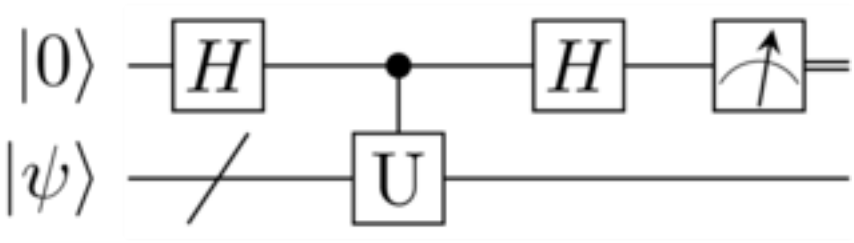
Part 1

Our goal is to simulate the hadamard test in python:



Methodology

First we must define some variables that will be applied repeatedly during a matrix operations:

```
#Definitions
ket_zero = np.array([1,0])
bra_zero = np.conj(ket_zero).T #Transpose conjugate vector
ket_one = np.array([0,1])
bra_one = np.conj(ket_one).T #Transpose conjugate vector
identity_matrix = [[1,0],[0,1]]
c1 = (1/2**0.5)
```

We define $|0\rangle$ and $|1\rangle$ by manually declaring their arrays. We define $\langle 0|$ and $\langle 1|$ by transposing the conjugate of their respective vectors. We also define the identity matrix: $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, and the fraction $1/\sqrt{2}$.

The formula from our hadamard gate is taken from the slides:

```
#Gates
h_gate = c1 * np.array([[1,1],[1,-1]])
```

Our code that simulates the full quantum circuit is as follows:

```
16 def simulate(u_gate):
17     #Calculate the state of both qubits to start
18     qubit_0 = ket_zero.copy()
19     qubit_1 = ket_zero.copy()
20
21     #Apply the Hadamard gate to the first qubit
22     qubit_0 = np.matmul(h_gate,qubit_0)
23
24     #Take the tensor product of both qubits to calculate the overall state
25     state_s0 = np.kron(qubit_0, qubit_1)
26
27     #Apply the controlled u gate to the state
28     state_s0 = np.matmul(u_gate, state_s0)
29
30     #Apply the second Hadamard gate to the first qubit only (using H otimes I)
31     state_s0 = np.matmul(np.kron(h_gate, identity_matrix), state_s0)
32
33     #Measure
34     p0 = np.kron(np.outer(ket_zero, bra_zero), identity_matrix)
35     probability_0 = linalg.norm(np.matmul(p0, state_s0))**2
36
37     return probability_0
```

On lines 18-19, we initialize both qubits.

On line 22, we apply a hadamard gate to the first qubit. Since the first qubit is $|0\rangle$, this sets the state of the first qubit to $1\sqrt(2)(|0\rangle + |1\rangle)$

One line 25, we tensor the first qubit with the second qubit in order to get the state of the quantum circuit in preparation for the controlled U gate. This gives us the state $1\sqrt(2)(|0\rangle + |1\rangle) \otimes |\psi\rangle$

The controlled U gate only applies the unitary operator U to $|\psi\rangle$ if the controlled qubit is $|1\rangle$. Since we position the hadamard gate on qubit 1, we have a 50% chance for the controlled U gate to apply. We construct our U gate in the main function.

```
#Calculate the controlled u gate using the formula from the slides and the rotation matrix
u = [[np.cos(2*np.pi*theta), -np.sin(2*np.pi*theta)],[np.sin(2*np.pi*theta), np.cos(2*np.pi*theta)]]
u_gate = np.kron(np.outer(ket_zero, bra_zero), identity_matrix) + np.kron(np.outer(ket_one, bra_one), u)
```

U is given by the rotation matrix (`wikipedia.org/wiki/Rotation_matrix`), where the U gate is given by the formula from the slides: $|0\rangle \langle 0| \otimes I + |1\rangle \langle 1| \otimes U$

On line 28, we apply the U gate to the state (both qubits).

One line 31, we apply another hadamard gate to just the first qubit. Since

state_s0 contains both qubits, we must apply the hadamard gate only to the first qubit portion of *state_s0*. This can be done by taking $H \otimes I$.

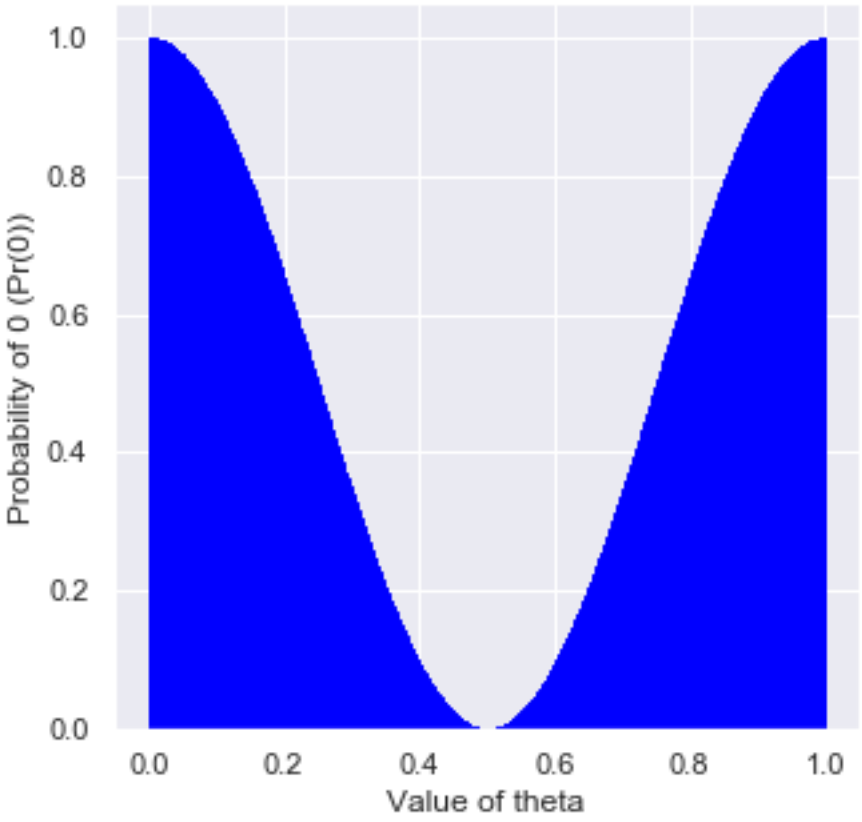
Finally, on line 34, we measure the state. We know from the slides that $Pr(0) = ||P_0 |\psi\rangle||^2$, where $P_0 = |0\rangle \langle 0| \otimes I$

Tests

Our main function tests 1000 evenly distributed values of θ, and plots the results into a bar graph.

```
39 def main():
40     pr_0 = []
41     for i in range(1000):
42 
43         #Increment theta in steps of 1/1000
44         theta = (i/1000)
45 
46         #Calculate the controlled u gate using the formula from the slides and the rotation matrix
47         u = [[np.cos(2*np.pi*theta), -np.sin(2*np.pi*theta)],[np.sin(2*np.pi*theta), np.cos(2*np.pi*theta)]]
48         u_gate = np.kron(np.outer(ket_zero, bra_zero), identity_matrix) + np.kron(np.outer(ket_one, bra_one), u)
49 
50         #Save result
51         pr_0.append(simulate(u_gate))
52 
53     plt.bar([(x/1000) for x in range(1000)], pr_0, 0.001, edgecolor='blue')
54     plt.ylabel("Probability of 0 (Pr(0))")
55     plt.xlabel("Value of theta")
56     plt.savefig('dist.png')
57     plt.show()
```

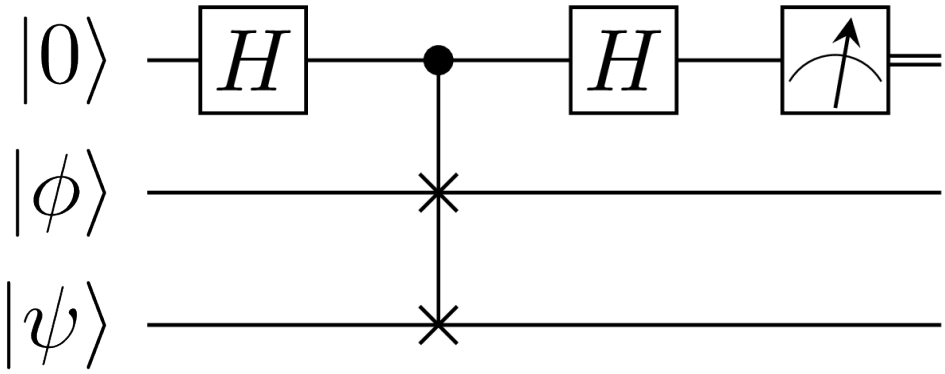
By running this program, we get the following output:



We can see from the graph that $Pr(0) = 1$ when theta = 0,1; and that $Pr(0) = 0$ when theta = .5

Part 2

Our goal is to simulate the swap test in python:



Methodology

Our definitions for part 2 of the assignment are as follows:

```
#Definitions
ket_zero = np.array([1,0])
bra_zero = np.conj(ket_zero).T #Transpose conjugate vector
identity_matrix = [[1,0],[0,1]]
c1 = (1/2**0.5)

#Gates
h_gate = c1 * np.array([[1,1],[1,-1]])
swap_gate = np.array(
        [[1,0,0,0,0,0,0,0],
         [0,1,0,0,0,0,0,0],
         [0,0,1,0,0,0,0,0],
         [0,0,0,1,0,0,0,0],
         [0,0,0,0,1,0,0,0],
         [0,0,0,0,0,0,1,0],
         [0,0,0,0,0,1,0,0],
         [0,0,0,0,0,0,0,1]])
```

We update our simulate function to execute the swap test instead of the hadamard test.

```
23 def simulate(theta):
24     #Calculate the state of both qubits to start
25     qubit_0 = ket_zero.copy()
26     qubit_1 = ket_zero.copy()
27     qubit_2 = np.cos(2*np.pi*theta)*ket_zero + np.sin(2*np.pi*theta)*ket_one
28 
29     #Apply the Hadamard gate to the first qubit
30     qubit_0 = np.matmul(h_gate,qubit_0)
31 
32     #Take the tensor product of both qubits to calculate the overall state
33     state_s0 = np.kron(np.kron(qubit_0, qubit_1), qubit_2)
34 
35     #Apply the controlled swap gate to the state
36     state_s0 = np.matmul(swap_gate, state_s0)
37 
38     #Apply the second Hadamard gate to the first qubit only (using H otimes I otimes I)
39     h_q0 = np.kron(np.kron(h_gate, identity_matrix), identity_matrix)
40     state_s0 = np.matmul(h_q0, state_s0)
41 
42 
43     #Measure
44     p0 = np.kron(np.outer(ket_zero, bra_zero), np.eye(4))
45     probability_0 = linalg.norm(np.matmul(p0, state_s0))**2
46 
47     return probability_0
```

On lines 25-27, we initialize our qubits.

Line 30 is the same as the hadamard test, we simply apply the hadamard gate to the first qubit.

On line 32, we take the tensor product of all three qubits together, once again getting the overall state of the quantum register. At this point, the state of the system is: $1\sqrt{(2)}(|0,\psi_1,\psi_2\rangle + |1,\psi_1,\psi_2\rangle)$.

On line 36, we apply the controlled swap gate, which swaps the value of the second and third qubits only if the first qubit is 1, giving us: $1\sqrt{(2)}(|0,\psi_1,\psi_2\rangle + |1,\psi_2,\psi_1\rangle)$.

On line 39, we again use a projector to extract the value of the first qubit. This time, we use the identity matrix of size 4, since we have a register of size 3 and we need the value of qubit 1 ($I_{2^{3-1}}$). In numpy, np.eye(x) gives us the identity matrix of size x.

Tests

Our main function is very similar to part 1:

```
49 def main():
50     pr_0 = []
51     for i in range(1000):
52         #Save result
53         pr_0.append(simulate(i/1000))
54 
55     plt.bar([(x/1000) for x in range(1000)], pr_0, 0.001, edgecolor='blue')
56     plt.ylabel("Probability of 0 (Pr(0))")
57     plt.xlabel("Value of theta")
58     plt.savefig('dist.png')
59     plt.show()
```

In this version, we pass θ to the simulate function in equally distributed increments of $1/1000$, in the range $[0, 1]$.

Executing this code gives the following plot: