S.BasisStates` `S.Amplitudes`	`BasisStates: "00" "01" "10" "11"` `Amplitudes: 0.7071 0 0 -0.7071`
Show the final state of the circuit such as the basis in which to represent each qubit	`f = formula(S)` `f2 = formula(S,Basis="X")`	`ans = "0.70711 * |00> + -0.70711 * |11>"` `ans = "0.70711 * |+-> + 0.70711 * |-+>"`
Show the possible states and the probability of measuring each state	`[states,P] = querystates(S)`	`states = "00"   P = 0.5` `"11"   0.5`

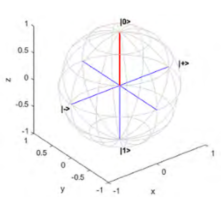
Randomly sample the quantum state of the circuit with any number of shots	`M = randsample(S,50);`
	`T = table(m.Counts,m.Probabilities,m.MeasuredStates, VariableNames= ["Counts", "Probabilites", "States"])`
		`ans =`
		`2×3 table`
		`Counts   Probabilities   States`
		`28   0.56   "00"`
		`22   0.44   "11"`

Run on AWS >> and IBM Quantum >>

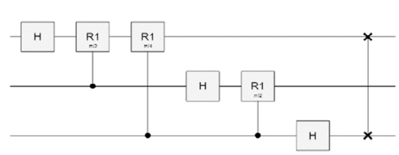
Function	Description
`dev = QuantumDeviceAWS ("SV1",Region=reg,S3Path=bucketPath)`	Connect to an Amazon Braket device, specifying the name of the device as well as its region and the path of the Amazon S3 bucket to store results
`dev = QuantumDeviceIBM("ibmq_qasm_simulator", ... AccountName=myAccountName,FileName=myFileName);`	Connect to an IBM quantum device, specifying the name of the device and the account name to authenticate using the associated credentials
`fetchDetails(device)`	Get additional information about the device
`task = run(circuit, dev);`	Create a task to run the circuit on the device
`wait(task)`	Check the status of a task
`meas = fetchOutput(task);`	Retrieve the result of the circuit run

Visualizations >>

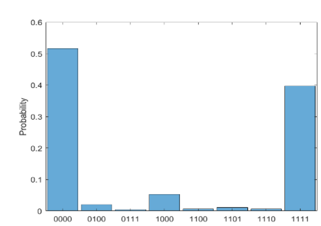
Matrix representation of |0> is [1;0] >> plotBlochSphere([1;0])



\>> plot(qftGate(1:3))



\>> histogram(states)



GitHub with Examples >>

Questions? quantum-computing-community-profile@groups.mathworks.com

mathworks.com