Task 1-Swap Test

Aris Skliros

February 11, 2021

Contents

1 Variational Circuit

The Variational circuit which is able to generate the most general 1 qubit state is the circuit that renders the following product of Unitaries:

$$\begin{aligned} U_{circuit} &= U_1 \times U_2 \\ U_{circuit} &= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} \times \begin{pmatrix} cos\left(\frac{\theta}{2}\right) & -sin\left(\frac{\theta}{2}\right) \\ sin\left(\frac{\theta}{2}\right) & cos\left(\frac{\theta}{2}\right) \end{pmatrix} \end{aligned} \tag{1}$$

where in this context the symbol $\times$ stands for the matrix multiplication. If the matrix in Equation 1 is premultiplied to the zero qubit state $|0\rangle$ where $|0\rangle$ can be expressed as:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2}$$

then we have that the output quantum state, $|Out\rangle$ is:

$$\begin{aligned} |Out\rangle &= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} \times \begin{pmatrix} cos\left(\frac{\theta}{2}\right) & -sin\left(\frac{\theta}{2}\right) \\ sin\left(\frac{\theta}{2}\right) & cos\left(\frac{\theta}{2}\right) \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ |Out\rangle &= \begin{pmatrix} cos\left(\frac{\theta}{2}\right) \\ e^{i\phi} sin\left(\frac{\theta}{2}\right) \end{pmatrix} = cos\left(\frac{\theta}{2}\right) \begin{pmatrix} 1 \\ 0 \end{pmatrix} + e^{i\phi} sin\left(\frac{\theta}{2}\right) \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ |Out\rangle &= cos\left(\frac{\theta}{2}\right) |0\rangle + e^{i\phi} sin\left(\frac{\theta}{2}\right) |1\rangle \end{aligned} \tag{3}$$

In Equation 3, if $\phi \in [0, 2\pi]$ and $\theta \in [0, \pi]$ then the resulting Qubit spans the whole Bloch Sphere [1].

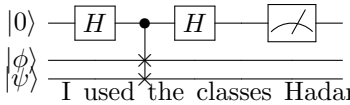
The Circuit can be represented, with the help of the Qcircuit package [2], as:

$|0\rangle$ — $\begin{pmatrix} cos\left(\frac{\theta}{2}\right) & -sin\left(\frac{\theta}{2}\right) \\ sin\left(\frac{\theta}{2}\right) & cos\left(\frac{\theta}{2}\right) \end{pmatrix}$ — $\begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$ — $|Out\rangle$

In file *project.py* I produced the class *Variational* which invokes classes *Rmatrix* and *Rz*.

2 Single Qubit Swap Test

I implemented the SWAP Gate Circuit in file *Project.py* in class SWAPGateCircuit.



I used the classes HadamardGate and SWAPGate. I then implemented the class SwapTest which takes as inputs the Ancilla Qubit, the ψ Qubit $|\psi\rangle$ and the ϕ Qubit $|\phi\rangle$ and renders as output te output of SwapGateCircuit applied to the Kronecker product of those 3 Qubits. I developed also the Simulation class. This generates the ψ Qubit by generating randomly θ, ϕ and by using the class Variational. We apply the circuit of the class Variational on $|0\rangle$ Qubit. We have thus the randomly generates $|\psi\rangle$. Then we invoke the *applySimulation* function of the class *Simulation*. We generate 100 values for θ and ϕ, 10000 cases and we generate thus candidates (in each simulation) of $|\phi\rangle$ by using the Variational Class. We invoke the SwapTest class by having as inputs the Ancilla, the $|\psi\rangle$ which we randomly generate and the $|\phi\rangle$ and we get the Output. The output is a 8×1 vector. In the exercise we need to simulate many times to find the probability of getting $|0\rangle$ for the output Ancilla. We can find however this just by getting the first 4 elements of the output. Those are the ones corresponding to measuring the Ancilla as $|0\rangle$. This is because they correspond to $000, 001, 010, 011$. Then we take the product of this subvector with its complex conjugate and we have the probability of measuring the Ancilla to 0. For the $|\psi\rangle, |\phi\rangle$ to be equal this probability has to be 1. We, in the program, stop the simulation when the probability is close to 1 with three decimals as precision. Let us demonstrate the code analytically:

- This is the code for the Swap test Circuit:

```
class SwapTest:
        def __init__(self,AncillaQubit,PsiQubit,PhiQubit):
                self.AncillaQubit=AncillaQubit
                self.PsiQubit=PsiQubit
                self.PhiQubit=PhiQubit
                self.SwapGate=SWAPGateCircuit()
        def applyGate(self):
                self.SwapGate.applyGate()
                self.Circuit=self.SwapGate.OperationMatrix
                self.Output=np.matmul(self.Circuit,
                np.kron(self.AncillaQubit,
                np.kron(self.PsiQubit,self.PhiQubit)))
```

- This is the part of the code in the Class Simulation for randomly generating the Ψ Qubit(and then wanting to replicate it with Φ Qubit).

```
def generatePsiQubit(self):
#Generate a Quantum State randomly:
#Defining the Qubit Psi randomly:
# RandomNumber=np.random.randint(0,2)
# if(RandomNumber==1):
#     self.PsiQubit=np.array([0,1])
# elif(RandomNumber==0):
#     self.PsiQubit=np.array([1,0])
# else:
#     self.PsiQubit=np.array([1,0])
#The qubit Psi is the randomly generated quantum state
theta0=np.random.uniform(0,math.pi)
phi0=np.random.uniform(0,2*math.pi)
Var0=Variational(theta0,phi0)
Var0.applyGate()
#Finding the Qubit Psi
self.PsiQubit=np.matmul(Var0.OperationMatrix,np.array([1,0]))
print("The randomly generated PsiQubit is:")
print(self.PsiQubit)
self.applySimulation()
```

- This is the part of the code in the Simulation Class that renders the Simulation. It searches 10000 cases for the Φ Qubit, 10000 different combinations of angles θ and ϕ until it gets the values thereof where the probability of measuring the zero Ancilla qubit is 1 [3]. Let us execute the function generatePsiQubit of the class Simulation once: This function generates randomly a $|\psi\rangle$ and then it invokes the applySimulation function which tries to find the pertinent angles by simulation to replicate the $|\psi\rangle$ by $|\phi\rangle$. we present the code in lines 199-201 and the results

```
Sim=Simulation()
Sim.generatePsiQubit()
Sim.Result

The randomly generated PsiQubit is:
[0.97009689+0.j          0.12588372+0.20752182j]
Yes
The approximated PhiQubit is:
[0.97236992+0.j          0.12508628+0.19710444j]
```

one more time:

```
The randomly generated PsiQubit is:
[ 0.98737   +0.j          -0.15221141+0.04395636j]
Yes
The approximated PhiQubit is:
[1.+0.j 0.+0.j]
```

3 Multi-Qubit Swap Test

Within the Simulation Class we create the function *generateNQubitState* for N qubits. For each qubit we perform what we did in Section 2. We create the N-Qubit state $|\psi\rangle$ and its approximation $|\phi\rangle$. This is the code:

```
def generateNQubitState(self,N):
self.NQubitPsiState=[]
self.NQubitPhiState=[]
self.NQubitPsi=np.array([1,0])
self.NQubitPhi=np.array([1,0])
for kkk in range(0,N):
theta0=np.random.uniform(0,math.pi)
phi0=np.random.uniform(0,2*math.pi)
Var0=Variational(theta0,phi0)
Var0.applyGate()
#Finding the Qubit Psi
self.PsiQubit=np.matmul(Var0.OperationMatrix,np.array([1,0]))
self.NQubitPsiState.insert(len(self.NQubitPsiState),self.PsiQubit)
print("The randomly generated PsiQubit is:")
print(self.PsiQubit)
#We now approximate the PhiQubit and we insert it in the list
self.applySimulation()
self.NQubitPhiState.insert(len(self.NQubitPhiState),self.PhiQubit)
if(kkk==0):
self.NQubitPsi=np.copy(self.PsiQubit)
self.NQubitPhi=np.copy(self.PhiQubit)
else:
self.NQubitPsi=np.kron(self.NQubitPsi,self.PsiQubit)
self.NQubitPhi=np.kron(self.NQubitPhi,self.PhiQubit)
```

In the code above N is the number of Qubits. The *applySimulation* function is invoked N times each time for each qubit. We ensure that the N Qubit states is not entangled by having them as Kronecker products of single Qubit states see definitions of $self.NQubitPsi, self.NQubitPhi$ respectively. We run the lines 199, 202 of the code for the case of N=10 Qubit states:

```
The randomly generated PsiQubit is:
[ 0.99683555+0.j          -0.03774287-0.06995967j]
Yes
The approximated PhiQubit is:
[ 0.99691733+0.j          -0.0461171 -0.06347474j]
The randomly generated PsiQubit is:
[ 0.32314002+0.j          -0.398735  -0.85824876j]
Yes
The approximated PhiQubit is:
[ 0.32391742+0.j          -0.40282355-0.85604363j]
The randomly generated PsiQubit is:
[0.99059381+0.j           0.12428988-0.05723568j]
```

```
Yes
The approximated PhiQubit is:
[0.9921147+0.j         0.1134049-0.0533643j]
The randomly generated PsiQubit is:
[0.88766194+0.j          0.35573181+0.29242292j]
Yes
The approximated PhiQubit is:
[0.89100652+0.j          0.34980569+0.28938444j]
The randomly generated PsiQubit is:
[ 0.97934122+0.j          -0.01946328-0.20127583j]
Yes
The approximated PhiQubit is:
[ 0.97922281+0.j          -0.02541599-0.20118826j]
The randomly generated PsiQubit is:
[0.51995902+0.j          0.84025645+0.15366105j]
Yes
The approximated PhiQubit is:
[0.52249856+0.j          0.83753756+0.15976883j]
The randomly generated PsiQubit is:
[ 0.33026634+0.j          -0.44552942+0.83212239j]
Yes
The approximated PhiQubit is:
[ 0.33873792+0.j          -0.45327277+0.8245001j]
The randomly generated PsiQubit is:
[ 0.78708485+0.j          -0.61070218-0.08683485j]
Yes
The approximated PhiQubit is:
[ 0.79015501+0.j          -0.6080741 -0.07681762j]
The randomly generated PsiQubit is:
[0.99875568+0.j          0.02041403-0.04550118j]
Yes
The approximated PhiQubit is:
[0.99888987+0.j          0.00882687-0.04627207j]
The randomly generated PsiQubit is:
[ 0.46448059+0.j          -0.67744776-0.57037033j]
Yes
The approximated PhiQubit is:
[ 0.46792981+0.j          -0.68095312-0.56333341j]
```

References

[1] Nielsen M. A., Chuang I.L., Quantum Computation and Quantum Information, 10th Anniversary Edition, Cambridge University Press, 2010, ISBN 978-1-107-00217-3

[2] qcircuit 2.6.0 Tutorial Bryan Eastin, Steve T Flammia Edits: Travis L Scholten https://github.com/CQuIC/qcircuit.

[3] https://en.wikipedia.org/wiki/Swap_test

A Manifesting that our measurement works

Assume that we have a one Qubit Circuit. The input thereof is the state 0, $|0\rangle$, the Unitary gate is a Black Box , i.e. we do not know what it is apart from being a Unitary, and the output is a Random Qubit $|Out\rangle = a|0\rangle + b|1\rangle$.

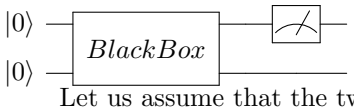
$|0\rangle$ — $\boxed{BlackBox}$ — $|Out\rangle$

We implement this in Python, without using any simulation. This will render a 2×1 vector. This vector will be:

$$a|0\rangle + b|1\rangle = a\begin{pmatrix}1\\0\end{pmatrix} + b\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}a\\b\end{pmatrix} \tag{4}$$

From the above equation, it is obvious that the probability of measuring $|0\rangle$ for $|Out\rangle$ is $a^\dagger a$ and the probability of measuring $|1\rangle$ for $|Out\rangle$ is $b^\dagger b$.

Assume now that we have a two Qubit circuit, with a Black Box Unitary applied on two qubits now:



Let us assume that the two output Qubits are

$$\begin{aligned}|Out_1\rangle &= a|0\rangle + b|1\rangle\\ |Out_2\rangle &= c|0\rangle + d|1\rangle\end{aligned} \tag{5}$$

The total output of the circuit is a 4×1 vector and is given by:

$$|Out\rangle = |Out_1\rangle \otimes |Out_2\rangle = (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle) = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle \tag{6}$$

Now we know

$$|00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix} \quad |01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix} |10\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix} \quad |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix} \tag{7}$$

So the $|Out\rangle$ becomes:

$$|Out\rangle = \begin{pmatrix}ac\\ad\\bc\\bd\end{pmatrix} \tag{8}$$

As we notice the case where the output qubit measures to $|0\rangle$ corresponds to the coefficient a. So, the first two elements of the output vector are the ones corresponding to output qubit 1 measuring to $|0\rangle$. Likewise for the case of three qubits.