Quantum _OFDM

Step 1: Create circuit for 2 user

Step2: Encode qubits

Step3: Apply QFT

Step4: Introduce Noisy Channel (Hence Kraus Operator , Non-unitary ,

hence no state vector only density matrices and measurement)

Step5: IQFT

Step6: Trans pile &Measurement

Step7: QBER Estimation Based on (Phase Noise= theta &

Q_OFDM.py 9+ | 2_user_QSMA.py 8 | QCDMA_Modified.py 8 | BER_Estimation.py 2

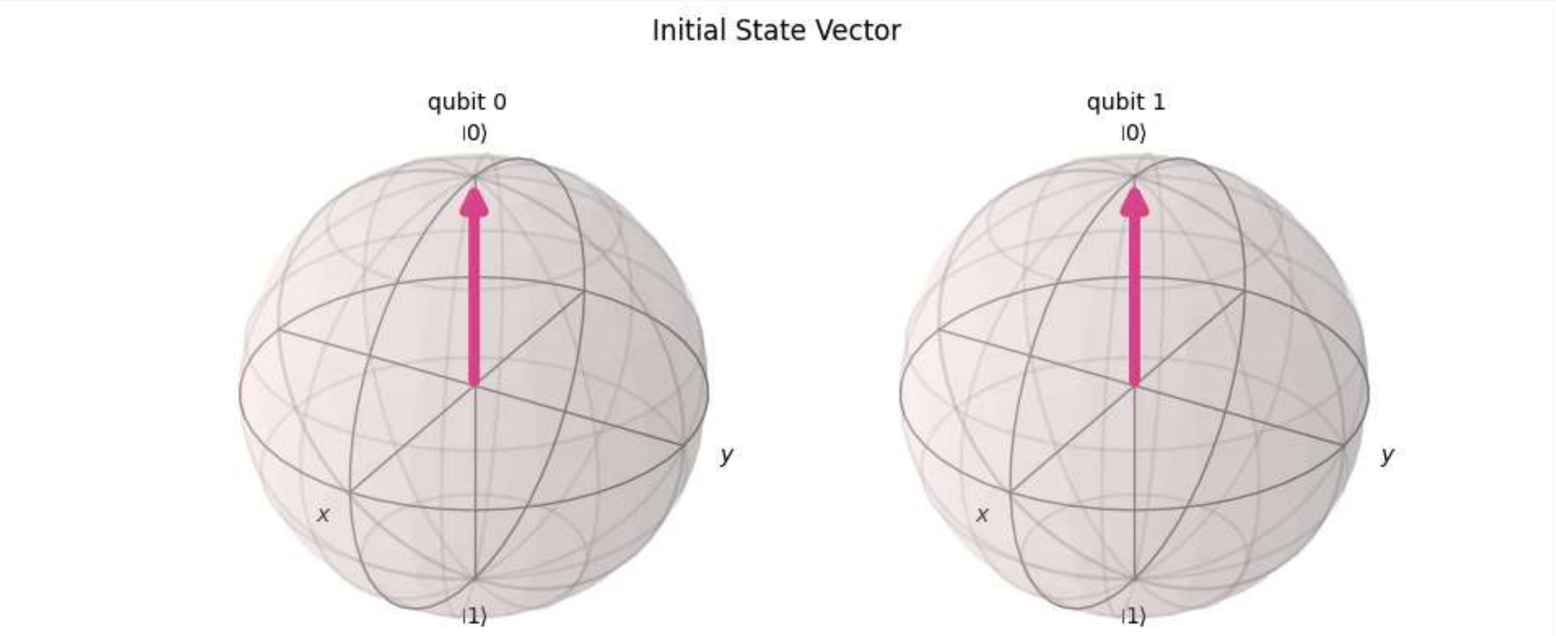
C: > Users > DELL > Soumen_2025_BSZ_8209 > QKD > Q_OFDM.py > QBER_Estimation

```python
def QBER_Estimation(counts,shots):
    )

    # Threshold plane
    ax.plot_surface(
        Theta, Gamma, QBER_plane,
        color='red',
        alpha=0.35
    )

    ax.set_xlabel("Phase noise θ (radians)")
    ax.set_ylabel("Amplitude damping γ")
    ax.set_zlabel("QBER")

    ax.set_title("QBER Surface with QKD Security Thresh

    fig.colorbar(surf, shrink=0.5, aspect=10, label="QE

    plt.show()
    return qber
    pass


def main():
    QBER=[]
    #for i in range(0,1):
    qc = QuantumCircuit(2, 2)
    state_vector= Statevector.from_instruction(qc)
    show_statevec(state_vector, "Initial State Vector")
    print(qc.draw('mpl'))
    plt.show()
    qc,state_vector_after_encoding=Encoding(qc)
    qc,state_vector_after_QFT=QFT_After_Encoding(qc)
    qc,state_vector_after_Noisy_Quantum_Channel=Noisy_C
    qc=Amplitude_Loss_Non_Unitary_Kraus_operator(qc)
    qc=IQFT_After_Noisy_Quantum_Channel(qc)
    counts=Measurement(qc)
    shots=4096
```

C:\Windows\system32\cmd.e

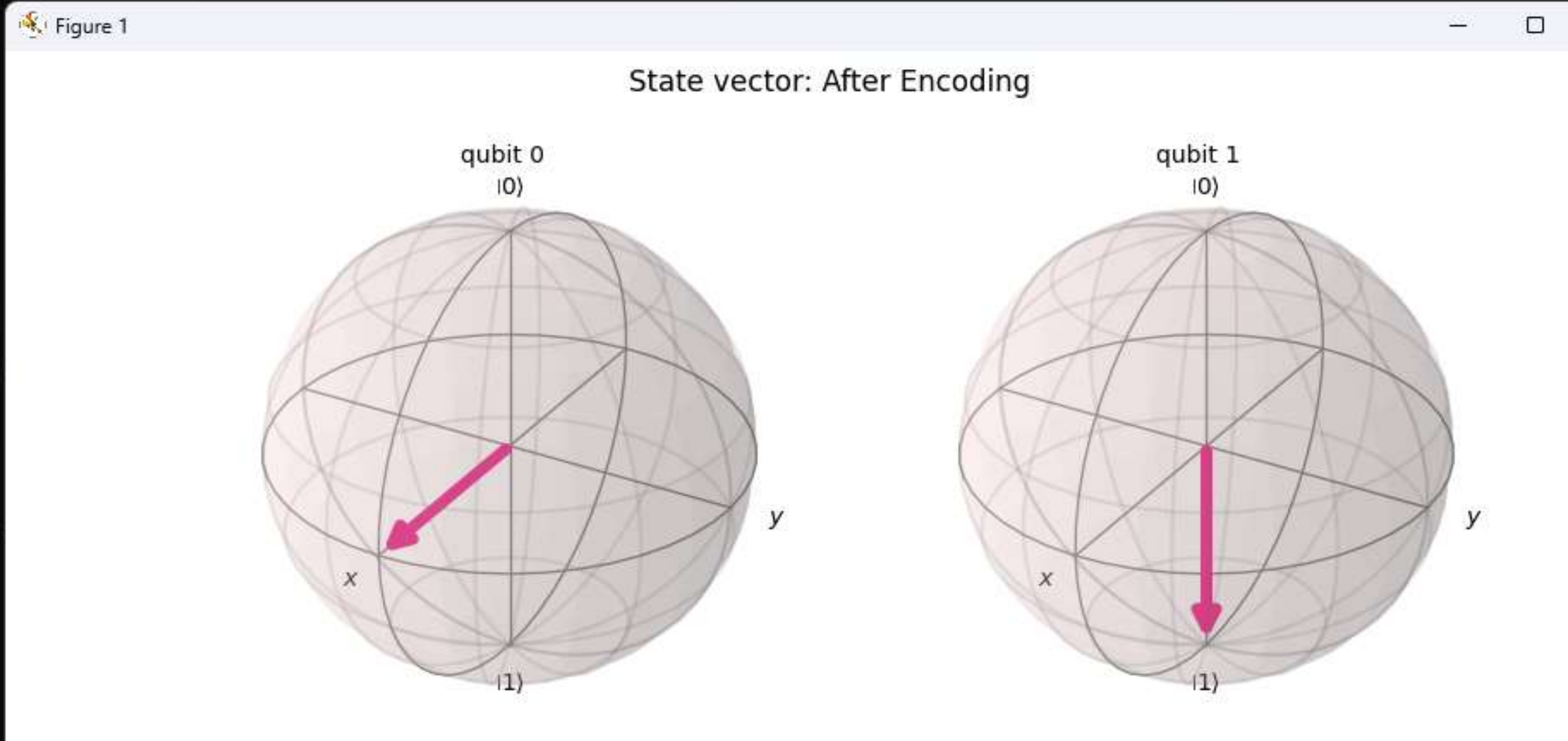
```
C:\Users\DELL\Soumen_2025_BSZ_8209\QKD>python Q_OFDM.py

==================================================================
Initial State Vector
==================================================================
|0000000000>  (1+0j)
```

Figure 1



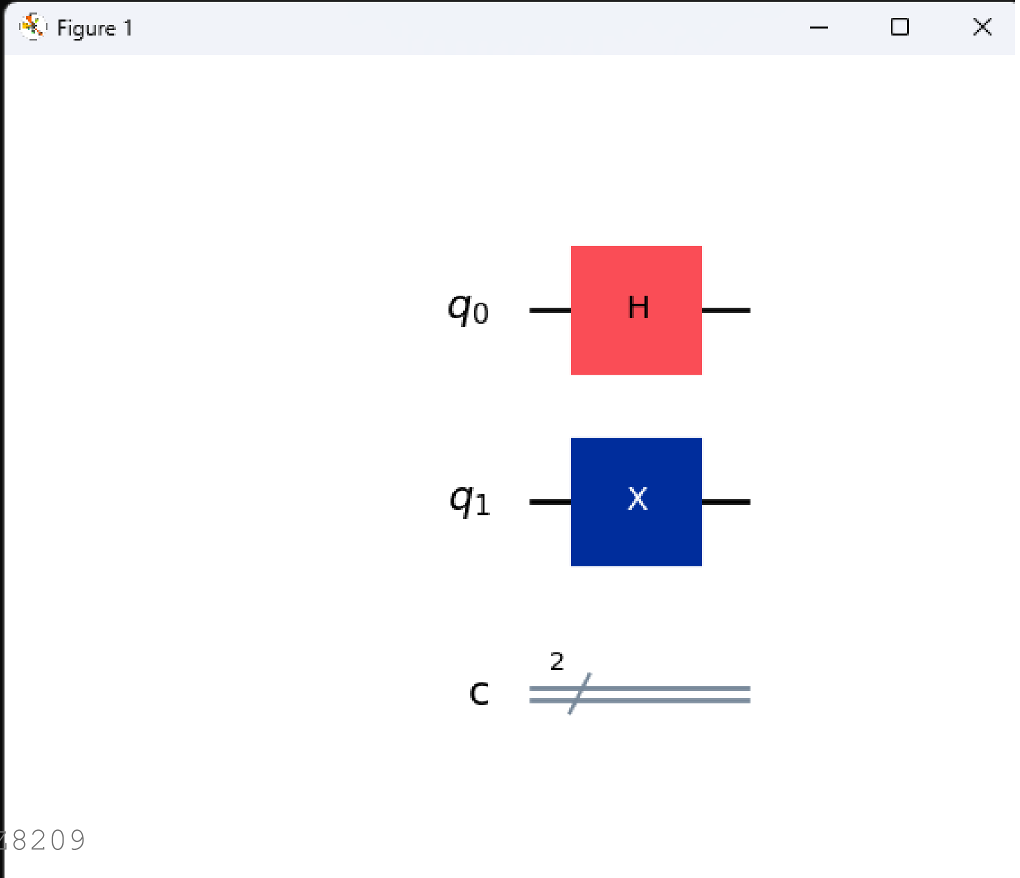
```python
# Threshold plane
ax.plot_surface(
    Theta, Gamma, QBER_plane,
    color='red',
    alpha=0.35
)

ax.set_xlabel("Phase noise θ (radians)")
ax.set_ylabel("Amplitude damping γ")
ax.set_zlabel("QBER")

ax.set_title("QBER Surface with QKD Security Thresh

fig.colorbar(surf, shrink=0.5, aspect=10, label="QE

plt.show()
return qber
pass


main():
QBER=[]
#for i in range(0,1):
qc = QuantumCircuit(2, 2)
state_vector= Statevector.from_instruction(qc)
show_statevec(state_vector, "Initial State Vector")
print(qc.draw('mpl'))
plt.show()
qc,state_vector_after_encoding=Encoding(qc)
qc,state_vector_after_QFT=QFT_After_Encoding(qc)
qc,state_vector_after_Noisy_Quantum_Channel=Noisy_(
qc=Amplitude_Loss_Non_Unitary_Kraus_operator(qc)
qc=IQFT_After_Noisy_Quantum_Channel(qc)
counts=Measurement(qc)
shots=4096
qber=QBER_Estimation(counts,shots)
QBER.append(qber)
print(QBER)

pass
```

```
C:\Users\DELL\Soumen_2025_BSZ_8209\QKD>python Q_OFDM.py

======================================================================
Initial State Vector
======================================================================
|0000000000>  (1+0j)
Figure(161.878x284.278)

======================================================================
State vector: After Encoding
======================================================================
|0000000010>  (0.7071067811865475+0j)
|0000000011>  (0.7071067811865475+0j)
```



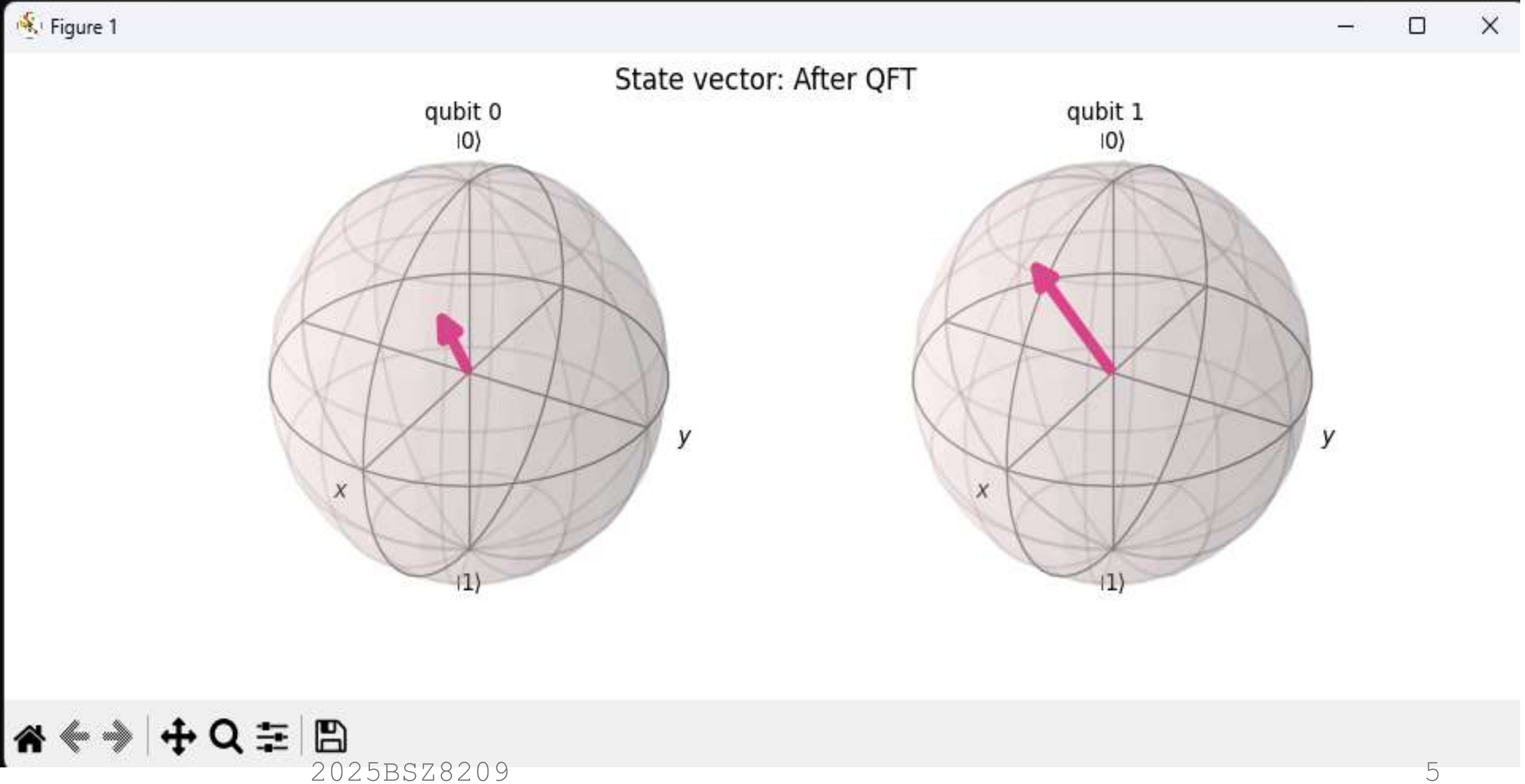
```python
    # Threshold plane
    ax.plot_surface(
        Theta, Gamma, QBER_plane,
        color='red',
        alpha=0.35
    )

    ax.set_xlabel("Phase noise θ (radians)")
    ax.set_ylabel("Amplitude damping γ")
    ax.set_zlabel("QBER")

    ax.set_title("QBER Surface with QKD Security Thresh

    fig.colorbar(surf, shrink=0.5, aspect=10, label="QE

    plt.show()
    return qber
    pass


f main():
    QBER=[]
    #for i in range(0,1):
    qc = QuantumCircuit(2, 2)
    state_vector= Statevector.from_instruction(qc)
    show_statevec(state_vector, "Initial State Vector")
    print(qc.draw('mpl'))
    plt.show()
    qc,state_vector_after_encoding=Encoding(qc)
    qc,state_vector_after_QFT=QFT_After_Encoding(qc)
    qc,state_vector_after_Noisy_Quantum_Channel=Noisy_Q
    qc=Amplitude_Loss_Non_Unitary_Kraus_operator(qc)
    qc=IQFT_After_Noisy_Quantum_Channel(qc)
    counts=Measurement(qc)
    shots=4096
    qber=QBER_Estimation(counts,shots)
    QBER.append(qber)
    print(QBER)

    pass
```

```
C:\Users\DELL\Soumen_2025_BSZ_8209\QKD>python Q_OFDM.py

======================================================================
Initial State Vector
======================================================================
|0000000000>  (1+0j)
Figure(161.878x284.278)

======================================================================
State vector: After Encoding
======================================================================
|0000000010>  (0.7071067811865475+0j)
|0000000011>  (0.7071067811865475+0j)
Figure(203.683x284.278)
```



```
ax.set_xlabel("Phase noise θ (radians)")
ax.set_ylabel("Amplitude damping γ")
ax.set_zlabel("QBER")

ax.set_title("QBER Surface with QKD Security Thresh

fig.colorbar(surf, shrink=0.5, aspect=10, label="QE

plt.show()
return qber
pass


main():
QBER=[]
#for i in range(0,1):
qc = QuantumCircuit(2, 2)
state_vector= Statevector.from_instruction(qc)
show_statevec(state_vector, "Initial State Vector")
print(qc.draw('mpl'))
plt.show()
qc,state_vector_after_encoding=Encoding(qc)
qc,state_vector_after_QFT=QFT_After_Encoding(qc)
qc,state_vector_after_Noisy_Quantum_Channel=Noisy_Q
qc=Amplitude_Loss_Non_Unitary_Kraus_operator(qc)
qc=IQFT_After_Noisy_Quantum_Channel(qc)
counts=Measurement(qc)
shots=4096
qber=QBER_Estimation(counts,shots)
QBER.append(qber)
print(QBER)

pass

()
```

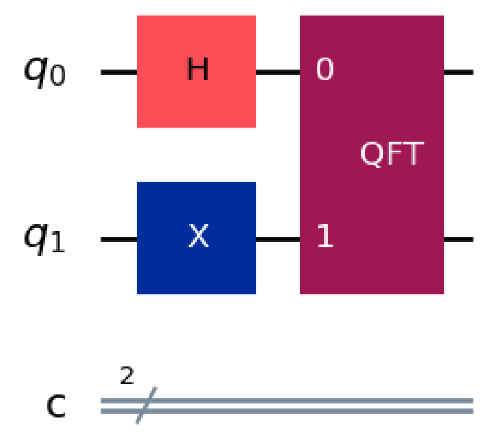
```
============================================================================
State vector: After Encoding
============================================================================
|0000000010>  (0.7071067811865475+0j)
|0000000011>  (0.7071067811865475+0j)
Figure(203.683x284.278)

============================================================================
State vector: After QFT
============================================================================
|0000000000>  (0.7071067811865474+0j)
|0000000001>  (-0.3535533905932737-0.3535533905932737j)
|0000000011>  (-0.3535533905932737+0.3535533905932737j)
```

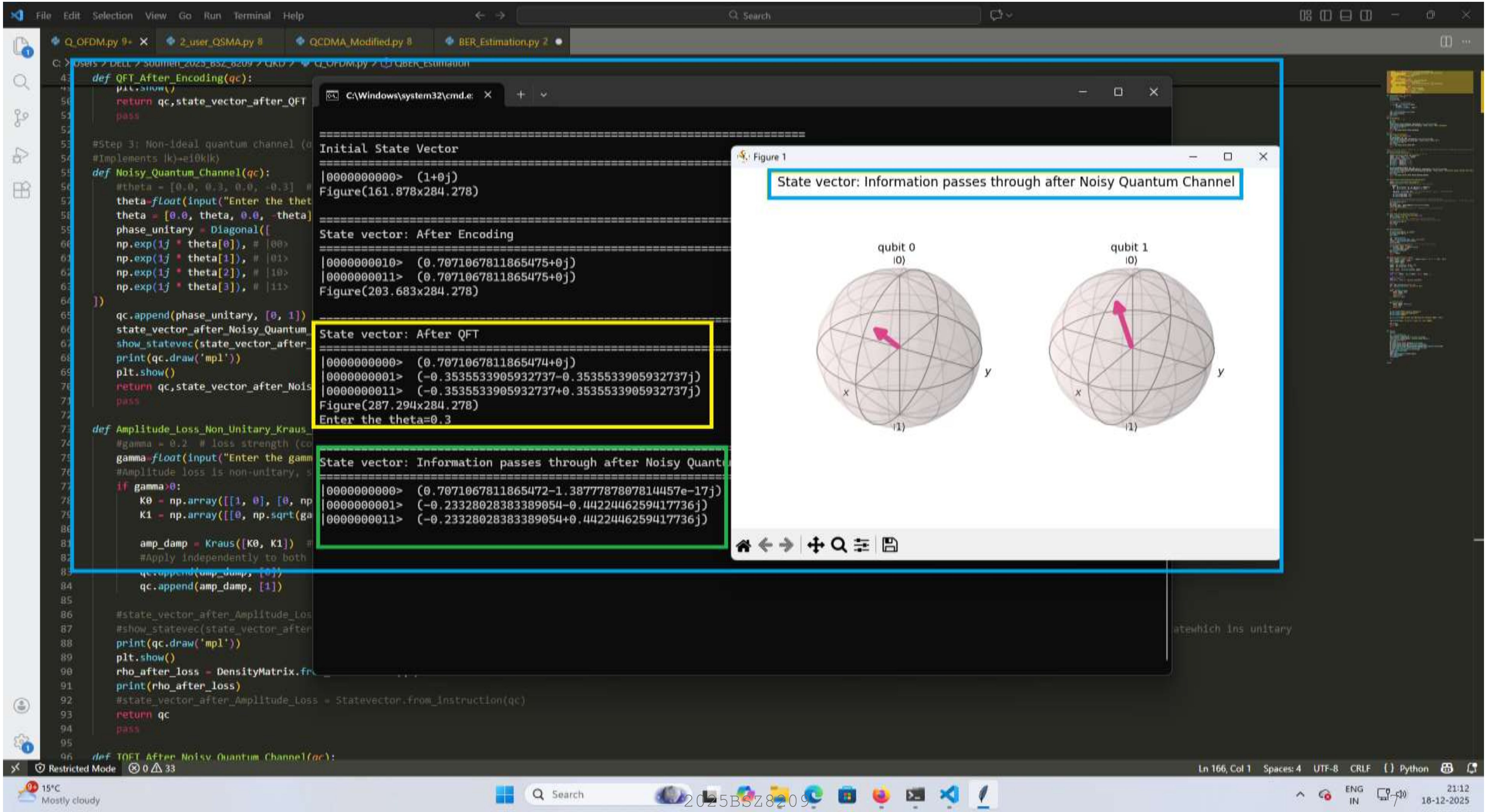


```
+ ✕   2_user_QSMA.py 8   QCDMA_Modified.py 8   BER_Estimation.py 2 ●
L > Soumen_2025_BSZ_8209 > QKD > Q_OFDM.py > QBER_Estimation
ER_Estimation(counts,shots):

 Threshold plane
.plot_surface(
  Theta, Gamma, QBER_plane,
  color='red',
  alpha=0.35


.set_xlabel("Phase noise θ (radians)")
.set_ylabel("Amplitude damping γ")
.set_zlabel("QBER")

.set_title("QBER Surface with QKD Security Thresh

g.colorbar(surf, shrink=0.5, aspect=10, label="QE

t.show()
turn qber
ss


in():
ER=[]
or i in range(0,1):
 = QuantumCircuit(2, 2)
ate_vector= Statevector.from_instruction(qc)
ow_statevec(state_vector, "Initial State Vector")
int(qc.draw('mpl'))
t.show()
,state_vector_after_encoding=Encoding(qc)
,state_vector_after_QFT=QFT_After_Encoding(qc)
,state_vector_after_Noisy_Quantum_Channel=Noisy_Q
=Amplitude_Loss_Non_Unitary_Kraus_operator(qc)
=IQFT_After_Noisy_Quantum_Channel(qc)
unts=Measurement(qc)
ots=4096
```

```
C:\Windows\system32\cmd.e:

C:\Users\DELL\Soumen_2025_BSZ_8209\QKD>python Q_OFDM.py

========================================================
Initial State Vector
========================================================
|0000000000>  (1+0j)
Figure(161.878x284.278)

========================================================
State vector: After Encoding
========================================================
|0000000010>  (0.7071067811865475+0j)
|0000000011>  (0.7071067811865475+0j)
Figure(203.683x284.278)

========================================================
State vector: After QFT
========================================================
|0000000000>  (0.7071067811865474+0j)
|0000000001>  (-0.35355339059327737-0.35355339059327737j)
|0000000011>  (-0.35355339059327737+0.35355339059327737j)
Figure(287.294x284.278)
```

Figure 1



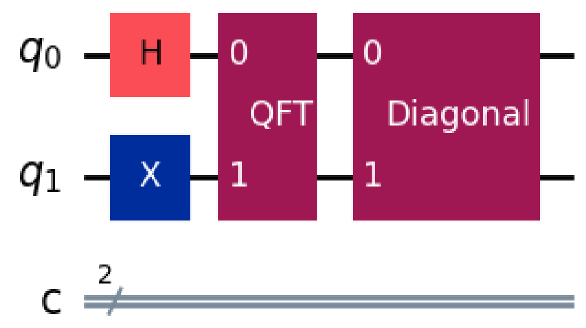

C:\Windows\system32\cmd.e
Initial State Vector
|0000000000> (1+0j)
Figure(161.878x284.278)
State vector: After Encoding
|0000000010> (0.7071067811865475+0j)
|0000000011> (0.7071067811865475+0j)
Figure(203.683x284.278)
State vector: After QFT
|0000000000> (0.7071067811865474+0j)
|0000000001> (-0.35355339059327373-0.3535533905932737j)
|0000000011> (-0.35355339059327373+0.3535533905932737j)
Figure(287.294x284.278)
Enter the theta=0.3
State vector: Information passes through after Noisy Quant
|0000000000> (0.7071067811865472-1.3877787807814457e-17j)
|0000000001> (-0.23328028383389054-0.44224462594177736j)
|0000000011> (-0.23328028383389054+0.44224462594177736j)
Figure 1
State vector: Information passes through after Noisy Quantum Channel
qubit 0
qubit 1
2025BSZ8209

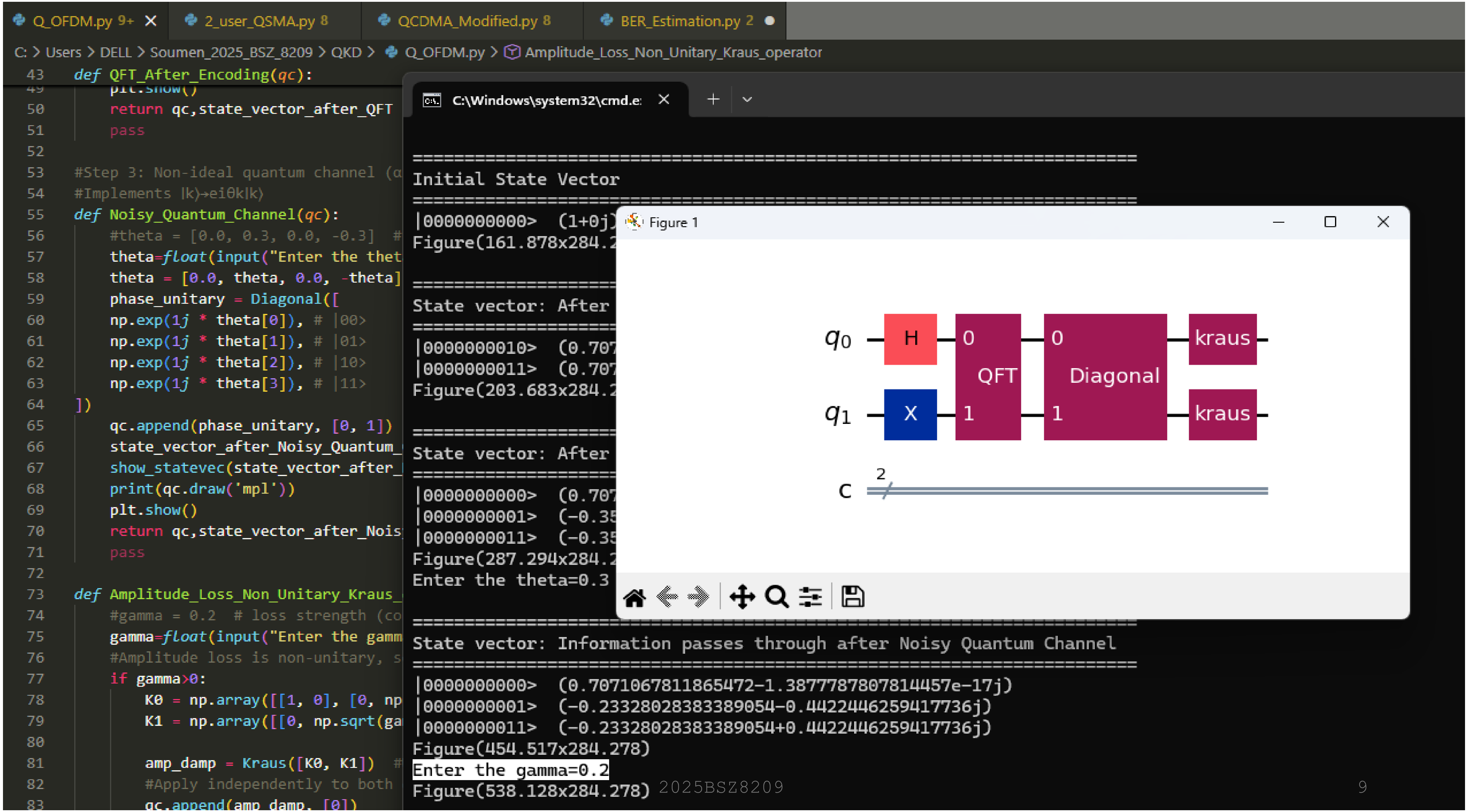
```
plt.show()
return qc,state_vector_after_QFT
pass

p 3: Non-ideal quantum channel (α
lements |k⟩→eiθk|k⟩
Noisy_Quantum_Channel(qc):
#theta = [0.0, 0.3, 0.0, -0.3]  #
theta=float(input("Enter the thet
theta = [0.0, theta, 0.0, -theta]
phase_unitary = Diagonal([
np.exp(1j * theta[0]), # |00>
np.exp(1j * theta[1]), # |01>
np.exp(1j * theta[2]), # |10>
np.exp(1j * theta[3]), # |11>

qc.append(phase_unitary, [0, 1])
state_vector_after_Noisy_Quantum_
show_statevec(state_vector_after_
print(qc.draw('mpl'))
plt.show()
return qc,state_vector_after_Nois
pass

Amplitude_Loss_Non_Unitary_Kraus_
#gamma = 0.2  # loss strength (co
gamma=float(input("Enter the gamm
#Amplitude loss is non-unitary, s
if gamma>0:
    K0 = np.array([[1, 0], [0, np
    K1 = np.array([[0, np.sqrt(ga

    amp_damp = Kraus([K0, K1])  #
    #Apply independently to both
    qc.append(amp_damp, [0])
```

C:\Windows\system32\cmd.e

```
============================================================================
Initial State Vector
============================================================================
|0000000000>  (1+0j)
Figure(161.878x284.278)

============================================================================
State vector: After Encoding
============================================================================
|0000000010>  (0.7071067811865475+0j)
|0000000011>  (0.7071067811865475+0j)
Figure(203.683x284.278)

============================================================================
State vector: After QFT
============================================================================
|0000000000>  (0.7071067811865474+0j)
|0000000001>  (-0.3535533905932737-0.3535533905932737j)
|0000000011>  (-0.3535533905932737+0.3535533905932737j)
Figure(287.294x284.278)
Enter the theta=0.3

============================================================================
State vector: Information passes through after Noisy Quantum Channel
============================================================================
|0000000000>  (0.7071067811865472-1.3877787807814457e-17j)
|0000000001>  (-0.2332802838338905 4-0.4422446259417736j)
|0000000011>  (-0.23328028383389054+0.4422446259417736j)
Figure(454.517x284.278)
```

Figure 1



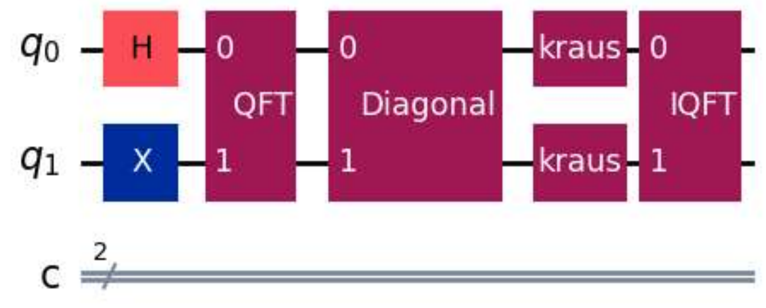

Q_OFDM.py 9+
2_user_QSMA.py 8
QCDMA_Modified.py 8
BER_Estimation.py 2
C: > Users > DELL > Soumen_2025_BSZ_8209 > QKD > Q_OFDM.py > Amplitude_Loss_Non_Unitary_Kraus_operator
def QFT_After_Encoding(qc):
plt.show()
return qc,state_vector_after_QFT
pass
#Step 3: Non-ideal quantum channel (α
#Implements |k⟩→eiθk|k⟩
def Noisy_Quantum_Channel(qc):
#theta = [0.0, 0.3, 0.0, -0.3] #
theta=float(input("Enter the thet
theta = [0.0, theta, 0.0, -theta]
phase_unitary = Diagonal([
np.exp(1j * theta[0]), # |00>
np.exp(1j * theta[1]), # |01>
np.exp(1j * theta[2]), # |10>
np.exp(1j * theta[3]), # |11>
])
qc.append(phase_unitary, [0, 1])
state_vector_after_Noisy_Quantum_
show_statevec(state_vector_after_
print(qc.draw('mpl'))
plt.show()
return qc,state_vector_after_Nois
pass
def Amplitude_Loss_Non_Unitary_Kraus_
#gamma = 0.2 # loss strength (co
gamma=float(input("Enter the gamm
#Amplitude loss is non-unitary, s
if gamma>0:
K0 = np.array([[1, 0], [0, np
K1 = np.array([[0, np.sqrt(ga
amp_damp = Kraus([K0, K1]) #
#Apply independently to both
qc.append(amp_damp, [0])
C:\Windows\system32\cmd.e
Initial State Vector
|0000000000> (1+0j)
Figure(161.878x284.2
State vector: After
|0000000010> (0.707
|0000000011> (0.707
Figure(203.683x284.2
State vector: After
|0000000000> (0.707
|0000000001> (-0.35
|0000000011> (-0.35
Figure(287.294x284.2
Enter the theta=0.3
State vector: Information passes through after Noisy Quantum Channel
|0000000000> (0.7071067811865472-1.3877787807814457e-17j)
|0000000001> (-0.23328028383389054-0.44224462594177736j)
|0000000011> (-0.23328028383389054+0.44224462594177736j)
Figure(454.517x284.278)
Enter the gamma=0.2
Figure(538.128x284.278)
Figure 1
q_0
q_1
H
X
0
QFT
1
0
Diagonal
1
kraus
kraus
c
2

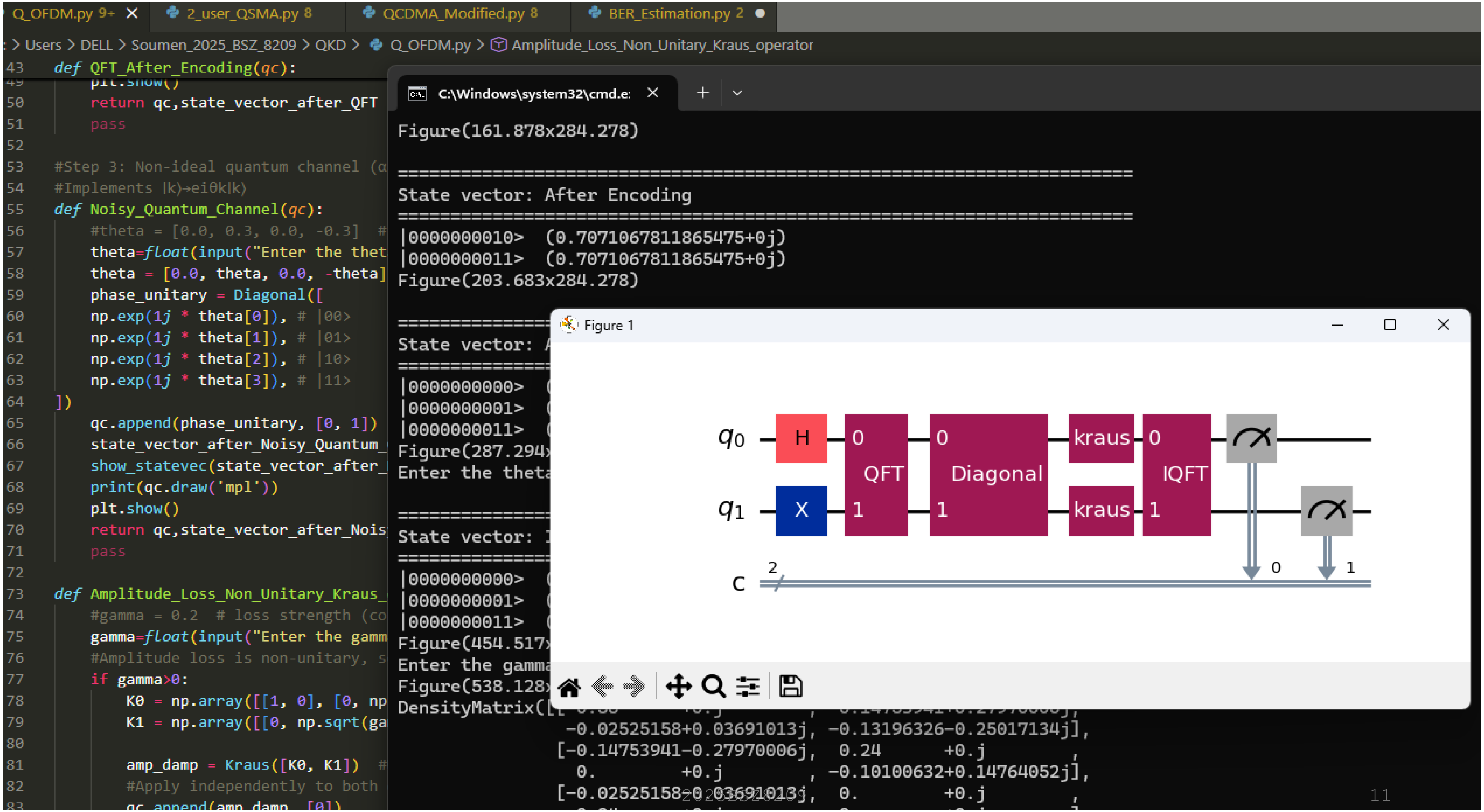
```python
        return qc,state_vector_after_QFT
        pass

    #Step 3: Non-ideal quantum channel (α
    #Implements |k⟩→eiθk|k⟩
    def Noisy_Quantum_Channel(qc):
        #theta = [0.0, 0.3, 0.0, -0.3]  #
        theta=float(input("Enter the thet
        theta = [0.0, theta, 0.0, -theta]
        phase_unitary = Diagonal([
        np.exp(1j * theta[0]), # |00>
        np.exp(1j * theta[1]), # |01>
        np.exp(1j * theta[2]), # |10>
        np.exp(1j * theta[3]), # |11>
    ])
        qc.append(phase_unitary, [0, 1])
        state_vector_after_Noisy_Quantum_
        show_statevec(state_vector_after_
        print(qc.draw('mpl'))
        plt.show()
        return qc,state_vector_after_Nois
        pass

    def Amplitude_Loss_Non_Unitary_Kraus_
        #gamma = 0.2  # loss strength (co
        gamma=float(input("Enter the gamm
        #Amplitude loss is non-unitary, s
        if gamma>0:
            K0 = np.array([[1, 0], [0, np
            K1 = np.array([[0, np.sqrt(ga

            amp_damp = Kraus([K0, K1])  #
            #Apply independently to both
            qc.append(amp_damp, [0])
            qc.append(amp_damp, [1])

        #state_vector_after_Amplitude_Los
        #show_statevec(state_vector_after
        print(qc.draw('mpl'))
        plt.show()
```

C:\Windows\system32\cmd.e

```
|0000000000>  (1+0j)
Figure(161.878x284.278)

============================================================================
State vector: After Encod
============================
|0000000010>  (0.70710678
|0000000011>  (0.70710678
Figure(203.683x284.278)

============================
State vector: After QFT
============================
|0000000000>  (0.70710678
|0000000001>  (-0.3535533
|0000000011>  (-0.3535533
Figure(287.294x284.278)
Enter the theta=0.3

============================
State vector: Informatior
============================
|0000000000>  (0.7071067811865472-1.3877787807814457e-17j)
|0000000001>  (-0.23328028383389054-0.44224462594177336j)
|0000000011>  (-0.23328028383389054+0.44224462594177336j)
Figure(454.517x284.278)
Enter the gamma=0.2
Figure(538.128x284.278)
DensityMatrix([[ 0.56      +0.j        , -0.14753941+0.27970006j,
                 -0.02525158+0.03691013j, -0.13196326-0.25017134j],
                [-0.14753941-0.27970006j,  0.24      +0.j        ,
                  0.        +0.j        , -0.10100632+0.14764052j],
                [-0.02525158-0.03691013j,  0.        +0.j        ,
                  0.04      +0.j        ,  0.        +0.j        ],
                [-0.13196326+0.25017134j, -0.10100632-0.14764052j,
                  0.        +0.j        ,  0.16      +0.j        ]],
               dims=(2, 2))
Figure(621.739x284.278)
```

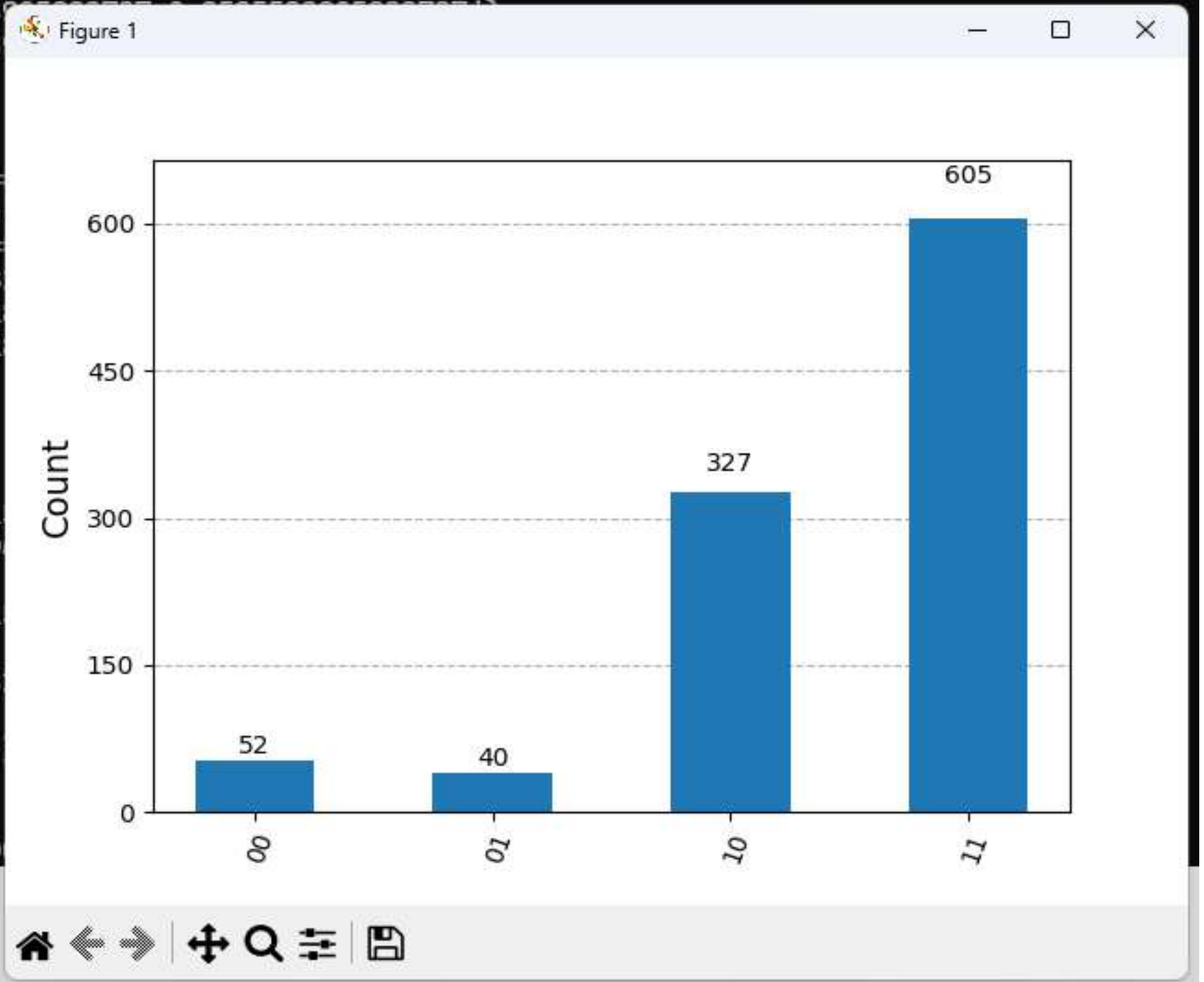
Figure 1




Q_OFDM.py 9+
2_user_QSMA.py 8
QCDMA_Modified.py 8
BER_Estimation.py 2
Users > DELL > Soumen_2025_BSZ_8209 > QKD > Q_OFDM.py > Amplitude_Loss_Non_Unitary_Kraus_operator
def QFT_After_Encoding(qc):
return qc,state_vector_after_QFT
pass
#Step 3: Non-ideal quantum channel (α
#Implements |k⟩→eiθk|k⟩
def Noisy_Quantum_Channel(qc):
#theta = [0.0, 0.3, 0.0, -0.3]
theta=float(input("Enter the thet
theta = [0.0, theta, 0.0, -theta]
phase_unitary = Diagonal([
np.exp(1j * theta[0]), # |00⟩
np.exp(1j * theta[1]), # |01⟩
np.exp(1j * theta[2]), # |10⟩
np.exp(1j * theta[3]), # |11⟩
])
qc.append(phase_unitary, [0, 1])
state_vector_after_Noisy_Quantum_
show_statevec(state_vector_after_
print(qc.draw('mpl'))
plt.show()
return qc,state_vector_after_Nois
pass
def Amplitude_Loss_Non_Unitary_Kraus_
#gamma = 0.2 # loss strength (co
gamma=float(input("Enter the gamm
#Amplitude loss is non-unitary, s
if gamma>0:
K0 = np.array([[1, 0], [0, np
K1 = np.array([[0, np.sqrt(ga
amp_damp = Kraus([K0, K1])
#Apply independently to both
C:\Windows\system32\cmd.e
Figure(161.878x284.278)
State vector: After Encoding
|0000000010> (0.7071067811865475+0j)
|0000000011> (0.7071067811865475+0j)
Figure(203.683x284.278)
State vector:
|0000000000>
|0000000001>
|0000000011>
Figure(287.294
Enter the theta
State vector:
|0000000000>
|0000000001>
|0000000011>
Figure(454.517
Enter the gamma
Figure(538.128
DensityMatrix([[
-0.02525158+0.03691013j, -0.13196326-0.25017134j],
[-0.14753941-0.27970006j, 0.24 +0.j ,
0. +0.j , -0.10100632+0.14764052j],
[-0.02525158-0.03691013j, 0. +0.j ,
Figure 1
q0
q1
C
H
X
QFT
Diagonal
kraus
kraus
IQFT
2
0
1

```python
    return qc,state_vector_after_QFT
    pass

#Step 3: Non-ideal quantum channel (α
#Implements |k⟩→eiθk|k⟩
def Noisy_Quantum_Channel(qc):
    #theta = [0.0, 0.3, 0.0, -0.3]  #
    theta=float(input("Enter the thet
    theta = [0.0, theta, 0.0, -theta]
    phase_unitary = Diagonal([
    np.exp(1j * theta[0]), # |00>
    np.exp(1j * theta[1]), # |01>
    np.exp(1j * theta[2]), # |10>
    np.exp(1j * theta[3]), # |11>
])
    qc.append(phase_unitary, [0, 1])
    state_vector_after_Noisy_Quantum_
    show_statevec(state_vector_after_
    print(qc.draw('mpl'))
    plt.show()
    return qc,state_vector_after_Nois
    pass

def Amplitude_Loss_Non_Unitary_Kraus_
    #gamma = 0.2  # loss strength (co
    gamma=float(input("Enter the gamm
    #Amplitude loss is non-unitary, s
    if gamma>0:
        K0 = np.array([[1, 0], [0, np
        K1 = np.array([[0, np.sqrt(ga

        amp_damp = Kraus([K0, K1])  #
        #Apply independently to both
        qc.append(amp_damp, [0])
        qc.append(amp_damp, [1])

    #state_vector_after_Amplitude_Los
    #show_statevec(state_vector_after
    print(qc.draw('mpl'))
    plt.show()
```

```
|0000000011>  (0.7071067811865475+0j)
Figure(203.683x284.278)

======================================================================
State vector: After QFT
======================================================================
|0000000000>  (0.7071067811865474+0j)
|0000000001>  (-0.3535533
|0000000011>  (-0.3535533
Figure(287.294x284.278)
Enter the theta=0.3

==========================
State vector: Information
==========================
|0000000000>  (0.70710678
|0000000001>  (-0.2332802
|0000000011>  (-0.2332802
Figure(454.517x284.278)
Enter the gamma=0.2
Figure(538.128x284.278)
DensityMatrix([[ 0.56
                -0.025251
               [-0.147539
                 0.
               [-0.025251
                 0.04
               [-0.131963
                 0.
              dims=(2, 2))
Figure(621.739x284.278)
Figure(788.961x284.278)
{'10': 327, '11': 605, '0
Measurement results:
10 327
11 605
00 52
01 40
```

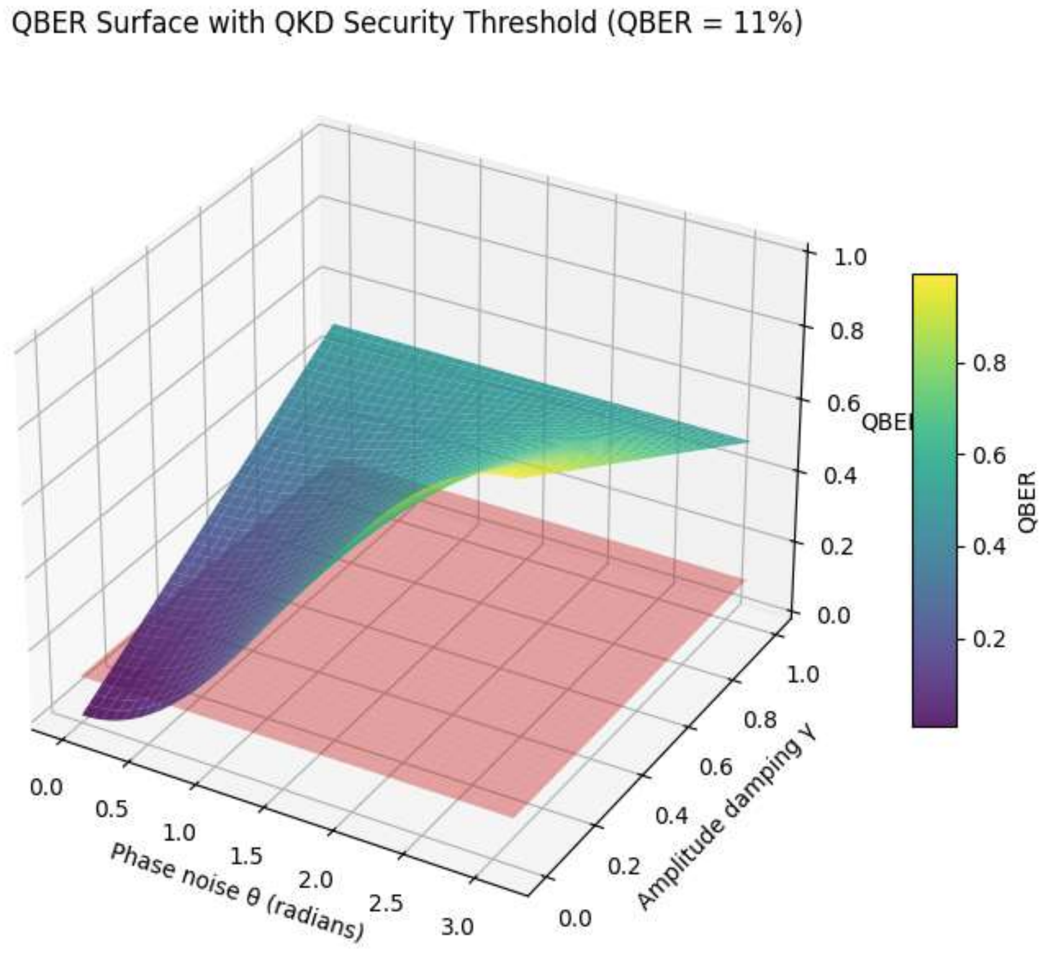


```python
    for outcome, count in counts.item
        print(outcome, count)
    plot_histogram(counts)
    plt.show()
    return counts
    pass

def QBER_Estimation(counts,shots):
    error_counts = sum(v for k, v in
    qber = error_counts / shots
    print("QBER=",qber)
    # Parameter ranges
    theta = np.linspace(0, np.pi, 120
    gamma = np.linspace(0, 1, 120)

    Theta, Gamma = np.meshgrid(theta,

    # QBER formula
    QBER = (1 - Gamma) * np.sin(Theta

    # Security threshold
    QBER_th = 0.11
    QBER_plane = QBER_th * np.ones_li

    # Plot
    fig = plt.figure(figsize=(10, 7))
    ax = fig.add_subplot(111, project

    # QBER surface
    surf = ax.plot_surface(
        Theta, Gamma, QBER,
        cmap='viridis',
        alpha=0.85,
        edgecolor='none'
    )

    # Threshold plane
    ax.plot_surface(
        Theta, Gamma, QBER_plane,
        color='red',
        alpha=0.35
```

C:\Windows\system32\cmd.e

```
Figure(203.683x284.278)

==========================
State vector: After QFT
==========================
|0000000000>  (0.7071067
|0000000001>  (-0.353553
|0000000011>  (-0.353553
Figure(287.294x284.278)
Enter the theta=0.3

==========================
State vector: Informatio
==========================
|0000000000>  (0.7071067
|0000000001>  (-0.233280
|0000000011>  (-0.233280
Figure(454.517x284.278)
Enter the gamma=0.2
Figure(538.128x284.278)
DensityMatrix([[ 0.56
                -0.02525
               [-0.14753
                 0.
               [-0.02525
                 0.04
               [-0.13196
                 0.
              dims=(2, 2
Figure(621.739x284.278)
Figure(788.961x284.278)
{'10': 327, '11': 605, '
Measurement results:
10 327
11 605
00 52
01 40
QBER= 0.157470703125
```

Figure 1



Thank You