Part 1

How did you get up to speed in basic linear algebra (e.g., Euler formula, linear transformations or operators, eigenvectors and eigenvalues) including which resources you consulted in the process?

When it came to getting up to speed with of the linear algebra we use in this course I was able to consult the great reference videos found in [1]. Although I already have linear algrebra experience from my engineering and math classes, this resource provided me with a useful reminder on some specific linear algrebra topics. In particular: null space, abstract vector spaces, and eigen-values. To get back up to speed with these topics I watched in videos linked in [1] and took some light notes.

How did you get started documenting linear algebra formulas (e.g., Euler formula) using LaTeX Markdown in Jupyter Notebooks?

Until this course, I have never used mark-down on Jupyter Notebooks. My only exposure to LaTex languages has been through overleaf when helping to write journal papers. And when I was writing in overleaf I have never written any linear algebra formulas. Thus, to get up to speed with writing linear algebra in LaTex Markdown I wrote at small cheat sheet of. As a reference I used [2] and [3] to help guide me through this process. My example cheat sheet can be found below.

Eq: two by two matrix

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Eq: L1-Norm of a vector size n

$$\sqrt{x_1^2 + .. + x_n^2}$$

Eq: inner product

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = a * x_1 * y_1 + b * x_2 * y_2 + c * x_1 * y_1 + d * x_2 * y_2$$

Eq: Matrix multiplication

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} y_1 & y_2 \\ y_2 & y_3 \end{bmatrix}$$

With resultant matrix C have the following values

$$C_{ij} = \sum_{k=0}^{n} a_{ik} * b_{kj}$$

How did you get started and experienced running your first quantum circuits using IBM Qiskit and Microsoft QDK Q# Jupyter Notebook platforms?

My first introduction to QisKit and Q# was through this course, SENG 480D: Quantum Computuing. I was introduced to both languages in during the lectures and have been able to gain expericne with them through course assignments. I have gained experiecne in writing algorithms and quantum circuits in both QisKit and Q#. Some examples of my code are shown below

Super Dense encoding algorthim adpated from https://qiskit.org/textbook/ch-algorithms/superdense-coding.html

In [3]:

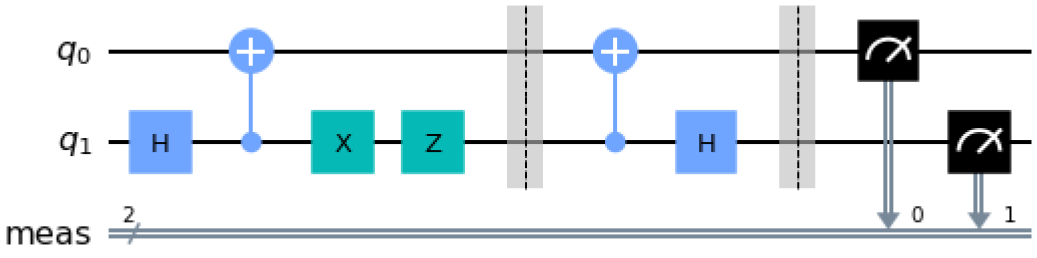
```
import qiskit
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit.visualization import plot_histogram
from qiskit.quantum_info import Statevector
from qiskit import Aer
from qiskit import transpile
import numpy as np

def create_bell_pair():
    q = QuantumCircuit(2)
    q.h(1)
    q.cx(1, 0)
    return q

def encode_message(q, qubit, msg):
    if len(msg) != 2 or not set(msg).issubset({"0","1"}): # error checking
        raise ValueError(f"message '{msg}' is invalid")

    if msg[1] == "1":
        q.x(qubit)

    if msg[0] == "1":
        q.z(qubit)

    return q

def decode_message(q):
    q.cx(1, 0)
    q.h(1)

    return q
```

In [4]:

```
# Charlie creates the entangled pair between Alice and Bob
q = create_bell_pair()


message = '11'
```

```
q = encode_message(q, 1, message)
q.barrier()

# Bob applies the recovery protocol:
q = decode_message(q)

# Finally, Bob measures his qubits to read Alice's message
q.measure_all()

# Draw our output
q.draw('mpl')
```

Out[4]:

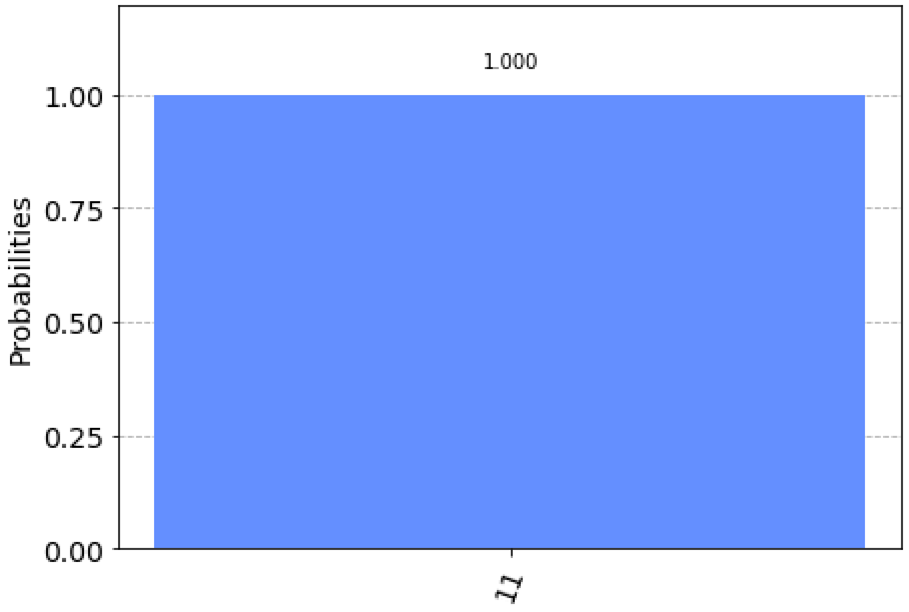


In [5]:

```
sim = Aer.get_backend('aer_simulator')
complied_circuit = transpile(q,sim)
result = sim.run(complied_circuit).result()
counts = result.get_counts(q)
print(counts)
plot_histogram(counts)
```

```
{'11': 1024}
```

Out[5]:



Quantum teleportation algorthim

Algorithm uses bell state $|\phi^{+}\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$ instead of the traditional $|\phi^{+}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$

In [2]:

```
open Microsoft.Quantum.Measurement;
open Microsoft.Quantum.Diagnostics;

operation Alice (q1 : Qubit, qInfo : Qubit) : (Bool, Bool) {

    mutable bool_qInfo = true;
    mutable bool_q1 = true;

    CNOT(qInfo,q1);

    H(qInfo);

    let result_qInfo = M(qInfo);
    let result_q1 = M(q1);

    if result_qInfo == Zero {
       set bool_qInfo = false;
    }
    if result_q1 == Zero {
       set bool_q1 = false;
    }

    return (bool_qInfo, bool_q1);
}

operation Eve_bell_state (q1 : Qubit, q2 : Qubit) : Unit {
    H(q1);
    CNOT(q1, q2);
    X(q1);

   // return (q1,q2);
}

operation Bob_bell_state (q2 : Qubit, (b1 : Bool, b2 : Bool)) : Unit {


    //Consider info qubit from eve

    //** Three states we have to consider |00>, |01>, |10>, |11>**//


    if b1 == false and b2 == false
    {
        X(q2);
    }

    if b1 == true and b2 == false
    {
        Z(q2);
        X(q2);
    }

    if b1 == true and b2 == true
    {
       Z(q2);
    }
    DumpMachine();
}

operation Teleportation2 (qInfo: Qubit) : Unit {
    use q = Qubit[2];

    Eve_bell_state(q[0],q[1]);
    let boolean_result = Alice(q[0],qInfo);
    Bob_bell_state(q[1],boolean_result);

    //DumpMachine();

    ResetAll(q);
}
```

Out[2]:

- Alice
- Eve_bell_state
- Bob_bell_state
- Teleportation2

In [4]:

```
// Your solution here
open Microsoft.Quantum.Measurement;
open Microsoft.Quantum.Diagnostics;

operation new_bell_state() : Unit {
    use q = Qubit();


    DumpMachine();

    Teleportation2(q);

    Reset(q);


}
```

Out[4]:

- new_bell_state

In [ ]:

```
%simulate new_bell_state
```

Part 2

Grovers Search algorithm

Grover's algorithm demonstrates the capability of speed searching from a dataset over its classical counterparts. This algorithm can speed up an unstructured search problem quadratically to the order of $O(\sqrt{N})$. It uses a trick or subroutine for identifying the target number by building an oracle. Whereas, the subroutine to obtain the quadratic runtime improvement is known as amplitude amplification.

This is a very powerful algorthim that is used often quantum computing. Any unstructured search uses a variant of this algorthim.

In [1]:

```
# Imports
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector
from qiskit.visualization import plot_histogram
#from amp_utils import plot_amplitude

# Initialization of qubits

qubits = 3
qc = QuantumCircuit(qubits + 1)   # One additional qubit for ancilary

# WRITE YOUR CODE HERE

for q in range(qubits + 1):
    qc.h(q)

qc.draw()

from qiskit.circuit.library.standard_gates import ZGate

c3z_gate = ZGate().control(2)
qc.x(0)
qc.x(2)
qc.append(c3z_gate,[0,1,2])
qc.x(0)
qc.x(2)
```

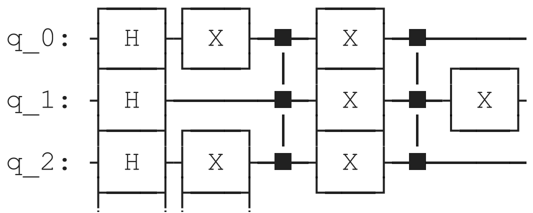
Out[1]:

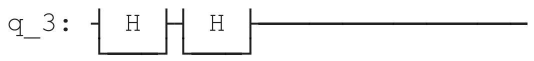
```
<qiskit.circuit.instructionset.InstructionSet at 0x24d91283140>
```

In [2]:

```
# Oracale 2: Make oracle to find only the second number

# WRITE YOUR CODE HERE
qc.x(1)
qc.append(c3z_gate,[0,1,2])
qc.x(1)



# WRITE YOUR CODE HERE

qc.h(3)   # Hadamard on ancilary qubit to store the phase value

qc.draw()
```

Out[2]:

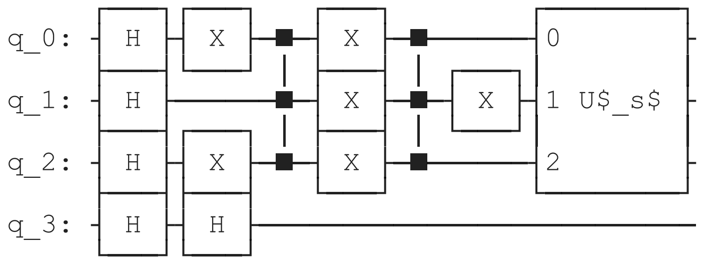




In [3]:

```
# Diffuser for Amplitude Amplification

#amplitude diffuser adapted from https://qiskit.org/textbook/ch-algorithms/grover.html
# WRITE YOUR CODE HERE
def diffuser(nqubits):
    qc = QuantumCircuit(nqubits)
    # Apply transformation |s> -> |00..0> (H-gates)
    for qubit in range(nqubits):
        qc.h(qubit)
    # Apply transformation |00..0> -> |11..1> (X-gates)
    for qubit in range(nqubits):
        qc.x(qubit)
    # Do multi-controlled-Z gate
    qc.h(nqubits-1)
    qc.mct(list(range(nqubits-1)), nqubits-1)  # multi-controlled-toffoli
    qc.h(nqubits-1)
    # Apply transformation |11..1> -> |00..0>
    for qubit in range(nqubits):
        qc.x(qubit)
    # Apply transformation |00..0> -> |s>
    for qubit in range(nqubits):
        qc.h(qubit)
    # We will return the diffuser as a gate
    U_s = qc.to_gate()
    U_s.name = "U$_s$"
    return U_s

diffuser = diffuser(3)
qc.draw()

qc.append(diffuser,[0,1,2])

qc.draw()


# WRITE YOUR CODE HERE
```

Out[3]:



Part 3

Quantum Phase Estimation

Quantum phase estimation is a very important algorthim in eigenvalue computation. It puts a number of quantum computing theories into practices. Primarily the concpet of that quantum circuits are invertiable because they are unitary matrix operations.

In [1]:
```
operation initialize (q: Qubit []) : Unit {
    // WRITE YOUR CODE HERE\
    for (bit in q) {
        H(bit);
    }

}
```

```
C:\snippet_.qs(3,9): warning QS3306: Deprecated syntax. Parentheses here are no longer r
equired and will not be supported in the future.
```

Out[1]:

- initialize

In [2]:
```
operation Rotation (q : Qubit, k : Int) : Unit is Adj+Ctl {
    R1Frac(2, k, q);
}

operation add_rotations(q: Qubit[], num_qubits: Int): Unit is Adj+Ctl {
    // WRITE YOUR CODE HERE
    // Use R1Frac for implementing controlled rotations.

    for ind in 1 .. Length(q) - 1 {
        Controlled Rotation([q[ind]], (q[0] , ind + 1));
    }

}

operation swap(q: Qubit[], num_qubits: Int): Unit is Adj+Ctl {
    for ind in 0 .. Length(q) / 2 - 1 {
        SWAP(q[ind], q[Length(q) - 1 - ind]);
    }
}
```

Out[2]:

- Rotation
- add_rotations
- swap

In [3]:
```
operation QFT(q: Qubit[], num_qubits: Int): Unit is Adj + Ctl {
    // WRITE YOUR CODE HERE
    // Use R1Frac for implementing controlled rotations.
    for ind in 0 .. Length(q) - 1 {
            add_rotations(q[ind ...],num_qubits);
    }
}
```

Out[3]:

- QFT

In [4]:
```
// Call all of the above functions in a single operation to build the QPE circuit.

operation QPE (): Unit {
    use q = Qubit[4];
    let num_qubits = Length(q);

    // Initialize the circuit
    initialize(q);

    // Add rotations wrt T gate.
    add_rotations(q, num_qubits);

    // Implement QFT-dagger (adjoint)
    Adjoint QFT(q, num_qubits);

    // R
    ResetAll(q);
}
```

Out[4]:

- QPE

In [5]:

```
// Call all of the above functions in a single operation to build the QPE circuit.

operation QPE (): Unit {
    use q = Qubit[4];
    let num_qubits = Length(q);

    // Initialize the circuit
    initialize(q);

    // Add rotations wrt T gate.
    add_rotations(q, num_qubits);

    // Implement QFT-dagger (adjoint)
    Adjoint QFT(q, num_qubits);

    // R
    ResetAll(q);
}
```

Out[5]:

- QPE

In [6]:

```
%trace QPE
```

In [7]:

```
%simulate QPE
```

Out[7]:

```
()
```

How would you motivate other students to join the journey into quantum computing given the motivational materials presented in class and found in the references? Your answers to this question will likely evolve during this course. Revisit regularly.

- The ability to tackle modeling problems that were thought to be very difficult
- That the field is growing at an extermely rapid pace and could be the next big computing revolution
- Thinking about computing from a quantum perspective rater than a classical one is interesting and fun
- That quantum computing has been a theory a growing slowly over the past 100 hundred years but is now really starting to take off

What are your personal insights, aha moments, and epiphanies you experienced in the first part of your quantum learning journey?

- That quantum teleportation and super encoding are very similar just the gates Alice used in quantum teleportation are the gates bob uses in super encoding and the gates Bob uses in teleportation are used by Alice in super encoding.

- The ability to reduce search time complexity to O(root(n)) for unstructured search spaces using groovers search algorithm

- The power of quantum computing to optimize eigenvalue based problems and do generally powerful min-max problems

- That classical and quantum computing will always be interlinked and the better the hybrid system the more oppurtunity we have to leverage quantum computing power

- The massive preformace improvment of quantum phase estimation

Bibliography

[1] L01-P1-Muller-Course-Overview-Slides UVIC: Accessed Jan 2022

[2] https://en.wikibooks.org/wiki/LaTeX/Mathematics: Accessed Jan 2022

[3] Muller Quantum Foundations and formulas juypter notebook: Accessed Jan 2022

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```