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 this process?

I got up to speed in basic linear algebra by doing simple practice questions like 2x2 matrices addition, subtraction, multiplication, and tensor product. I've used Google a lot, which includes Wikipedia and ChatGPT. However, ChatGPT is not always reliable, as quite often I found errors in its calculations. Nevertheless, It is faster than searching on Google when seeking for definitions and formulas.

What are your personal insights, aha moments, and epiphanies you experienced in the first part of this course?

My personal insights are watch the lecture videos and work on the lecture slides problems and go through them line by line until i fully understand them. My aha moments were very trivial but at the time I just couldn't figure out. For example, I thought

$$|0\rangle + |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Ket zero plus Ket one can be writen as 2x2 matrix but this is wrong and it took me awhile to figure this out and when i did it really made my learning a whole lot easier. In actuality

$$|0\rangle + |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

How did you experience Generative AI as a learning tool

It really did improve my quality of life. It speeds up my search process and gives me a decent answer as long as the input is well rounded. However like what i said earlier I need to double check AI's return prompt because it is horrible at doing tensor product as an example.

All the Greek letters that will be used.

- **Alpha:** α
- **Beta:** β
- **Gamma:** γ
- **Delta:** δ

- **Theta:** θ
- **Pi:** π
- **Phi:** ϕ
- **Psi:** ψ

Quantum Gates and Their Effects

Pauli Gates

- **Pauli-X Gate:**

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

  - Effect on states:
    - |0⟩: $X|0\rangle = |1\rangle$
    - |1⟩: $X|1\rangle = |0\rangle$
    - |ψ⟩: $X|\psi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} b \\ a \end{bmatrix}$
- **Pauli-Y Gate:**

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

  - Effect on states:
    - |0⟩: $Y|0\rangle = i|1\rangle$
    - |1⟩: $Y|1\rangle = -i|0\rangle$
    - |ψ⟩: $Y|\psi\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} -ib \\ ia \end{bmatrix}$
- **Pauli-Z Gate:**

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

  - Effect on states:
    - |0⟩: $Z|0\rangle = |0\rangle$
    - |1⟩: $Z|1\rangle = -|1\rangle$
    - |ψ⟩: $Z|\psi\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a \\ -b \end{bmatrix}$

Hadamard Gate

- **Hadamard Gate:**

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

  - Effect on states:
    - |0⟩: $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
    - |1⟩: $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
    - |ψ⟩: $H|\psi\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} a+b \\ a-b \end{bmatrix}$

In Assignment 1, we were tasked with implementing a simple Quantum Key Distribution (QKD) protocol, specifically the BB84 key distribution protocol, using Qiskit.

BB84 Key Distribution Protocol in Qiskit

In Part 1 of this assignment, you learnt about a simple quantum key distribution protocol (called BB84, after C. H. Bennett and G. Brassard) for securely generating random bitstrings which can then be used to encrypt information to securely share it between parties.

In [3]:
```
from qiskit import QuantumCircuit, Aer
from qiskit.quantum_info import Statevector
from qiskit.primitives import Sampler
from qiskit.visualization import plot_histogram
import random
```

In [4]:
```
#Returns a random bit (0 or 1) with equal probability.
def RandomBit():
    random_int = random.choice([0,1])
    return random_int
```

This operation prepares each of the qubits in the input array in one of the $|0\rangle, |1\rangle, |+\rangle$ or $|-\rangle$. states randomly.

In [5]:
```
#Alice encodes a random bitstring
def alice_encoding(n):
    quantumCircuits = []
    randomBits = []
    randomBases = []
```

```
    for num in range(n):
        randomBits.append(RandomBit())
        randomBases.append(RandomBit())

    for num in range(n):
        qc = QuantumCircuit(1,1)
        if randomBits[num] != 0:
            qc.x(0)
        if randomBases[num] != 0:
            qc.h(0)
        quantumCircuits.append(qc)
    return quantumCircuits, randomBits, randomBases
```

Measures each qubit in a randomly chosen basis, X or Z, and returns the chosen bases and the measurement results.

In [6]:
```
#Bob measures the circuits in the encoded message in some random bases (Z or
def bob_decoding(encoded_message):
    measured_bases = []
    decoded_message = []

    for qubit in range(len(encoded_message)):
        random_basis = RandomBit()
        measured_bases.append(random_basis)
        qc = encoded_message[qubit]
        if random_basis != 0:
            qc.h(0)
        qc.measure(0,0)
        #run on sim
        sampler = Sampler()
        sampler.set_options(shots=1024)
        results = sampler.run(qc).result()
        decoded_message.append(results.quasi_dists[0])

    return measured_bases, decoded_message
```

This operation acts as the intermediary exchanging both classical and quantum information between parties. N controls the number of qubits transmitted, so our final key will be of length less than or equal to N.

In [7]:
```
#Putting it all together
def bb84_protocol(n):
    alice_qc, alice_randomBits, alice_randomBases = alice_encoding(n)
    bob_measured_bases, bob_decoded_message = bob_decoding(alice_qc)
    alice_key = []
    bob_key = []
    for index in range(n):
        if alice_randomBases[index] == bob_measured_bases[index]:
```

```
            alice_key.append(alice_randomBases[index])
        if len(bob_decoded_message[index]) == 1:
            bob_key.append(list(bob_decoded_message[index].keys())[0])
            bob_decoded_message[index] = list(bob_decoded_message[index].key

    return alice_randomBases, bob_measured_bases, alice_randomBits, bob_decc
```

Check your work

In [8]:
```
def formatOutput(basesS, basesR, bitS, bitR, key1, key2):
    keyCopy = key2.copy()
    same = "|"
    basisSentChar = "|"
    basisRecChar = "|"
    bitSent = "|"
    bitRec = "|"
    keyS = "|"
    keyR = "|"
    stateSent = "|"
    for i in range(len(basesR)):
        bitSent += ' 1 |' if bitS[i] == 1 else ' 0 |'
        bitRec += ' 1 |' if bitR[i] == 1 else ' 0 |'
        same += ' y |' if basesR[i] == basesS[i] else ' n |'
        keyS += ' {} |'.format(bitS[i]) if basesR[i] == basesS[i] else "   |
        keyR += ' {} |'.format(key2.pop(0)) if basesR[i] == basesS[i] else "
        basisRecChar += ' Z |' if basesR[i] == 0 else ' X |'
        if basesS[i] == 0:
            stateSent += "|0>|" if bitS[i] == 0 else "|1>|"
            basisSentChar += " Z |"
        else:
            stateSent += "|+>|" if bitS[i] == 0 else "|->|"
            basisSentChar += " X |"

    print("Let's compare this to the selected bases, and the transmitted qub
    print("Alice's bases were:       {}".format(basisSentChar))
    print("Bob's bases were:         {}".format(basisRecChar))
    print("Both bases match (yes/no):{}".format(same))
    print("Alice's encoded bit:      {}".format(bitSent))
    print("The states sent were:     {}".format(stateSent))
    print("Bob measured:             {}".format(bitRec))
    print("Notice how the key is formed by the bits where bases are equal")
    print("Bob's key:                {}".format(keyR))
    print("Alice's key:              {}".format(keyS))
    print("The key that was generated was {}\n".format(keyCopy))
(basesS, basesR, bitS, bitR, alice_key, bob_key) = bb84_protocol(10)
formatOutput(basesS, basesR, bitS, bitR, alice_key, bob_key)
```

```
Let's compare this to the selected bases, and the transmitted qubit states
Alice's bases were:       | X | X | Z | Z | Z | X | Z | Z | X | Z |
Bob's bases were:         | X | Z | Z | X | Z | Z | Z | X | X | Z |
Both bases match (yes/no):| y | n | y | n | y | n | y | n | y | y |
Alice's encoded bit:      | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
The states sent were:     ||->||+>||0>||0>||1>||+>||1>||0>||->||0>|
Bob measured:             | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
Notice how the key is formed by the bits where bases are equal
Bob's key:                | 1 |   | 0 |   | 1 |   | 1 |   | 1 | 0 |
Alice's key:              | 1 |   | 0 |   | 1 |   | 1 |   | 1 | 0 |
The key that was generated was [1, 0, 1, 1, 1, 0]
```