

## 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 &

2025BSZ8209

C:\Users\DELL\Soumen\_2025\_BS209\QKD> Q\_OFDM.py QBER Estimation

```

125 def QBER_Estimation(counts,shots):
152 )
153
154 # Threshold plane
155 ax.plot_surface(
156     Theta, Gamma, QBER_plane,
157     color='red',
158     alpha=0.35
159 )
160
161 ax.set_xlabel("Phase noise  $\theta$  (radians)")
162 ax.set_ylabel("Amplitude damping  $\gamma$ ")
163 ax.set_zlabel("QBER")
164
165 ax.set_title("QBER Surface with QKD Security Thresh")
166
167 fig.colorbar(surf, shrink=0.5, aspect=10, label="QBER")
168
169 plt.show()
170 return qber
171 pass
172
173
174 def main():
175     QBER=[]
176     #for i in range(0,1):
177     qc = QuantumCircuit(2, 2)
178     state_vector= Statevector.from_instruction(qc)
179     show_statevec(state_vector, "Initial State Vector")
180     print(qc.draw('mpl'))
181     plt.show()
182     qc,state_vector_after_encoding=Encoding(qc)
183     qc,state_vector_after_QFT=QFT_After_Encoding(qc)
184     qc,state_vector_after_Noisy_Quantum_Channel=Noisy_Channel(qc)
185     qc=Amplitude_Loss_Non_Unitary_Kraus_operator(qc)
186     qc=IQFT_After_Noisy_Quantum_Channel(qc)
187     counts=Measurement(qc)
188     shots=4096

```

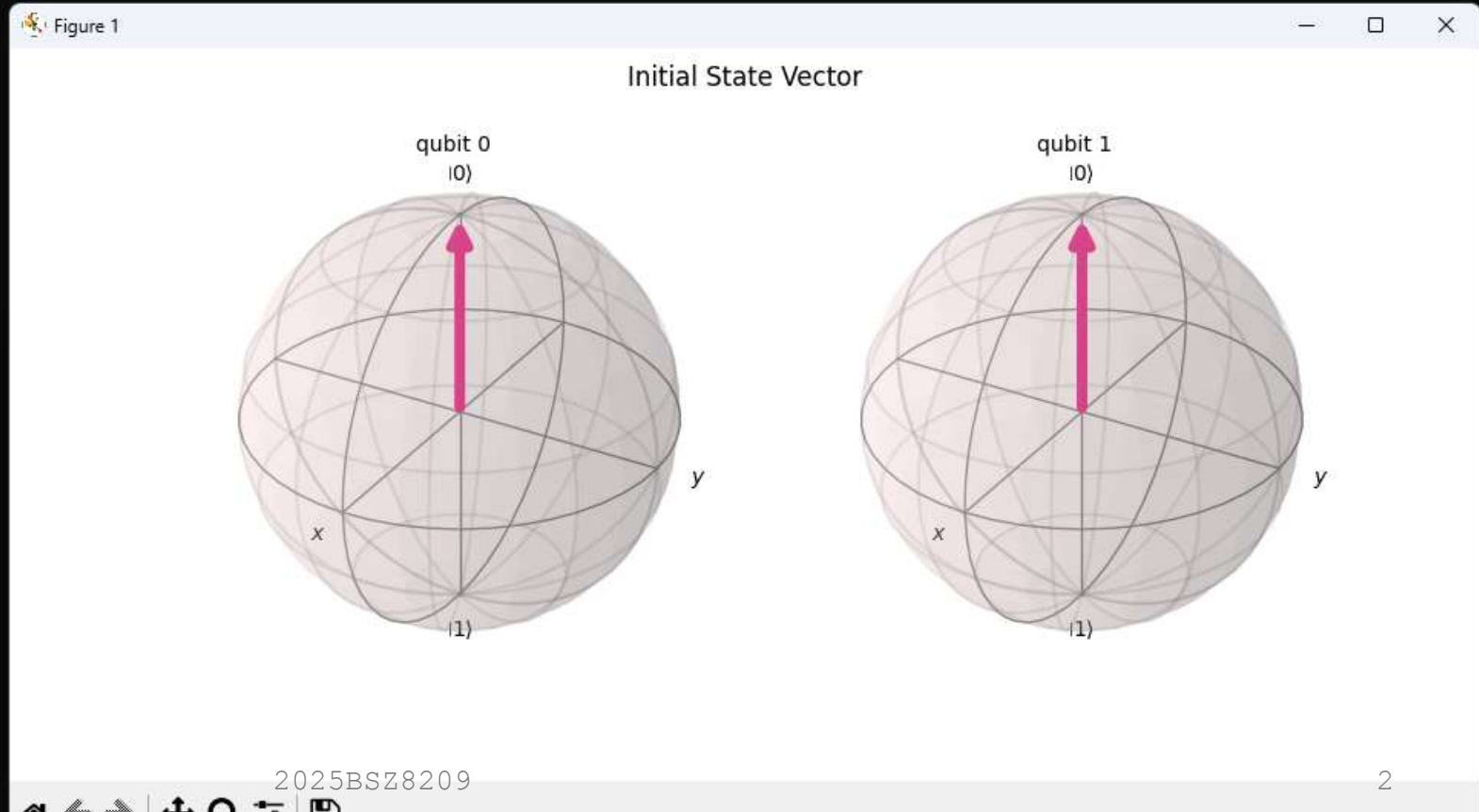
C:\Windows\system32\cmd.e: x + v

C:\Users\DELL\Soumen\_2025\_BS209\QKD>python Q\_OFDM.py

```

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

```



```

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

ax.set_xlabel("Phase noise  $\theta$  (radians)")
ax.set_ylabel("Amplitude damping  $\gamma$ ")
ax.set_zlabel("QBER")

ax.set_title("QBER Surface with QKD Security Thresh")

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

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_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

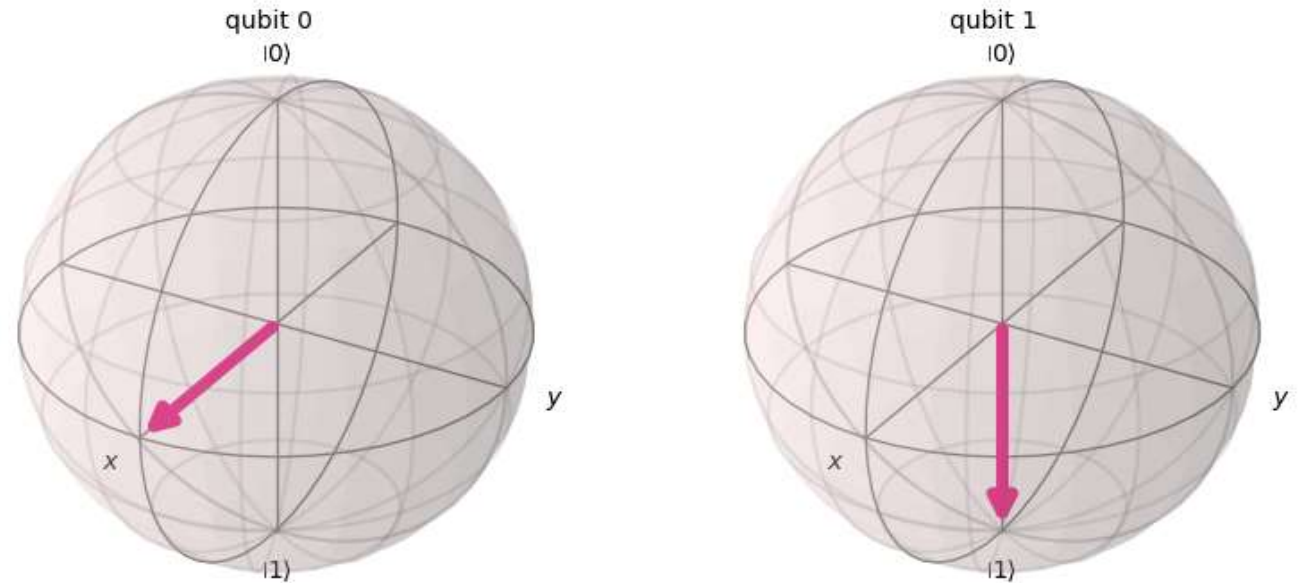
|0000000000> (1+0j)  
Figure(161.878x284.278)

State vector: After Encoding

|0000000010> (0.7071067811865475+0j)  
|0000000011> (0.7071067811865475+0j)

Figure 1

State vector: After Encoding



```

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

ax.set_xlabel("Phase noise  $\theta$  (radians)")
ax.set_ylabel("Amplitude damping  $\gamma$ ")
ax.set_zlabel("QBER")

ax.set_title("QBER Surface with QKD Security Threshold")

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

plt.show()
return qber
pass

if 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_Channel(qc)
    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_BS2_8209\QKD>python Q_OFDM.py
```

```

=====
Initial State Vector
=====

```

```

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

```

```

=====
State vector: After Encoding
=====

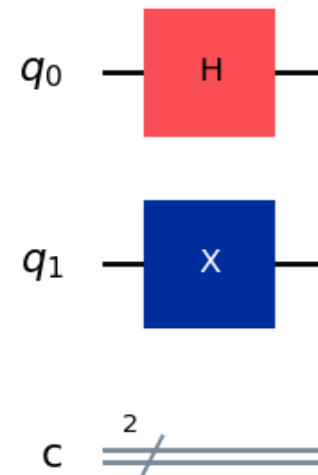
```

```

|00000000010> (0.7071067811865475+0j)
|00000000011> (0.7071067811865475+0j)
Figure(203.683x284.278)

```

Figure 1





```

ax.set_xlabel("Phase noise  $\theta$  (radians)")
ax.set_ylabel("Amplitude damping  $\gamma$ ")
ax.set_zlabel("QBER")

ax.set_title("QBER Surface with QKD Security Threshold")

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

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_Channel(qc)
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)

```

```

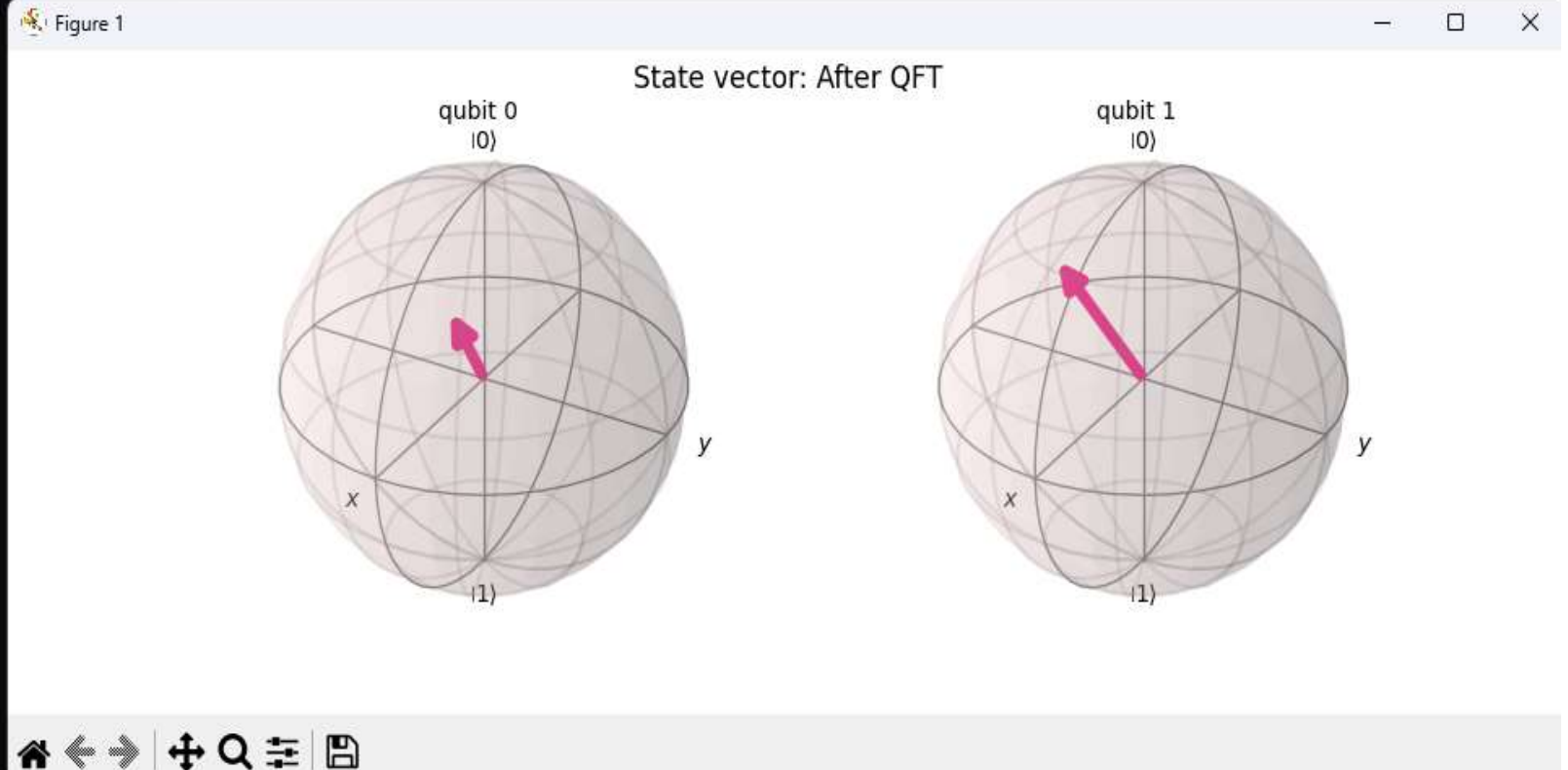
=====
State vector: After QFT
=====

```

```

|0000000000> (0.7071067811865474+0j)
|0000000001> (-0.3535533905932737-0.3535533905932737j)
|0000000011> (-0.3535533905932737+0.3535533905932737j)

```



```

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

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

k.set_xlabel("Phase noise  $\theta$  (radians)")
k.set_ylabel("Amplitude damping  $\gamma$ ")
k.set_zlabel("QBER")

k.set_title("QBER Surface with QKD Security Thresh

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

t.show()
return qber
ss

in():
BER=[]
for i in range(0,1):
t = QuantumCircuit(2, 2)
state_vector= Statevector.from_instruction(qc)
show_statevec(state_vector, "Initial State Vector")
print(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)
ounts=Measurement(qc)
ots=4096

```

```

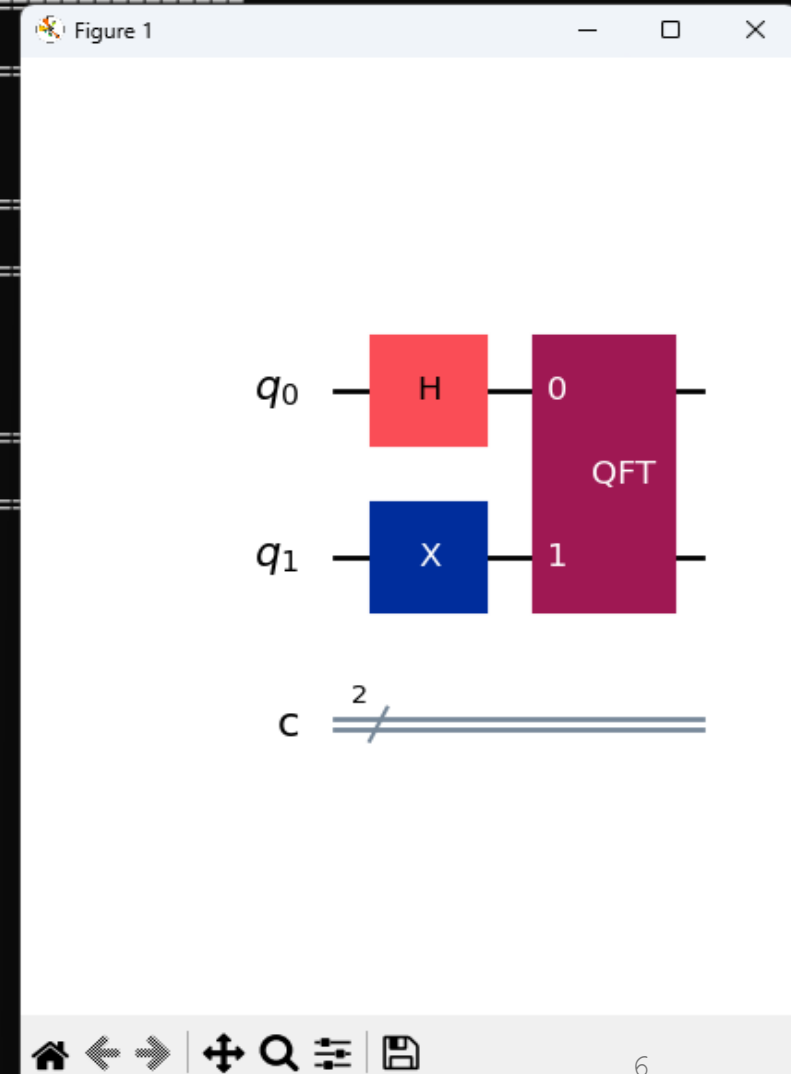
C:\Windows\system32\cmd.e: X + v
C:\Users\DELL\Soumen_2025_BSZ_8209\QKD>python Q_OFDM.py

=====
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> (-0.3535533905932737-0.3535533905932737j)
|0000000011> (-0.3535533905932737+0.3535533905932737j)
Figure(287.294x284.278)

```



Q\_OFDM.py 9+  
2\_user\_QSMA.py 8  
QCDMA\_Modified.py 8  
BER\_Estimation.py 2

C:\Users> DELL 2025BSZ8209 QKD QCDMA\_Modified.py QBER\_Estimation

```
def QFT_After_Encoding(qc):  
    pass  
    return qc, state_vector_after_QFT  
    pass  
  
#Step 3: Non-ideal quantum channel (o  
#Implements |k>+ei0k|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()  
    rho_after_loss = DensityMatrix.fr  
    print(rho_after_loss)  
    #state_vector_after_Amplitude_Loss = Statevector.from_instruction(qc)  
    return qc  
    pass  
  
def TOFT_After_Noisy_Quantum_Channel(qc):
```

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 Quant  
=====

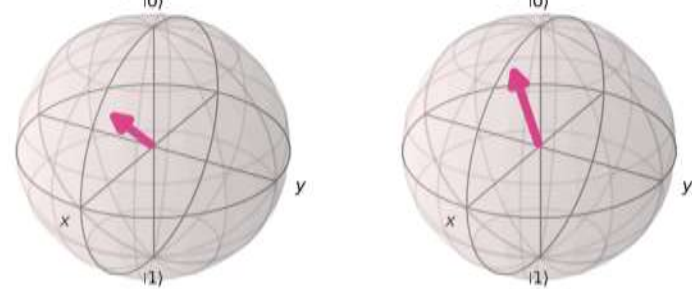
```
|0000000000> (0.7071067811865472-1.3877787807814457e-17j)  
|0000000001> (-0.23328028383389054-0.4422446259417736j)  
|0000000011> (-0.23328028383389054+0.4422446259417736j)
```

Figure 1

State vector: Information passes through after Noisy Quantum Channel

qubit 0  
|0>

qubit 1  
|0>



at which ins unitary

Ln 166, Col 1 Spaces: 4 UTF-8 CRLF Python

2025BSZ8209

21:12 18-12-2025



```

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

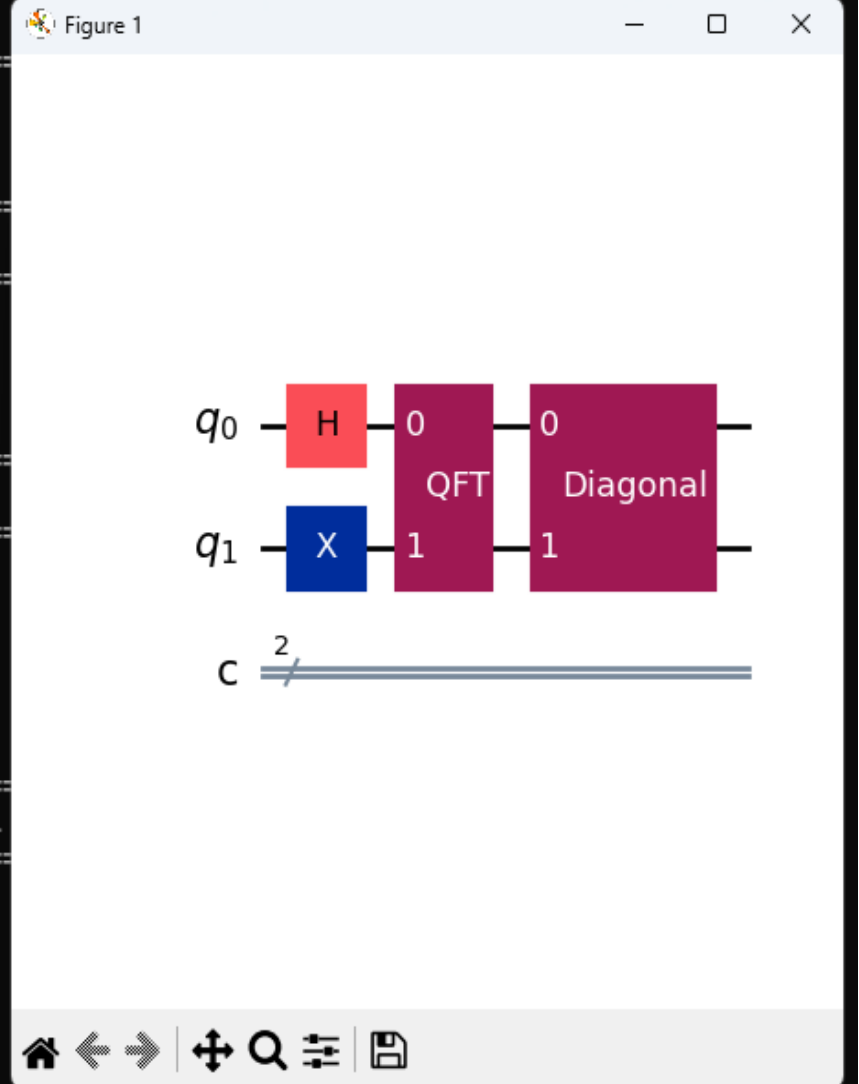
```

State vector: Information passes through after Noisy Quantum Channel

```

|0000000000> (0.7071067811865472-1.3877787807814457e-17j)
|0000000001> (-0.23328028383389054-0.4422446259417736j)
|0000000011> (-0.23328028383389054+0.4422446259417736j)
Figure(454.517x284.278)

```





Q\_OFDM.py 9+ X 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

```

43 def QFT_After_Encoding(qc):
44     plt.show()
50     return qc, state_vector_after_QFT
51     pass
52
53 #Step 3: Non-ideal quantum channel (α)
54 #Implements |k>→eiθk|k>
55 def Noisy_Quantum_Channel(qc):
56     #theta = [0.0, 0.3, 0.0, -0.3] #
57     theta=float(input("Enter the theta"))
58     theta = [0.0, theta, 0.0, -theta]
59     phase_unitary = Diagonal([
60         np.exp(1j * theta[0]), # |00>
61         np.exp(1j * theta[1]), # |01>
62         np.exp(1j * theta[2]), # |10>
63         np.exp(1j * theta[3]), # |11>
64     ])
65     qc.append(phase_unitary, [0, 1])
66     state_vector_after_Noisy_Quantum_Channel = show_statevec(state_vector_after_Noisy_Quantum_Channel)
67     print(qc.draw('mpl'))
68     plt.show()
69     return qc, state_vector_after_Noisy_Quantum_Channel
70     pass
71
72
73 def Amplitude_Loss_Non_Unitary_Kraus_operator(qc, state_vector):
74     #gamma = 0.2 # loss strength (coefficient)
75     gamma=float(input("Enter the gamma"))
76     #Amplitude loss is non-unitary, so we use Kraus operators
77     if gamma>0:
78         K0 = np.array([[1, 0], [0, np.sqrt(1-gamma)]]).T
79         K1 = np.array([[0, np.sqrt(gamma)], [0, 0]]).T
80
81         amp_damp = Kraus([K0, K1]) #
82         #Apply independently to both qubits
83         qc.append(amp_damp, [0])

```

```

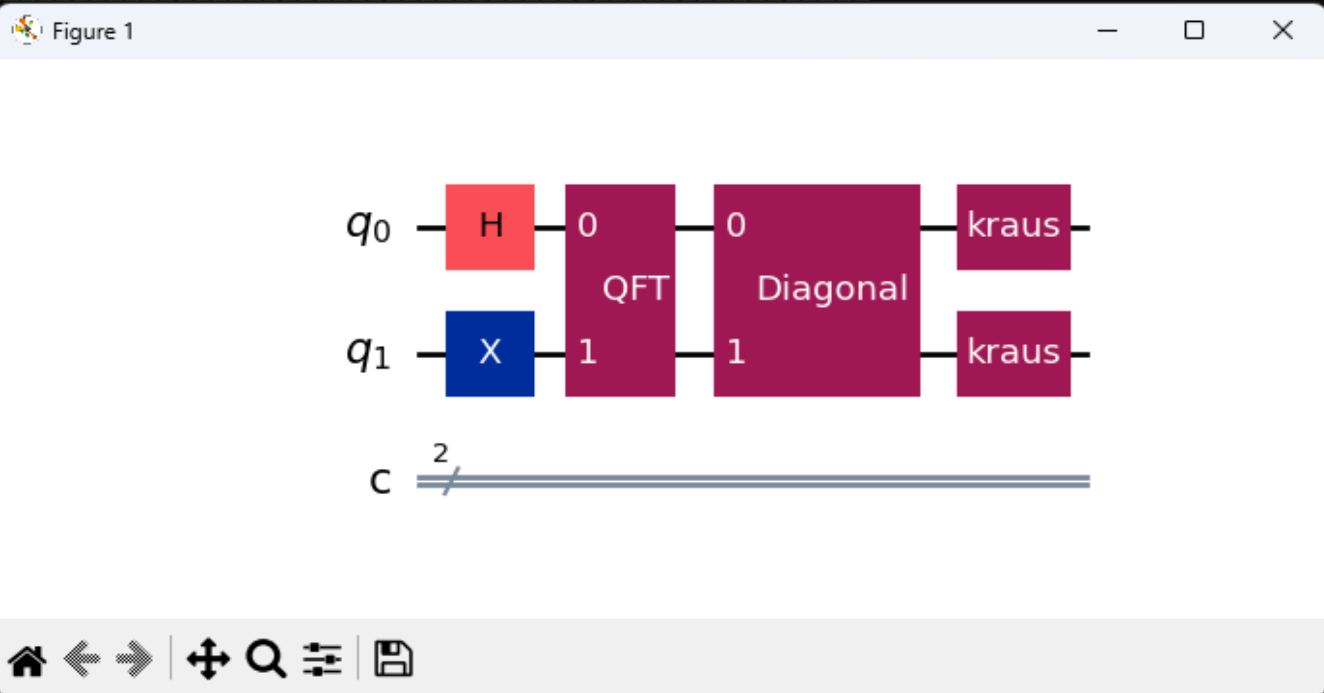
C:\Windows\system32\cmd.exe
=====
Initial State Vector
=====
|0000000000> (1+0j)
Figure(161.878x284.278)
=====
State vector: After QFT
=====
|00000000010> (0.7071067811865472-1.3877787807814457e-17j)
|00000000011> (-0.7071067811865472-1.3877787807814457e-17j)
Figure(203.683x284.278)
=====
State vector: After Amplitude Loss
=====
|00000000000> (0.7071067811865472-1.3877787807814457e-17j)
|00000000001> (-0.3535533905932738-0.3535533905932738j)
|00000000011> (-0.3535533905932738+0.3535533905932738j)
Figure(287.294x284.278)
Enter the theta=0.3

```

```

=====
State vector: Information passes through after Noisy Quantum Channel
=====
|00000000000> (0.7071067811865472-1.3877787807814457e-17j)
|00000000001> (-0.23328028383389054-0.4422446259417736j)
|00000000011> (-0.23328028383389054+0.4422446259417736j)
Figure(454.517x284.278)
Enter the gamma=0.2
Figure(538.128x284.278)

```



```

50     return qc, state_vector_after_QFT
51     pass
52
53 #Step 3: Non-ideal quantum channel (α)
54 #Implements |k⟩→eiθkk⟩
55 def Noisy_Quantum_Channel(qc):
56     #theta = [0.0, 0.3, 0.0, -0.3] #
57     theta=float(input("Enter the theta="))
58     theta = [0.0, theta, 0.0, -theta]
59     phase_unitary = Diagonal([
60         np.exp(1j * theta[0]), # |00>
61         np.exp(1j * theta[1]), # |01>
62         np.exp(1j * theta[2]), # |10>
63         np.exp(1j * theta[3]), # |11>
64     ])
65     qc.append(phase_unitary, [0, 1])
66     state_vector_after_Noisy_Quantum_Channel = state_vector_after_QFT
67     show_statevec(state_vector_after_Noisy_Quantum_Channel)
68     print(qc.draw('mpl'))
69     plt.show()
70     return qc, state_vector_after_Noisy_Quantum_Channel
71     pass
72
73 def Amplitude_Loss_Non_Unitary_Kraus(qc):
74     #gamma = 0.2 # loss strength (coefficient)
75     gamma=float(input("Enter the gamma="))
76     #Amplitude loss is non-unitary, so we use Kraus operators
77     if gamma>0:
78         K0 = np.array([[1, 0], [0, np.sqrt(1-gamma**2)]]
79         K1 = np.array([[0, np.sqrt(gamma**2)], [0, 0]])
80
81         amp_damp = Kraus([K0, K1]) #
82         #Apply independently to both
83         qc.append(amp_damp, [0])
84         qc.append(amp_damp, [1])
85
86     #state_vector_after_Amplitude_Loss = state_vector_after_Noisy_Quantum_Channel
87     #show_statevec(state_vector_after_Amplitude_Loss)
88     print(qc.draw('mpl'))
89     plt.show()

```

```

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

```

```

=====  

State vector: After Encoding  

=====  

|0000000010> (0.7071067811865472-1.3877787807814457e-17j)  

|0000000011> (0.7071067811865472+1.3877787807814457e-17j)  

Figure(203.683x284.278)

```

```

=====  

State vector: After QFT  

=====  

|0000000000> (0.7071067811865472-1.3877787807814457e-17j)  

|0000000001> (-0.3535533905932738+0.3535533905932738j)  

|0000000011> (-0.3535533905932738-0.3535533905932738j)  

Figure(287.294x284.278)  

Enter the theta=0.3

```

```

=====  

State vector: Information  

=====  

|0000000000> (0.7071067811865472-1.3877787807814457e-17j)  

|0000000001> (-0.23328028383389054-0.4422446259417736j)  

|0000000011> (-0.23328028383389054+0.4422446259417736j)  

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      ]],  

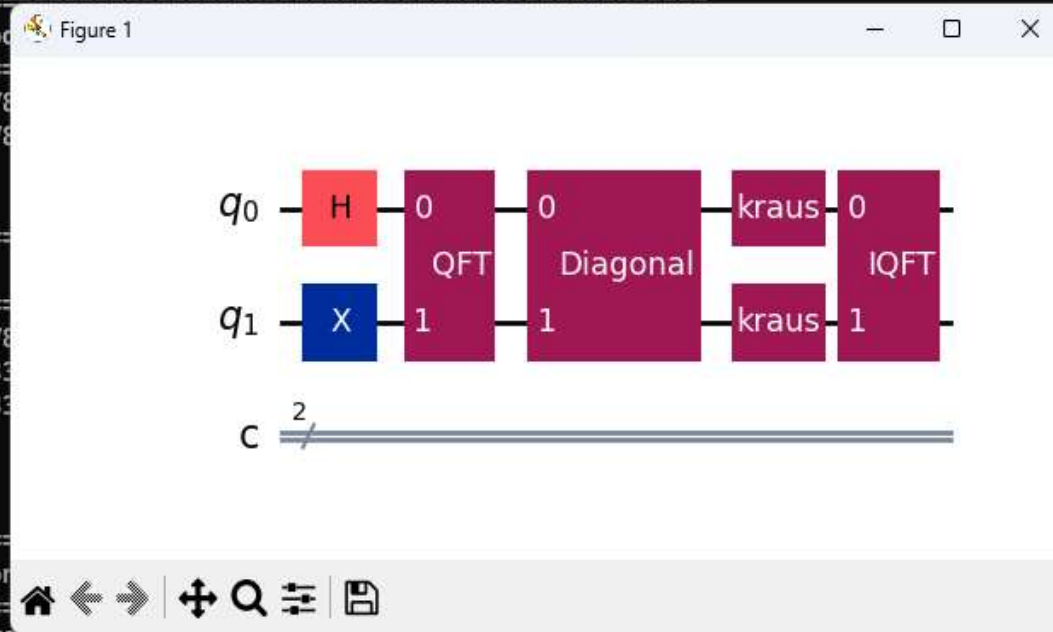
               dims=(2, 2))

```

```

Figure(621.739x284.278)

```



```

Q_OFDM.py 9+ X 2_user_QSMA.py 8 QCDMA_Modified.py 8 BER_Estimation.py 2
> Users > DELL > Soumen_2025_BS2_8209 > QKD > Q_OFDM.py > Amplitude_Loss_Non_Unitary_Kraus_operator
43 def QFT_After_Encoding(qc):
44     plt.show()
50     return qc, state_vector_after_QFT
51     pass
52
53 #Step 3: Non-ideal quantum channel (α)
54 #Implements |k>→eiθk|k>
55 def Noisy_Quantum_Channel(qc):
56     #theta = [0.0, 0.3, 0.0, -0.3] #
57     theta=float(input("Enter the theta"))
58     theta = [0.0, theta, 0.0, -theta]
59     phase_unitary = Diagonal([
60         np.exp(1j * theta[0]), # |00>
61         np.exp(1j * theta[1]), # |01>
62         np.exp(1j * theta[2]), # |10>
63         np.exp(1j * theta[3]), # |11>
64     ])
65     qc.append(phase_unitary, [0, 1])
66     state_vector_after_Noisy_Quantum_Channel = show_statevec(state_vector_after_QFT)
67     print(qc.draw('mpl'))
68     plt.show()
69     return qc, state_vector_after_Noisy_Quantum_Channel
70     pass
71
72
73 def Amplitude_Loss_Non_Unitary_Kraus_operator(qc):
74     #gamma = 0.2 # loss strength (coefficient)
75     gamma=float(input("Enter the gamma"))
76     #Amplitude loss is non-unitary, so we use Kraus operators
77     if gamma>0:
78         K0 = np.array([[1, 0], [0, np.sqrt(1-gamma)]]).T
79         K1 = np.array([[0, np.sqrt(gamma)], [0, 0]]).T
80
81         amp_damp = Kraus([K0, K1]) #
82         #Apply independently to both qubits
83         qc.append(amp_damp, [0, 1])

```

Figure(161.878x284.278)

State vector: After Encoding

```

=====
|0000000010> (0.7071067811865475+0j)
|0000000011> (0.7071067811865475+0j)
Figure(203.683x284.278)
=====

```

State vector: A

```

=====
|0000000000> (0.7071067811865475+0j)
|0000000001> (0.7071067811865475+0j)
|0000000010> (0.7071067811865475+0j)
|0000000011> (0.7071067811865475+0j)
Figure(287.294x284.278)
Enter the theta
=====

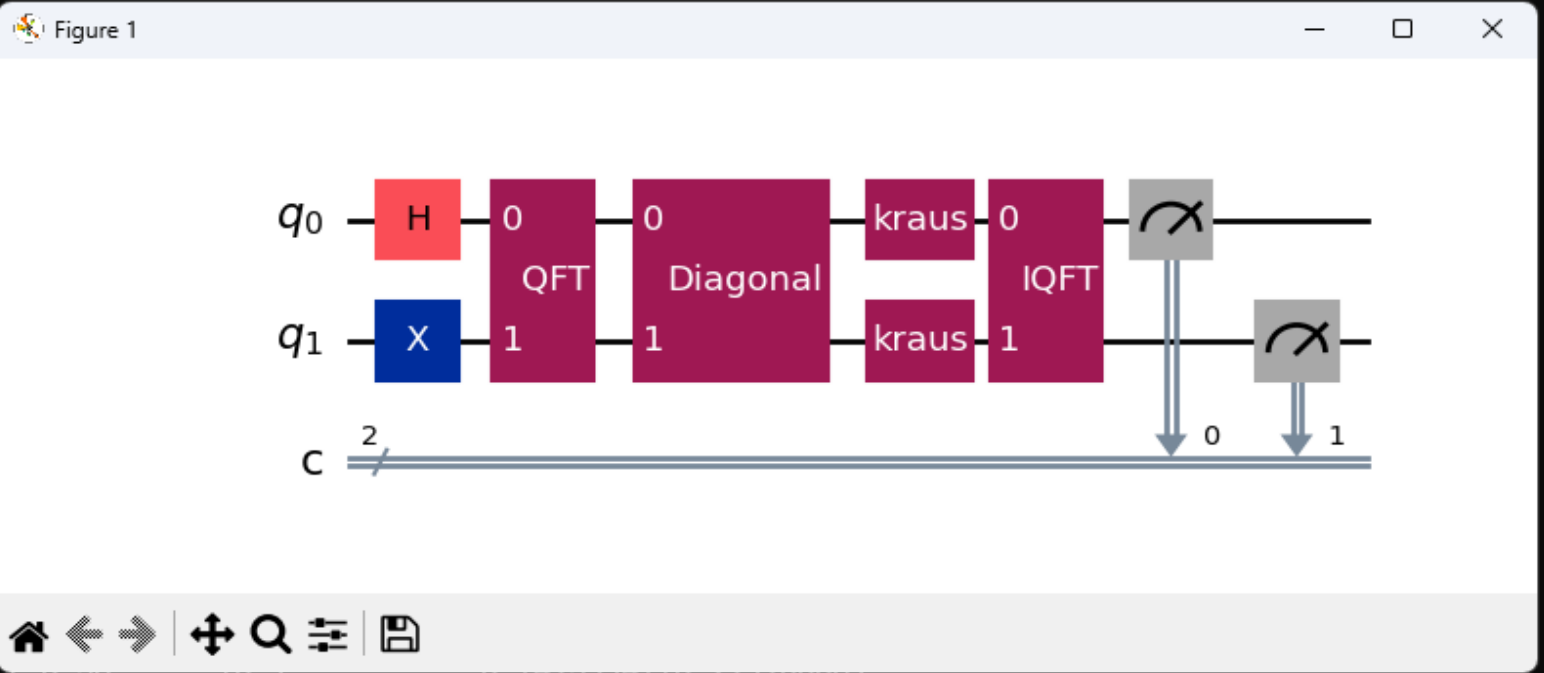
```

State vector: I

```

=====
|0000000000> (0.7071067811865475+0j)
|0000000001> (0.7071067811865475+0j)
|0000000010> (0.7071067811865475+0j)
|0000000011> (0.7071067811865475+0j)
Figure(454.517x284.278)
Enter the gamma
Figure(538.128x284.278)
DensityMatrix([

```





```

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 theta"))
    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_Channel = show_statevec(state_vector_after_QFT)
    print(qc.draw('mpl'))
    plt.show()
    return qc, state_vector_after_Noisy_Quantum_Channel
pass

def Amplitude_Loss_Non_Unitary_Kraus(qc):
    #gamma = 0.2 # loss strength (coefficient)
    gamma=float(input("Enter the gamma"))
    #Amplitude loss is non-unitary, so we use Kraus operators
    if gamma>0:
        K0 = np.array([[1, 0], [0, np.sqrt(1-gamma**2)]])
        K1 = np.array([[0, np.sqrt(gamma**2)], [0, 0]])
        amp_damp = Kraus([K0, K1]) #
        #Apply independently to both qubits
        qc.append(amp_damp, [0])
        qc.append(amp_damp, [1])

    #state_vector_after_Amplitude_Loss = show_statevec(state_vector_after_Noisy_Quantum_Channel)
    #show_statevec(state_vector_after_Amplitude_Loss)
    print(qc.draw('mpl'))
    plt.show()

```

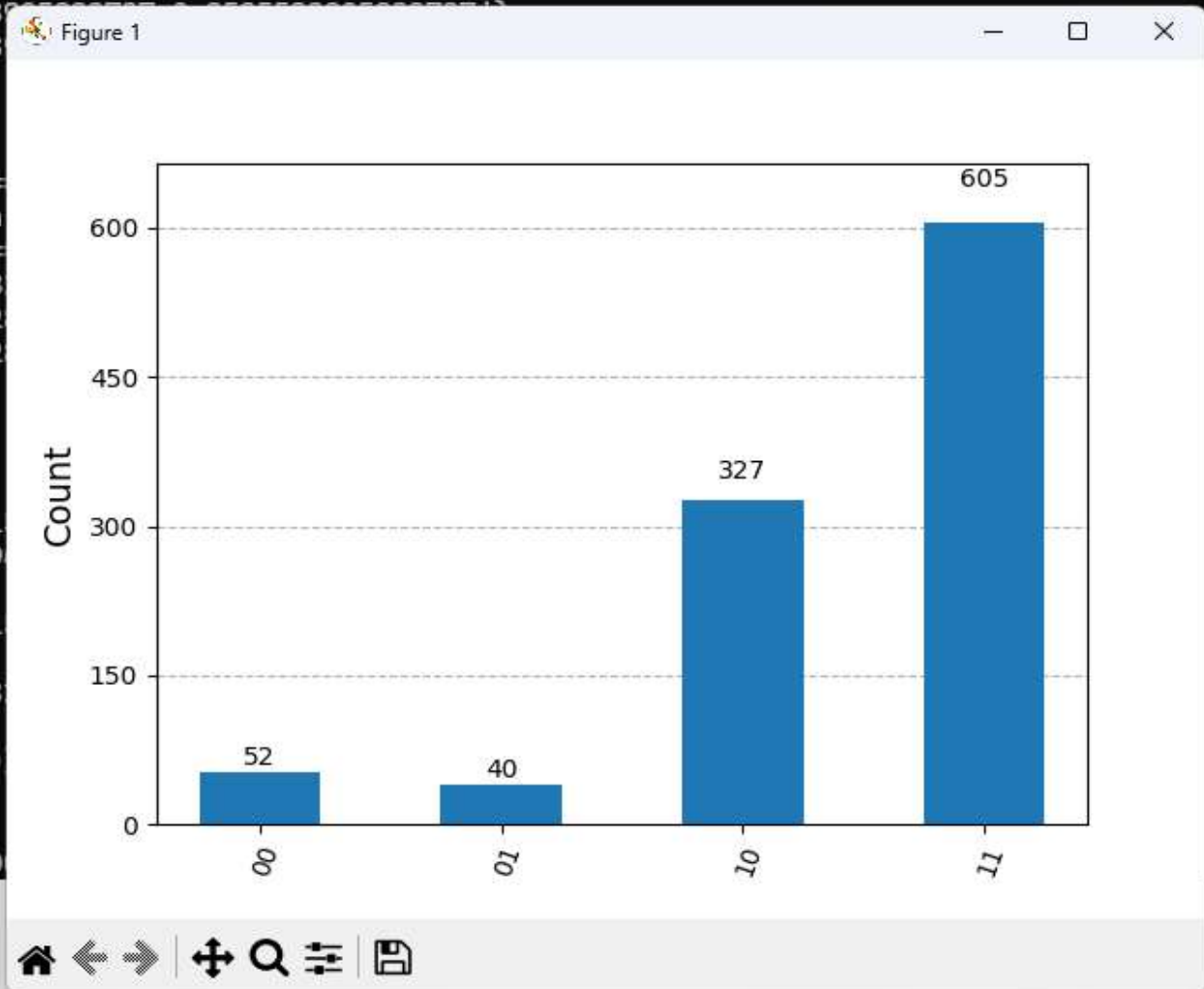
```

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

=====
State vector: After QFT
=====
|0000000000> (0.7071067811865474+0j)
|0000000001> (-0.3535533905932738+0j)
|0000000010> (-0.3535533905932738+0j)
Figure(287.294x284.278)
Enter the theta=0.3

=====
State vector: Information
=====
|0000000000> (0.7071067811865475+0j)
|0000000001> (-0.233280213480498+0j)
|0000000010> (-0.233280213480498+0j)
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.
                  -0.131963
                   0.
                  -0.147539
                   0.
                  -0.025251
                   0.
                  -0.025251
                   0.])
Figure(621.739x284.278)
Figure(788.961x284.278)
{'10': 327, '11': 605, '00': 52, '01': 40}
Measurement results:
10 327
11 605
00 52
01 40

```





```

for outcome, count in counts.items():
    print(outcome, count)
plot_histogram(counts)
plt.show()
return counts
pass

def QBER_Estimation(counts, shots):
    error_counts = sum(v for k, v in counts.items() if k != '00')
    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, gamma)

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

    # Security threshold
    QBER_th = 0.11
    QBER_plane = QBER_th * np.ones_like(Theta)

    # Plot
    fig = plt.figure(figsize=(10, 7))
    ax = fig.add_subplot(111, projection='3d')

    # 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
    )

```

```

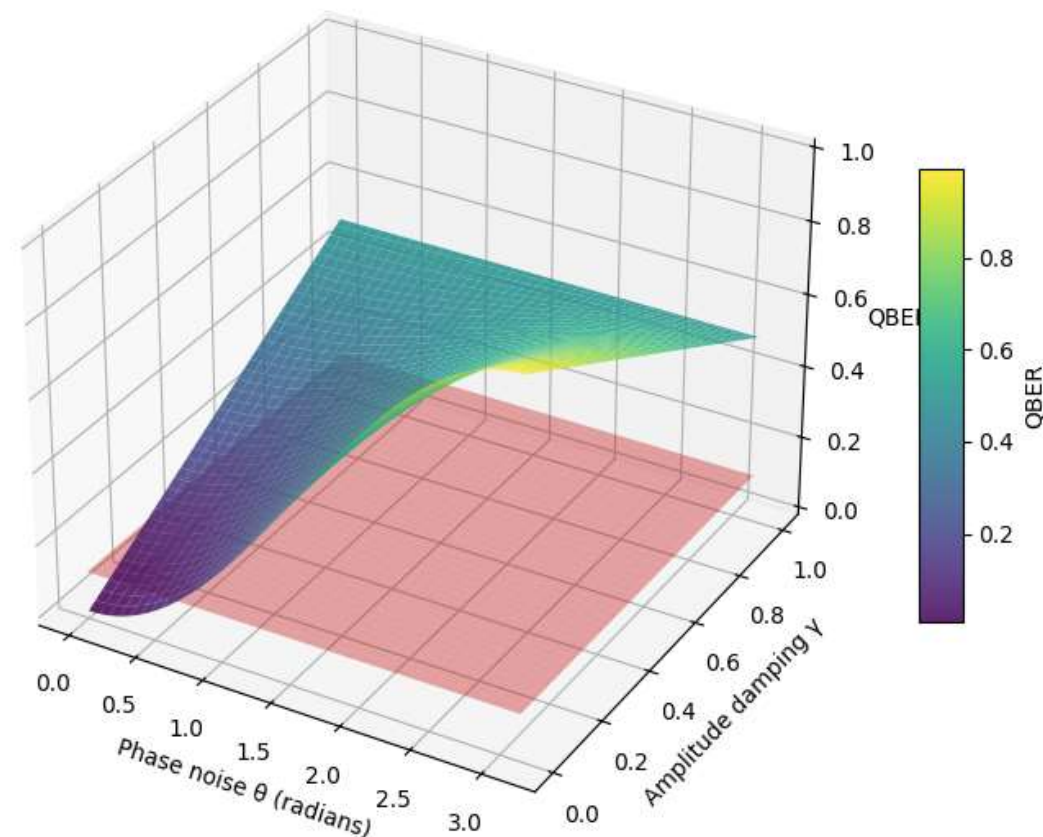
C:\Windows\system32\cmd.exe
Figure(203.683x284.278)

=====
State vector: After QFT
=====
|0000000000> (0.70710678)
|0000000001> (-0.35355339)
|0000000011> (-0.35355339)
Figure(287.294x284.278)
Enter the theta=0.3

=====
State vector: Information
=====
|0000000000> (0.70710678)
|0000000001> (-0.23328095)
|0000000011> (-0.23328095)
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.
                  [0.13196
                   0.04
                  [-0.14753
                   0.
                  [-0.02525
                   -0.02525
                   0.56]])
Figure(621.739x284.278)
Figure(788.961x284.278)
{'10': 327, '11': 605, '00': 52, '01': 40}
Measurement results:
10 327
11 605
00 52
01 40
QBER= 0.157470703125

```

QBER Surface with QKD Security Threshold (QBER = 11%)



Thank You