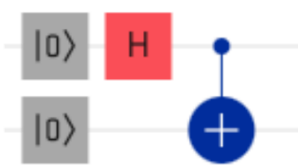
James Amidei Dr. Tim Lei PHYS 4681 - Quantum Computing Algorithms Homework 1

```
import cirq
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

Table of Contents

Question 1

1) For the following quantum circuit discussed in class,



a) Use the Google Cirq library to calculate the output quantum wavefunction.
b) Add two measurement gates to the two qubits at the end of the quantum circuit and simulate 10,000 times for the measurement. Tally the measurement results and plot the measurement histogram.
c) Now, using Python and NumPy alone, build your own quantum computing simulator to repeat a) and b).
d) Compare the results obtained from CIRQ and from your own simulator to check whether there are any discrepancies in the simulated results.

1.a.

- Table of Contents

i.) Create two qubits: q0 and q1 and initialize quantum circuit.

ii.) Apply Hadamard gate to q0 to turn it into a superposition state.

$$ H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \rightarrow \frac{1}{\sqrt{2}} (|0\rangle \langle 0| + |1\rangle \langle 0| + |0\rangle \langle 1| - |1\rangle \langle 1|) \hspace{0.5cm} \text{(Hadamard gate)} $$

$$H\Big|0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle\langle 0|+|1\rangle\langle 0|+|0\rangle\langle 1|-|1\rangle\langle 1|\right)\Big)0\rangle$$

$$\text{¿}\frac{1}{\sqrt{2}}\left(|0\rangle\langle 0|0\rangle+|1\rangle\langle 0|0\rangle+|0\rangle\langle 1|0\rangle-|1\rangle\langle 1|0\rangle\right)$$

$$ \biggr(\langle 0 | 0 \rangle = 1 \hspace{0.5cm} \text{\&} \hspace{0.5cm} \langle 1|0 \rangle = 0 \biggr) \hspace{0.5cm} \text{(by orthogonality)} $$

$$H \vee 0\rangle = \frac{1}{\sqrt{2}}\text{¿}$$

iii.) Apply C_{not} gate, using q0 as the control and q1 as the target, creating an entangled state for the system.

$$ C_{not} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \rightarrow |00\rangle \langle00| + |01 \rangle \langle 01| + |11 \rangle \langle 10| + |10 \rangle \langle 11| \hspace{0.5cm} \text{(Cnot gate)}$$

First, take tensor product of superposition state q0 and q1.

$$H|0\rangle \otimes |0\rangle = \frac{1}{\sqrt{2}}\text{¿}$$

$$ \biggr(|0\rangle \otimes |0\rangle \sim |00\rangle \hspace{0.5cm} \text{\&} \hspace{0.5cm} |1\rangle \otimes |0\rangle \sim |10\rangle) \biggr)$$

$$H|0\rangle \otimes |0\rangle = \frac{1}{\sqrt{2}}\text{¿}$$

Apply C_{not} gate to entangled state.

$$C_{not}\left(H|0\rangle \otimes |0\rangle\right) = C_{not}\text{¿}$$

$$\text{¿}\left(|00\rangle\langle 00|+|01\rangle\langle 01|+|11\rangle\langle 10|+|10\rangle\langle 11|\right)\text{¿}$$

$$\text{¿}\frac{1}{\sqrt{2}}\left(|00\rangle\langle 00|00\rangle+|01\rangle\langle 01|00\rangle+|11\rangle\langle 10|00\rangle+|10\rangle\langle 11|00\rangle\right)+\frac{1}{\sqrt{2}}\left(|00\rangle\langle 00|10\rangle+|01\rangle\langle 01|10\rangle+|11\rangle\langle 10|10\right.$$

$$ \biggr(\langle00|00\rangle = \langle10|10\rangle = 1 \hspace{0.5cm} \text{\&} \hspace{0.5cm} \langle 01|00\rangle = \langle 10|00\rangle = \langle 11|00\rangle = \langle 00|10 = \langle 01|10\rangle = \langle 11|10\rangle = 0\biggr)$$

So, the final wavefunction will be,

$$C_{not}\left(H|0\rangle \otimes |0\rangle\right) = \frac{1}{\sqrt{2}} ¿$$

with the two states $¿00\rangle$ and $¿11\rangle$ having equal probability amplitudes of $\frac{1}{\sqrt{2}} \approx 0.707$.

1.a. Code Cell

- Table of Contents

```
q0 = cirq.GridQubit(0, 0) # creates qubits that are stored in a grid.
q1 = cirq.GridQubit(0, 1) # based on what I've read, is more
applicable.

circuit = cirq.Circuit() # initialize empty circuit

circuit.append(cirq.H(q0)) # Hadamard to q0
circuit.append(cirq.CNOT(q0, q1)) # CNOT, q0 is control, q1 is target

print(circuit) # print visualization of the circuit

sim = cirq.Simulator() # create simulator
result = sim.simulate(circuit) # execute simulation of quant. circuit

basis_states = [f"|{x:02b}>" for x in range(4)] # Cirq indexes the
basis states with an integer values. This creates
# a list that matches each integer index with it's binary string
equivalent with Dirac notation brackets around it
# so that each basis state has the proper label.

state_map = [(basis_states[i], amp) for i, amp in
enumerate(result.final_state_vector)] # creates a list where the
corrected name
# of each basis state is indexed together with its probability
amplitude.

print('\nWavefunction: ') # print states and prob. amplitudes
for state, amp in state_map:
    print(f'{state}: {amp}')
```

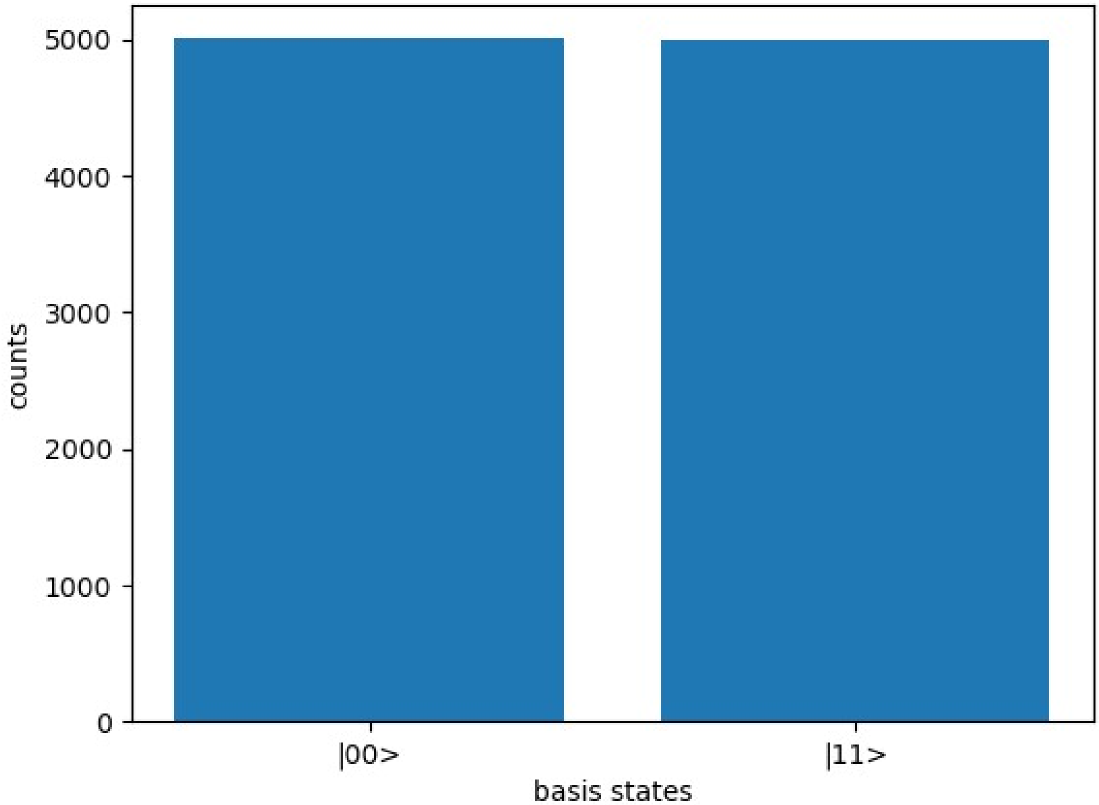
```
(0, 0): ───H───@───
               │
(0, 1): ───────X───

Wavefunction:
|00>: (0.7071067690849304+0j)
|01>: 0j
|10>: 0j
|11>: (0.7071067690849304+0j)
```

1.b.

- Table of Contents


```
circuit = cirq.Circuit() # resetcircuit after running the simulation
inthe cell above.
circuit.append(cirq.H(q0))
circuit.append(cirq.CNOT(q0, q1))

circuit.append(cirq.measure(q0, q1, key='measurement')) # add
measurement to end

print(circuit)
print()

results = sim.run(circuit, repetitions=10000)
#result
counts = results.histogram(key='measurement')
from_cirq_results_dict = {basis_states[outcome]: count for outcome,
count in counts.items()}

#print(from_cirq_results_dict)

for state, amp in from_cirq_results_dict.items():
    print(f"{state}: {amp}")

plt.bar(from_cirq_results_dict.keys(),
from_cirq_results_dict.values()) # plot histogram
plt.xlabel('basis states')
plt.ylabel('counts')
plt.show()
```

```
(0, 0): ───H───@───M('measurement')───
               │   │
(0, 1): ───────X───M──────────────────

|00>: 5004
|11>: 4996
```



1.c.

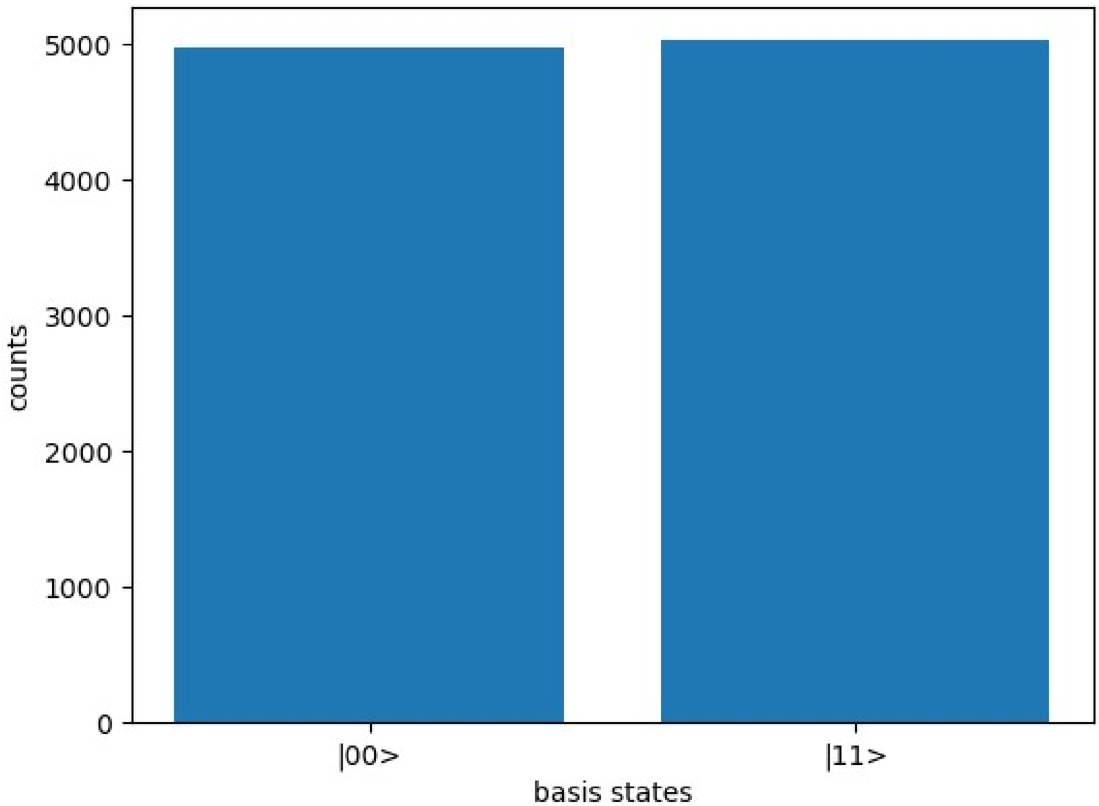
- Table of Contents

```
# this was my first attempt at constructing a sim from scratch.

H = 1/np.sqrt(2)*np.array([[1, 1], # Hadamard Gate
                          [1, -1]])

CNOT = np.array([[1, 0, 0, 0], # CNOT Gate
                [0, 1, 0, 0],
                [0, 0, 0, 1],
                [0, 0, 1, 0]])

q0 = np.array([[1],
               [0]]) # first qubit in state |0>

q1 = np.array([[1],
               [0]]) # second qubit in state |0>

#q = print(np.kron(q0, q1))

q0 = H @ q0 # act H on q0 to get superposition of q0
#print(f"q0 = {q0}")

q = np.kron(q0, q1) # combine q0 and q1, not yet entangled; i.e.,
```

```
still separable.

qc = CNOT @ q # final entangled state

print('\nWavefunction: ')
for i in range(4):
    print(f"{basis_states[i]}: {qc[i]}")

prob_amp = np.abs(qc.flatten())**2 # probabilities of basis states
# square the probability amplitude of each basis state in qc and
converts it into a 1D array
# This is important for the use of the numpy.random.choice() function,
which needs a probability
# input in the form of a 1D array.

#basis_states = [f"|{x:02b}>" for x in range(4)]

meas = np.random.choice(basis_states, size=10000, p=prob_amp) # picks
random states from basis_states list
# each of which have a probability assigned to them from the prob_amp
array. This then runs 1e4 times.

unique, counts = np.unique(meas, return_counts=True) # numpy.unique()
finds all unique measurement results
# return_counts=True returns how many times each state appears.
results = dict(zip(unique, counts)) # pairs each basis_state entry
with its count and creates a dictionary

print('\nState: Counts') # prints counts per state and the percentage
of total events
for key, value in results.items():
    print(f"{key}: {value}")
    print(f"{value/100}% of total reps")

from_scratch_results_dict = results.copy()  # copy results to keep
them for later use

print()
plt.bar(results.keys(), results.values()) # plot histogram
plt.xlabel('basis states')
plt.ylabel('counts')
plt.show()
```

```
Wavefunction:
|00>: [0.70710678]
|01>: [0.]
|10>: [0.]
|11>: [0.70710678]

State: Counts
```

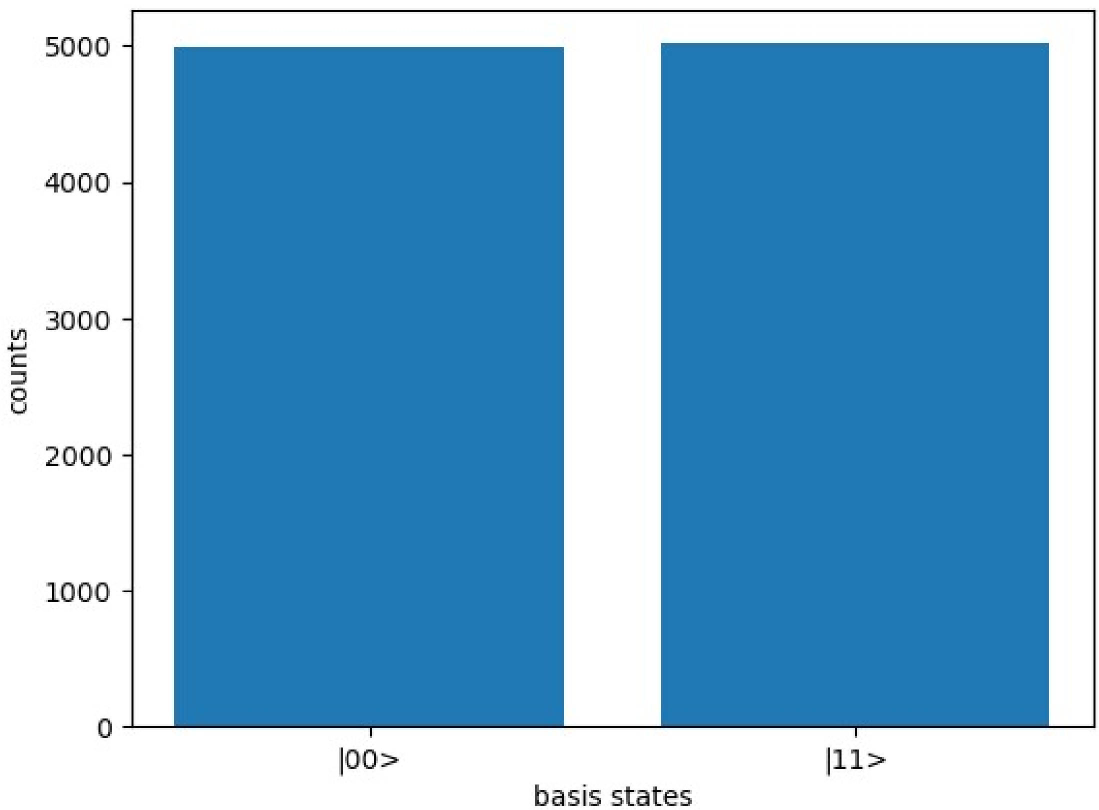
```
|00>: 4973
49.73% of total reps
|11>: 5027
50.27% of total reps
```




```
# alternate method, following method in class notes

H = 1/np.sqrt(2)*np.array([[1, 1],
                           [1, -1]])

I = np.eye(2) # Identity matrix

CNOT = np.array([[1, 0, 0, 0],
                [0, 1, 0, 0],
                [0, 0, 0, 1],
                [0, 0, 1, 0]])

q0 = np.array([[1],
               [0]])

q1 = np.array([[1],
               [0]])
```

```python
q = np.kron(q0, q1) # combine the two-qubits and treating that as the initial state
# this makes a 2x1 column vector

H = np.kron(H, I) # scaling up H so that the tensor product of H and q can be taken

q = H @ q

qc = CNOT @ q

print('\nWavefunction: ')
for i in range(4):
    print(f"{basis_states[i]}: {qc[i]}")

prob_amp = np.abs(qc.flatten())**2

meas = np.random.choice(basis_states, size=10000, p=prob_amp)

unique, counts = np.unique(meas, return_counts=True)
results = dict(zip(unique, counts))

print('\nState: Counts')
for key, value in results.items():
    print(f"{key}: {value}")
    print(f"{value/100}% of total reps")

from_scratch_results_dict2 = results.copy()

print()
plt.bar(results.keys(), results.values())
plt.xlabel('basis states')
plt.ylabel('counts')
plt.show()
```

```
Wavefunction:
|00>: [0.70710678]
|01>: [0.]
|10>: [0.]
|11>: [0.70710678]

State: Counts
|00>: 4984
49.84% of total reps
|11>: 5016
50.16% of total reps
```



1.d.


```
print(f"From Cirq simulation: {from_cirq_results_dict}")
print(f"From Numpy simulation: {from_scratch_results_dict}")

From Cirq simulation: {'|00>': 5010, '|11>': 4990}
From Numpy simulation: {'|00>': 4914, '|11>': 5086}
```

For a simple 2-qubit system, the results of the CIRQ simulation and the constructed Numpy simulation seem to work just as well as each other. This ought to be true for any system with a relatively small number of qubits. However, when we start to scale the number of qubits, we will quickly find that the size of our operators, and the number of tensor products increases. This introduces greater numerical error, meaning that our simulations will stop being as accurate. If we wanted to scale this up, we ought to treat the initial state as one column vector with n-number of entries. We then have to take the tensor product of the Hadamard gate with the identity matrix n-1 times, causing the Hadamard to scale to a $2^n \times 2^n$ matrix. We then have to apply the CNOT gate n-1 time to each qubit for each qubit that we use as the control to every other qubit as the target.

Question 2

2) Come up with a quantum circuit to create an entangled wavefunction for a 4-qubit system.

$$|\Psi \geq \frac{1}{\sqrt{2}}(|0000 > +|1111 >)$$

Write your own simulator and compare the simulated results using CIRQ, as in problem #1.

- Table of Contents

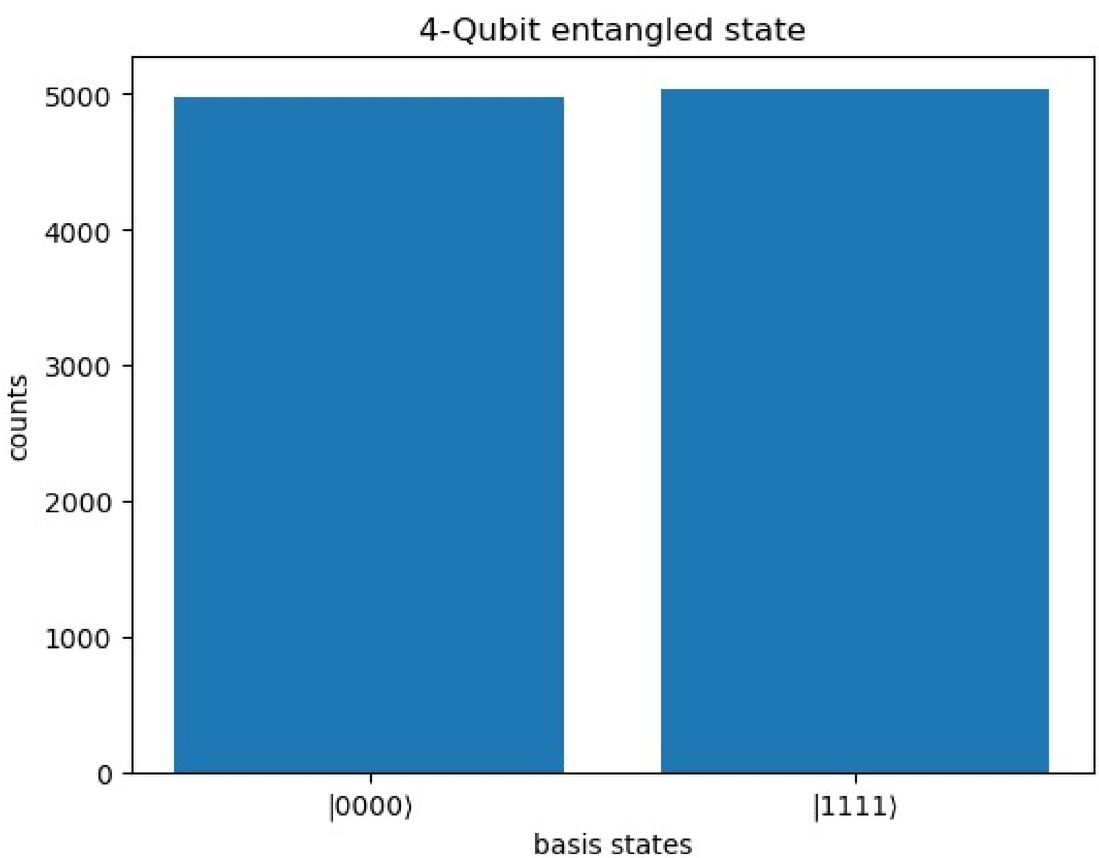
For a 4-qubit system, we can get the entangled wavefunction by applying the Hadamard gate to the first qubit, and then applying the CNOT to the other 3 qubits, with q0 as the control and the other 3 as the targets.

2 code cell 1

```
q0 = cirq.GridQubit(0,0)
q1 = cirq.GridQubit(0,1)
q2 = cirq.GridQubit(0,2)
q3 = cirq.GridQubit(0,3)

circuit = cirq.Circuit()

circuit.append(cirq.H(q0))  # Hadamard on first qubit

circuit.append(cirq.CNOT(q0, q1))  # CNOT gates
circuit.append(cirq.CNOT(q0, q2))
circuit.append(cirq.CNOT(q0, q3))

circuit.append(cirq.measure(q0, q1, q2, q3, key='measurement'))  # Measure

print(circuit)
print()

simulator = cirq.Simulator()
results = simulator.run(circuit, repetitions=10000)

counts = results.histogram(key='measurement')

basis_states = [f"|{x:04b}⟩" for x in range(2**4)]

from_cirq_results_dict_4 = {basis_states[outcome]: count for outcome, count in counts.items()}

# Print histogram data
print("\nMeasurement results:")
```

```python
for state, amp in from_cirq_results_dict_4.items():
    print(f"{state}: {amp}")

plt.bar(from_cirq_results_dict_4.keys(),
from_cirq_results_dict_4.values())
plt.xlabel('basis states')
plt.ylabel('counts')
plt.title('4-Qubit entangled state')
plt.show()
```

```
(0, 0): ───H───@───@───@───M('measurement')───
               │   │   │   │
(0, 1): ───────X───┼───┼───M──────────────────
                   │   │   │
(0, 2): ───────────X───┼───M──────────────────
                       │   │
(0, 3): ───────────────X───M──────────────────


Measurement results:
|0000⟩: 4970
|1111⟩: 5030
```

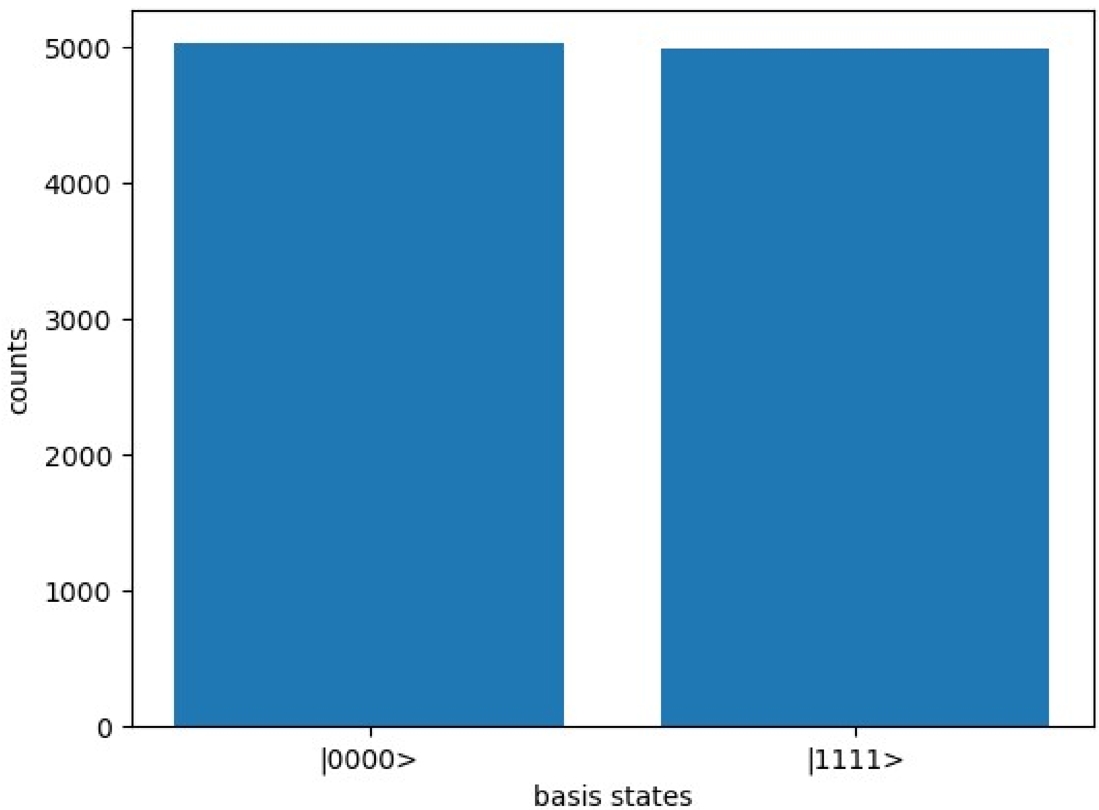


2 code cell 2

-

```
n = 4 # number of qubits
psi_init = np.zeros(2**n, dtype=complex) # empty wavefunction
psi_init[0]=1

H = 1/np.sqrt(2)*np.array([[1,1], # Hadamard matrix
                           [1,-1]])

I = np.eye(2) # Identity matrix

def cnot_n(n, control, target): # This was written in collaboraton with Daniel Mazin
    m = 2**n
    CNOT = np.zeros((m,m))
    for i in range(m):
        b = list(np.binary_repr(i, width=n))
        if b[control] == '0':
                 j = i
        else:
            b[target] = '1' if b[target] == '0' else '0'
            j = int("".join(b), 2)
        CNOT[j, i] = 1
    return CNOT

H_4 = np.kron(np.kron(np.kron(H, I), I), I)

CNOT1 = cnot_n(n, 0, 1) # CNOT q0->q1
CNOT2 = cnot_n(n, 0, 2) # CNOT q0->q2
CNOT3 = cnot_n(n, 0, 3) # CNOT q0->q3

psi_H = H_4@psi_init # Hadamard on first qubut
psi_CNOT1 = CNOT1@psi_H
psi_CNOT2 = CNOT2@psi_CNOT1
psi_CNOT3 = CNOT3@psi_CNOT2

print("\nWavefuntion")
print(psi_CNOT3) # wavefuntion what shows |0000> and |1111> have 1/\sqrt(2) prob amplitudes
print()

probs = np.abs(psi_CNOT3)**2 # probabilities

outcomes = np.where(np.random.rand(10000) < 0.5, "|0000>", "|1111>")

unique, counts = np.unique(outcomes, return_counts=True)
results = dict(zip(unique, counts))
print(results)
print()
```

```
plt.bar(results.keys(), results.values())
plt.xlabel('basis states')
plt.ylabel('counts')
plt.show()


Wavefuntion
[0.70710678+0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.         +0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.         +0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.         +0.j 0.         +0.j 0.         +0.j 0.70710678+0.j]

{'|0000>': 5021, '|1111>': 4979}
```



2 comparison

- Table of Contents

```
print(from_cirq_results_dict_4)
print(results)

{'|0000⟩': 4970, '|1111⟩': 5030}
{'|0000>': 5021, '|1111>': 4979}
```

Like in Question 1, the results of the CIRQ simulation and the Numpy simulation are comparable. However, the difference in complexity for the Numpy simulation is noticable, with the need to create a more convoluted CNOT gate, the need to take the tensor product of the Hadamard gate with the Identity matrix iteratively several times, as well as the need to apply the different CNOT gates several times. From this we can see how the the difficulty in building such a simulation will become increasinly complex as the number of qubits increase, even running into substantial complexity on the level of 5>¿ qubit systems.