Cheatography

Qiskit Programming - QML Cheat Sheet

by Dinesh1102 via cheatography.com/172447/cs/36251/

Quantum Circuits

```
#Define Simulator
S_simu lator = Aer.ba cke nds (na me= 'st ate vec tor _si mul ato r')[0]
M_simu lator = Aer.ba cke nds (na me= 'qa sm_ sim ula tor ')[0]
#Define Register (1 qubit)
qreg_q = Quantu mRe gis ter (1,'q')
creg_c = Classi cal Reg ist er( 1,'c')
#Define quantum circuit
qc = Quantu mCi rcu it( qre g_q ,cr eg_c)
#Add Quantum gates to Circuit
qc.h(q reg _q[0])
# Add Measure
qc.mea sur e(q reg _q, creg_c)
#RUn circuit on Simulator
job = execut e(q c,M _si mul ator)
result = job.re sult()
result.ge t_c oun ts(qc)
```

Quantum Gates

Names	Example	Notes
I, Identity	`qc.id(0) or qc.i(o)`	Applies I gate to qubit 0.
H, Hadamard	`qc.h(0)`	Applies H gate to qubit 0.
X	`qc.x(0)`	Applies X gate to qubit 0.
Y	`qc.y([ 0,1,2])`	Applies Y gates to qubits 0, 1, and 2.
Z	`qc.z(0)`	Applies Z gate to qubit 0. Equivalent to P gate with π phase rotation.
P, Phase	`qc.p(m ath.pi /2,0)`	Applies P gate with π/2 phase rotation to qubit 0.
S	`qc.s(0)`	Applies S gate to qubit 0. Equivalent to P gate with π/2 phase rotation.
S†	`qc.sdg(0)`	Applies S† gate to qubit 0. Equivalent to P gate with 3π/2 phase rotation.
SX	`qc.sx(0)`	Applies SX (square root of X) gate to qubit 0. Equivalent to RX gate with π/2 rotation.
T	`qc.t(0)`	Applies T gate to qubit 0. Equivalent to P gate with π/4 phase rotation.
T†	`qc.tdg(0)`	Applies T† gate to qubit 0. Equivalent to P gate with 7π/4 phase rotation.
RX	`qc.rx( mat h.p i/4,0)`	Applies RX gate with π/4 rotation to qubit 0.
RY	`qc.ry( mat h.p i/8,0)`	Applies RY gate with π/8 rotation to qubit 0.
RZ	`qc.rz( mat h.p i/2,0)`	Applies RZ gate with π/2 rotation to qubit 0.
U	`qc.u(m ath.pi /2, 0,m ath.pi,5)`	Applies rotation with 3 Euler angles to qubit 5.

Quantum Teleportation

```
from qiskit import *
from qiskit.to ols.ju pyter import *
```

Quantum Teleportation (cont)

```
from qiskit.vi sua liz ation import *
import matplo tli b.p yplot as plotter
import numpy as np
from IPytho n.d isplay import display, Math, Latex
%matpl otlib inline
sim = Aer.ge t_b ack end ('a er_ sim ula tor')
qr = Quantu mRe gis ter(3)
crz = Classi cal Reg ist er(1)
crx = Classi cal Reg ist er(2) # we will need seperates registers for using 'c_if' later.
qc = Quantu mCi rcu it( qr, crz ,crx)
qc.x(0)
qc.h(0) # 'psi' can't be unknown to us as we are creating it here. Let us take '-' state as our 'psi'.
This is done by operating X and H gate on the q0 i.e., H.X|0>
                                                                            # We will verify later if the
'-' is been telepo rted.
qc.dra w(' mpl')
qc.h(1)
qc.cx(1,2) # creating a bell state
qc.bar rier() # Use barrier to separate steps, everything till this barrier is just intial isa tion.
qc.dra w(' mpl')
qc.cx(0,1) # '0' and '1' are with Alice and '2' is with Bob.
# psi_1 prepared.
qc.bar rier() # Use barrier to separate steps
qc.dra w(' mpl')
qc.h(0)
# psi_2 prepared.
qc.bar rier()
qc.dra w(' mpl')
qc.mea sur e(0,0)
qc.mea sur e(1,1)
qc.bar rier()
qc.dra w(' mpl')
qc.x(2 ).c _if (crx,1) # 'c_if' compares a classical register with a value (either 0 or 1) and performs
the
qc.z(2 ).c _if (crz,1) # operation if they are equal.
qc.dra w(' mpl')
qc.h(2)
qc.mea sur e(2 ,cr x[1])
qc.dra w(' mpl')
qobj = assemb le(qc)
```

Quantum Teleportation (cont)

```
result = sim.ru n(q obj ).r esult()
counts = result.ge t_c ounts()
plot_h ist ogr am( counts)
```

QKD with BB84 protocol

```
from qiskit import *
from qiskit.co mpiler import transpile, assemble
from qiskit.to ols.ju pyter import *
from qiskit.vi sua liz ation import *
import matplo tli b.p yplot as plotter
import numpy as np
from IPytho n.d isplay import display, Math, Latex
import math as m
%matpl otlib inline
from qiskit import *
from qiskit.vi sua liz ation import plot_h ist ogram
%config Inline Bac ken d.f igu re_ format = 'svg'
qc_ab = Quantu mCi rcu it(6,6) #Create a quantum circuit with 6 qubits and 6 classical bits
##ENCODE BIT STRING
#The random bit sequence Alice needs to encode is: 100100, so the first and fourth qubits are flipped from
|0> -> |1>
qc_ab.x(0) #The first qubit is indexed at 0, following Python being zero-i ndexed. From now on it'll be
referred to as qubit 0 and so on.
qc_ab.x(3)
qc_ab.b ar rier()
##ALICE CHOOSES
#Alice randomly chooses to apply an X or an H.
#Note that since the state is already either a |0> or |1>, a Z essent ially leaves the qubit state
unchanged. But let's write it anyway, shall we?
qc_ab.h(0) # or qc.z(0) # switch these based on your own choice
qc_ab.z(1) # or qc.h(1)
qc_ab.z(2) # or qc.h(2)
qc_ab.h(3) # or qc.z(3)
qc_ab.z(4) # or qc.h(4)
qc_ab.h(5) # or qc.z(5)
qc_ab.b ar rier()
##BOB CHOOSES
#Alice sends the qubit sequence to Bob, and Bob randomly chooses measur ements
qc_ab.h(0) # or qc.z(0) # switch these based on your own choice
qc_ab.z(1) # or qc.h(1)
qc_ab.h(2) # or qc.z(2)
```

QKD with BB84 protocol (cont)

```
qc_ab.h(3) # or qc.z(3)
qc_ab.z(4) # or qc.h(4)
qc_ab.z(5) # or qc.h(5)
qc_ab.b ar rier()
##PUBL ICIZE CHOICES
#Alice and Bob publicize their choices and only retain those for which their choices match. In this case: qubits 0,1,3,4.
#Note: techni cally Bob performs the measur ement BEFORE public izing, but we're combining the two here since no one is actually commun ica ting.
qc_ab.m ea sur e(0,0)
qc_ab.m ea sur e(1,1)
qc_ab.m ea sur e(3,3)
qc_ab.m ea sur e(4,4)
#qc_ab.me asu re(2,2) #come back to uncomment these to see what happens to the results after you've run this once
#qc_ab.me asu re(5,5)
qc_ab.d ra w(o utp ut= 'mpl') #let's see what this circuit looks like!
##EXECUTE
result = execut e(q c_ab, Aer.ge t_b ack end ('q asm _si mul ato r') ).r esu lt( ).g et_ cou nts() #We're only making use of the simulator. Refer to [2] to see how you can run this on a real quantum computer.
plot_h ist ogr am( result)
#Same situation but now with an eavesd ropper (Eve)
qc_aeb = Quantu mCi rcu it(6,6) #Create a quantum circuit with 6 qubits and 6 classical bits
##ENCODE BIT STRING
qc_aeb.x(0)
qc_aeb.x(3)
qc_aeb.ba rrier()
##ALICE CHOOSES
qc_aeb.h(0)
qc_aeb.z(1)
qc_aeb.z(2)
qc_aeb.h(3)
qc_aeb.z(4)
qc_aeb.h(5)
qc_aeb.ba rrier()
##EVE CHOOSES
qc_aeb.h(0) #play around with these to see how many states with non-zero probab ilities show up at the end for a fixed set of Alice's and Bob's choices
qc_aeb.z(1)
qc_aeb.h(2)
qc_aeb.h(3)
qc_aeb.z(4)
```

QKD with BB84 protocol (cont)

```
qc_aeb.z(5)
qc_aeb.ba rrier()
##BOB CHOOSES
qc_aeb.h(0)
qc_aeb.z(1)
qc_aeb.h(2)
qc_aeb.h(3)
qc_aeb.z(4)
qc_aeb.z(5)
qc_aeb.ba rrier()
##PUBL ICIZE CHOICES
qc_aeb.me asu re(0,0)
qc_aeb.me asu re(1,1)
qc_aeb.me asu re(3,3)
qc_aeb.me asu re(4,4)
#qc_ae b.m eas ure (2,2) #come back to uncomment these to see what happens to the results after you've run this once
#qc_ae b.m eas ure (5,5)
qc_aeb.dr aw( out put ='mpl') #let's see what this circuit looks like!
##EXECUTE
result = execut e(q c_aeb, Aer.ge t_b ack end ('q asm _si mul ato r') ).r esu lt( ).g et_ cou nts()
plot_h ist ogr am( result)
```

Import Libraries

```
from qiskit import
QuantumRegister ,
ClassicalRegister,
QuantumCircuit , Aer , execute
from qiskit.vi sua liz ation
import visual ize _tr ans ition
import numpy as np
import math as m
```