EE2025 Independent Project (2019-20)
Programming Assignment - 1

GUGULOTHU YASHWANTH NAIK (EE18BTECH11017)

P AASHRITH (EE18BTECH11035)

1 Simulation results :

$\frac{E_b}{N_o}$ **in dB**	**Bit Error rate**	$Q(\sqrt{\frac{2E_b}{N_o}})$
10	0.3298	0.32736
-5	0.2141	0.21322
0	0.07927	0.07868
5	0.005090	0.005953

Table 1: results

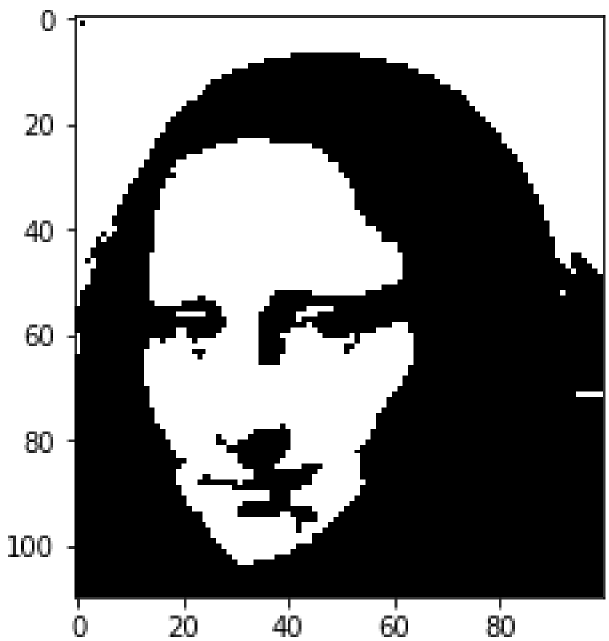
As value of $\frac{E_b}{N_o}$ in dB increases the BER(bit error rate) decreases.
The obtained energy per information bit is T/2.
$s(t) = x_1 cos(2\pi f_c t) + x_2 sin(2\pi f_c t)$
$variance = f_s(\frac{N_o}{2})$

Simulation Results

January 29, 2020

[9]:
```
#printing the given original MONALISA using matplotlib
import numpy as np
import matplotlib.pyplot as plt
MonaLisa=np.load('binary_image.npy')
print('The given image:')

plt.imshow(MonaLisa,'gray')
plt.show()
```

```
The given image:
```



[39]:
```
#calculating the energy per information bit
import numpy as np
import matplotlib.pyplot as plt
MonaLisa=np.load('binary_image.npy')                    #Loads the Input␣
 ↪bits given as a matrix        from given .npy file
```

```
b = np.reshape(MonaLisa, (1,np.product(MonaLisa.shape)))[0] # b converts␣
↪(110,100) matrix to 1d array
T=(10)**(-6)
x=np.zeros(11000)                                    # Creates a row matrix with␣
↪11000 elements as zeroes
T_s=2*(10**(-8))                                        # Defining Time period␣
↪of signal
f_c=2*(10**(6))                                              # Defining␣
↪Frequency of the signal
f_s=50*(10**(6))                                        # Defining Sampling␣
↪frequency

# This for loop encodes bits into constellations

for i in range (0,11000):
        if (b[i]==0):
                x[i]=1
        else:
                x[i]=(-1)



# Modulation
# Discrete time model
s_t=[]
for j in range (0,5500):                # Creates a waveform to transmit
        for n in range(50*(j),50*(j+1)):
                        s_t.append(x[2*j]*(np.cos(2*(np.
↪pi)*(f_c)*(n*(T_s))))+(x[(2*j)+1]*(np.sin(2*(np.pi)*(f_c)*(n*(T_s))))))




# Energy Calculation per informtion bit
l=0
for k in range(0,275000):
        l=l+((s_t[k])**2)
m=(l/550000)*(T)
print('The energy per information bit is :',m)
```

```
('The energy per information bit is :', 5.00000000000075e-07)
```

[43]:
```
#computes bit error rate,no of error bits and final recieved image for E_b/N_o␣
↪= 5 in dB
import numpy as np
import matplotlib.pyplot as plt
MonaLisa=np.load('binary_image.npy')                      #Loads the Input␣
↪bits given as a matrix        from given .npy file
```

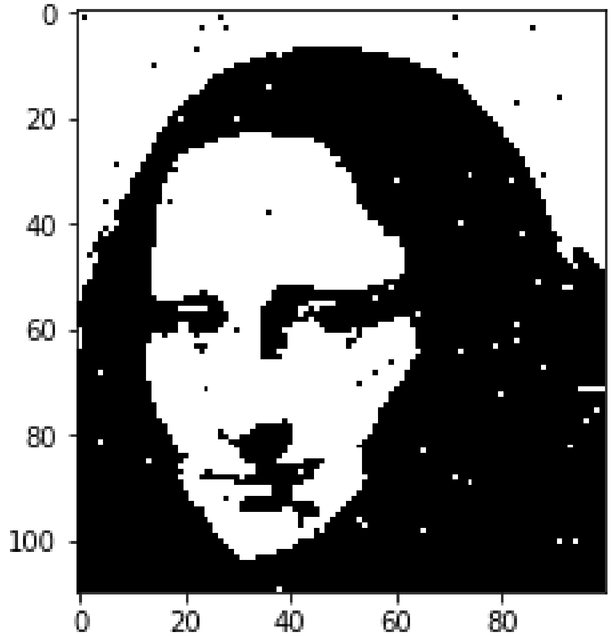
```
b = np.reshape(MonaLisa, (1,np.product(MonaLisa.shape)))[0] # b converts␣
 ↪(110,100) matrix to 1d array
T=(10)**(-6)
x=np.zeros(11000)                                    # Creates a row matrix with␣
 ↪11000 elements as zeroes
T_s=2*(10**(-8))                                         # Defining Time period␣
 ↪of signal
f_c=2*(10**(6))                                                 # Defining␣
 ↪Frequency of the signal
f_s=50*(10**(6))                                         # Defining Sampling␣
 ↪frequency

# This for loop encodes bits into constellations

for i in range (0,11000):
        if (b[i]==0):
                x[i]=1
        else:
                x[i]=(-1)



# Modulation
# Discrete time model
s_t=[]
for j in range (0,5500):                # Creates a waveform to transmit
        for n in range(50*(j),50*(j+1)):
                        s_t.append(x[2*j]*(np.cos(2*(np.
 ↪pi)*(f_c)*(n*(T_s))))+(x[(2*j)+1]*(np.sin(2*(np.pi)*(f_c)*(n*(T_s))))))




# Energy Calculation per informtion bit
l=0
for k in range(0,275000):
        l=l+((s_t[k])**2)
m=(l/550000)*(T)
print('The energy per information bit is :',m)


# Calculating power spectral Density of Gaussian Random process from Obtained␣
 ↪E_b
k=int(input('enter the value of E_b/N_o :'))
v=m/2
k_1=float(k/10)
sigma=np.sqrt((v)*(f_s)/((10)**(k_1)))
```

```
# Creates Discrete AWGN channel with mean:0 and variance:N_o/2
w=np.random.normal(0,np.sqrt(3.952),275000)


# Recieved wave form Through AWGN channel
r=s_t+w


# Demodulation by minimum distance decoding
# Creating four signals s1,s2,s3,s4 as constellation in 4-QAM modulation scheme
s1=np.zeros(275000)
s2=np.zeros(275000)
s3=np.zeros(275000)
s4=np.zeros(275000)
# u1,u2,u3,u4 are distances of r from constellations s1,s2,s3,s4 respectively
u1=np.zeros(5500)
u2=np.zeros(5500)
u3=np.zeros(5500)
u4=np.zeros(5500)
# This for loop calculate distances u1,u2,u3,u4
# Calcualting distances of 50 samples from both s_i(i=1,2,3,4) and r and␣
↪storing it in u1,u2,u3,u4 respectively

for e in range(0,5500):
        for n in range(50*(e),50*(e+1)):
                s1[n]=((np.cos(2*(np.pi)*(f_c)*(n*(T_s))))+(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s)))))
                s2[n]=(np.cos(2*(np.pi)*(f_c)*(n*(T_s))))-(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s))))
                s3[n]=-(np.cos(2*(np.pi)*(f_c)*(n*(T_s))))+(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s))))
                s4[n]=-(np.cos(2*(np.pi)*(f_c)*(n*(T_s))))-(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s))))
                u1[e]=u1[e]+(r[n]-s1[n])**2
                u2[e]=u2[e]+(r[n]-s2[n])**2
                u3[e]=u3[e]+(r[n]-s3[n])**2
                u4[e]=u4[e]+(r[n]-s4[n])**2

# Taking the minimum values from [u1[i],u2[i],u3[i],u4[i] ] for i from 0 to 5499
y_1=[]
for o in range(0,5500):
        y=[u1[o],u2[o],u3[o],u4[o]]
        y_1.append(min(y))


# Codes u1,u2,u3,u4 to 1,2,3,4 and stores the values in y_2
```

```
y_2=[]
for h in range(0,5500):
        if (y_1[h]== u1[h]):
                y_2.append(1)
        elif (y_1[h]==u2[h]):
                y_2.append(2)
        elif (y_1[h]==u3[h]):
                y_2.append(3)
        elif (y_1[h]==u4[h]):
                y_2.append(4)
c=np.zeros(11000)

# assigning bits to corresponding constellation points and stores it in array c
for p in range(0,5500):
        if (y_2[p]==1):
                c[2*p]=0
                c[2*(p)+1]=0
        elif (y_2[p]==2):
                c[2*p]=0
                c[2*(p)+1]=1
        elif (y_2[p]==3):
                c[2*p]=1
                c[2*(p)+1]=0
        elif (y_2[p]==4):
                c[2*p]=1
                c[2*(p)+1]=1

# Reshapes the array c to matrix(110,100)
d = c.reshape(110,100)


#Calculates bit error rate and no.of error bits
z_1=np.zeros(11000)
for q in range(0,11000):
        if (c[q]==b[q]):
                z_1[q]=0
        else :
                z_1[q]=1
k_1=np.count_nonzero(z_1)
k=0
for i in range(0,11000):
        k=k+z_1[i]
print('The number of error bits is:',k_1)
print('The bit error rate is:',k/11000)

# plots the final image
plt.imshow(d,'gray')
```

```
plt.show()
```

```
('The energy per information bit is :', 5.00000000000075e-07)
enter the value of E_b/N_o :5
('The number of error bits is:', 67)
('The bit error rate is:', 0.006090909090909091)
```



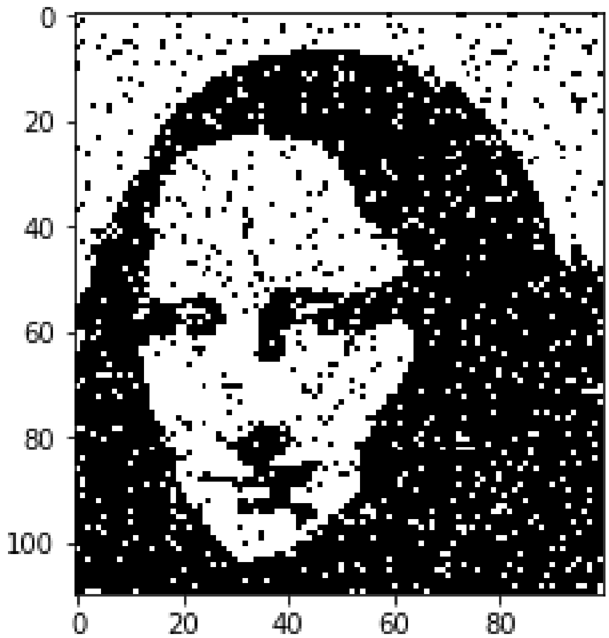
[41]:
```
#computes bit error rate,no of error bits and final recieved image for E_b/N_o␣
↪= 0 in dB
import numpy as np
import matplotlib.pyplot as plt
MonaLisa=np.load('binary_image.npy')                            #Loads the Input␣
↪bits given as a matrix        from given .npy file

b = np.reshape(MonaLisa, (1,np.product(MonaLisa.shape)))[0] # b converts␣
↪(110,100) matrix to 1d array
T=(10)**(-6)
x=np.zeros(11000)                                         # Creates a row matrix with␣
↪11000 elements as zeroes
T_s=2*(10**(-8))                                              # Defining Time period␣
↪of signal
f_c=2*(10**(6))                                                      # Defining␣
↪Frequency of the signal
f_s=50*(10**(6))                                              # Defining Sampling␣
↪frequency
```

```
# This for loop encodes bits into constellations

for i in range (0,11000):
        if (b[i]==0):
                x[i]=1
        else:
                x[i]=(-1)


# Modulation
# Discrete time model
s_t=[]
for j in range (0,5500):                # Creates a waveform to transmit
        for n in range(50*(j),50*(j+1)):
                        s_t.append(x[2*j]*(np.cos(2*(np.
 ↪pi)*(f_c)*(n*(T_s))))+(x[(2*j)+1]*(np.sin(2*(np.pi)*(f_c)*(n*(T_s))))))



# Energy Calculation per informtion bit
l=0
for k in range(0,275000):
        l=l+((s_t[k])**2)
m=(l/550000)*(T)
print('The energy per information bit is :',m)


# Calculating power spectral Density of Gaussian Random process from Obtained␣
 ↪E_b
k=int(input('enter the value of E_b/N_o :'))
v=m/2
k_1=float(k/10)
sigma=np.sqrt((v)*(f_s)/((10)**(k_1)))

# Creates Discrete AWGN channel with mean:0 and variance:N_o/2
w=np.random.normal(0,sigma,275000)


# Recieved wave form Through AWGN channel
r=s_t+w


# Demodulation by minimum distance decoding
# Creating four signals s1,s2,s3,s4 as constellation in 4-QAM modulation scheme
s1=np.zeros(275000)
s2=np.zeros(275000)
```

```
s3=np.zeros(275000)
s4=np.zeros(275000)
# u1,u2,u3,u4 are distances of r from constellations s1,s2,s3,s4 respectively
u1=np.zeros(5500)
u2=np.zeros(5500)
u3=np.zeros(5500)
u4=np.zeros(5500)
# This for loop calculate distances u1,u2,u3,u4
# Calcualting distances of 50 samples from both s_i(i=1,2,3,4) and r and␣
↪storing it in u1,u2,u3,u4 respectively

for e in range(0,5500):
        for n in range(50*(e),50*(e+1)):
                s1[n]=((np.cos(2*(np.pi)*(f_c)*(n*(T_s))))+(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s)))))
                s2[n]=(np.cos(2*(np.pi)*(f_c)*(n*(T_s))))-(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s))))
                s3[n]=-(np.cos(2*(np.pi)*(f_c)*(n*(T_s))))+(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s))))
                s4[n]=-(np.cos(2*(np.pi)*(f_c)*(n*(T_s))))-(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s))))
                u1[e]=u1[e]+(r[n]-s1[n])**2
                u2[e]=u2[e]+(r[n]-s2[n])**2
                u3[e]=u3[e]+(r[n]-s3[n])**2
                u4[e]=u4[e]+(r[n]-s4[n])**2

# Taking the minimum values from [u1[i],u2[i],u3[i],u4[i] ] for i from 0 to 5499
y_1=[]
for o in range(0,5500):
        y=[u1[o],u2[o],u3[o],u4[o]]
        y_1.append(min(y))


# Codes u1,u2,u3,u4 to 1,2,3,4 and stores the values in y_2
y_2=[]
for h in range(0,5500):
        if (y_1[h]== u1[h]):
                y_2.append(1)
        elif (y_1[h]==u2[h]):
                y_2.append(2)
        elif (y_1[h]==u3[h]):
                y_2.append(3)
        elif (y_1[h]==u4[h]):
                y_2.append(4)
c=np.zeros(11000)

# assigning bits to corresponding constellation points and stores it in array c
```

```python
for p in range(0,5500):
        if (y_2[p]==1):
                c[2*p]=0
                c[2*(p)+1]=0
        elif (y_2[p]==2):
                c[2*p]=0
                c[2*(p)+1]=1
        elif (y_2[p]==3):
                c[2*p]=1
                c[2*(p)+1]=0
        elif (y_2[p]==4):
                c[2*p]=1
                c[2*(p)+1]=1

# Reshapes the array c to matrix(110,100)
d = c.reshape(110,100)


#Calculates bit error rate and no.of error bits
z_1=np.zeros(11000)
for q in range(0,11000):
        if (c[q]==b[q]):
                z_1[q]=0
        else :
                z_1[q]=1
k_1=np.count_nonzero(z_1)
k=0
for i in range(0,11000):
        k=k+z_1[i]
print('The number of error bits is:',k_1)
print('The bit error rate is:',k/11000)

# plots the final image
plt.imshow(d,'gray')
plt.show()
```

```
('The energy per information bit is :', 5.00000000000075e-07)
enter the value of E_b/N_o :0
('The number of error bits is:', 865)
('The bit error rate is:', 0.07863636363636364)
```



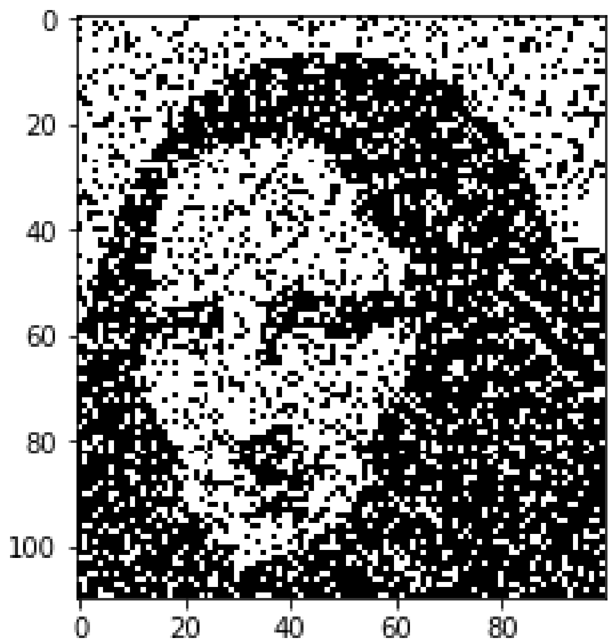
[44]:
```
#computes bit error rate,no of error bits and final recieved image for E_b/N_o␣
↪= -5 in dB
import numpy as np
import matplotlib.pyplot as plt
MonaLisa=np.load('binary_image.npy')                              #Loads the Input␣
↪bits given as a matrix       from given .npy file

b = np.reshape(MonaLisa, (1,np.product(MonaLisa.shape)))[0] # b converts␣
↪(110,100) matrix to 1d array
T=(10)**(-6)
x=np.zeros(11000)                                          # Creates a row matrix with␣
↪11000 elements as zeroes
T_s=2*(10**(-8))                                             # Defining Time period␣
↪of signal
f_c=2*(10**(6))                                                     # Defining␣
↪Frequency of the signal
f_s=50*(10**(6))                                             # Defining Sampling␣
↪frequency

# This for loop encodes bits into constellations

for i in range (0,11000):
        if (b[i]==0):
                x[i]=1
        else:
```

```
                x[i]=(-1)


# Modulation
# Discrete time model
s_t=[]
for j in range (0,5500):                # Creates a waveform to transmit
        for n in range(50*(j),50*(j+1)):
                        s_t.append(x[2*j]*(np.cos(2*(np.
 ↪pi)*(f_c)*(n*(T_s))))+(x[(2*j)+1]*(np.sin(2*(np.pi)*(f_c)*(n*(T_s))))))




# Energy Calculation per informtion bit
l=0
for k in range(0,275000):
        l=l+((s_t[k])**2)
m=(l/550000)*(T)
print('The energy per information bit is :',m)


# Calculating power spectral Density of Gaussian Random process from Obtained␣
 ↪E_b
k=int(input('enter the value of E_b/N_o :'))
v=m/2
k_1=float(k/10)
sigma=np.sqrt((v)*(f_s)/((10)**(k_1)))

# Creates Discrete AWGN channel with mean:0 and variance:N_o/2
w=np.random.normal(0,np.sqrt(39.5284),275000)


# Recieved wave form Through AWGN channel
r=s_t+w


# Demodulation by minimum distance decoding
# Creating four signals s1,s2,s3,s4 as constellation in 4-QAM modulation scheme
s1=np.zeros(275000)
s2=np.zeros(275000)
s3=np.zeros(275000)
s4=np.zeros(275000)
# u1,u2,u3,u4 are distances of r from constellations s1,s2,s3,s4 respectively
u1=np.zeros(5500)
u2=np.zeros(5500)
u3=np.zeros(5500)
u4=np.zeros(5500)
```

```
# This for loop calculate distances u1,u2,u3,u4
# Calcualting distances of 50 samples from both s_i(i=1,2,3,4) and r and␣
↪storing it in u1,u2,u3,u4 respectively

for e in range(0,5500):
        for n in range(50*(e),50*(e+1)):
                s1[n]=((np.cos(2*(np.pi)*(f_c)*(n*(T_s))))+(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s)))))
                s2[n]=(np.cos(2*(np.pi)*(f_c)*(n*(T_s))))-(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s))))
                s3[n]=-(np.cos(2*(np.pi)*(f_c)*(n*(T_s))))+(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s))))
                s4[n]=-(np.cos(2*(np.pi)*(f_c)*(n*(T_s))))-(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s))))
                u1[e]=u1[e]+(r[n]-s1[n])**2
                u2[e]=u2[e]+(r[n]-s2[n])**2
                u3[e]=u3[e]+(r[n]-s3[n])**2
                u4[e]=u4[e]+(r[n]-s4[n])**2

# Taking the minimum values from [u1[i],u2[i],u3[i],u4[i] ] for i from 0 to 5499
y_1=[]
for o in range(0,5500):
        y=[u1[o],u2[o],u3[o],u4[o]]
        y_1.append(min(y))


# Codes u1,u2,u3,u4 to 1,2,3,4 and stores the values in y_2
y_2=[]
for h in range(0,5500):
        if (y_1[h]== u1[h]):
                y_2.append(1)
        elif (y_1[h]==u2[h]):
                y_2.append(2)
        elif (y_1[h]==u3[h]):
                y_2.append(3)
        elif (y_1[h]==u4[h]):
                y_2.append(4)
c=np.zeros(11000)

# assigning bits to corresponding constellation points and stores it in array c
for p in range(0,5500):
        if (y_2[p]==1):
                c[2*p]=0
                c[2*(p)+1]=0
        elif (y_2[p]==2):
                c[2*p]=0
                c[2*(p)+1]=1
```

```
        elif (y_2[p]==3):
                c[2*p]=1
                c[2*(p)+1]=0
        elif (y_2[p]==4):
                c[2*p]=1
                c[2*(p)+1]=1

# Reshapes the array c to matrix(110,100)
d = c.reshape(110,100)


#Calculates bit error rate and no.of error bits
z_1=np.zeros(11000)
for q in range(0,11000):
        if (c[q]==b[q]):
                z_1[q]=0
        else :
                z_1[q]=1
k_1=np.count_nonzero(z_1)
k=0
for i in range(0,11000):
        k=k+z_1[i]
print('The number of error bits is:',k_1)
print('The bit error rate is:',k/11000)

# plots the final image
plt.imshow(d,'gray')
plt.show()
```

```
('The energy per information bit is :', 5.00000000000075e-07)
enter the value of E_b/N_o :-5
('The number of error bits is:', 2362)
('The bit error rate is:', 0.21472727272727274)
```



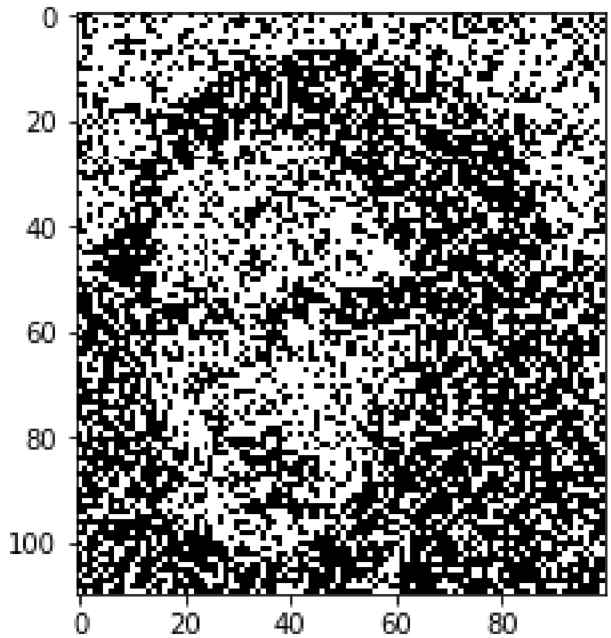
[42]:
```
#computes bit error rate,no of error bits and final recieved image for E_b/N_o␣
↪= -10 in dB
import numpy as np
import matplotlib.pyplot as plt
MonaLisa=np.load('binary_image.npy')                              #Loads the Input␣
↪bits given as a matrix        from given .npy file

b = np.reshape(MonaLisa, (1,np.product(MonaLisa.shape)))[0] # b converts␣
↪(110,100) matrix to 1d array
T=(10)**(-6)
x=np.zeros(11000)                                        # Creates a row matrix with␣
↪11000 elements as zeroes
T_s=2*(10**(-8))                                            # Defining Time period␣
↪of signal
f_c=2*(10**(6))                                                     # Defining␣
↪Frequency of the signal
f_s=50*(10**(6))                                            # Defining Sampling␣
↪frequency

# This for loop encodes bits into constellations

for i in range (0,11000):
        if (b[i]==0):
                x[i]=1
        else:
```

```
                x[i]=(-1)


# Modulation
# Discrete time model
s_t=[]
for j in range (0,5500):                # Creates a waveform to transmit
        for n in range(50*(j),50*(j+1)):
                        s_t.append(x[2*j]*(np.cos(2*(np.
 ↪pi)*(f_c)*(n*(T_s))))+(x[(2*j)+1]*(np.sin(2*(np.pi)*(f_c)*(n*(T_s))))))



# Energy Calculation per informtion bit
l=0
for k in range(0,275000):
        l=l+((s_t[k])**2)
m=(l/550000)*(T)
print('The energy per information bit is :',m)


# Calculating power spectral Density of Gaussian Random process from Obtained␣
 ↪E_b
k=int(input('enter the value of E_b/N_o :'))
v=m/2
k_1=float(k/10)
sigma=np.sqrt((v)*(f_s)/((10)**(k_1)))

# Creates Discrete AWGN channel with mean:0 and variance:N_o/2
w=np.random.normal(0,sigma,275000)


# Recieved wave form Through AWGN channel
r=s_t+w


# Demodulation by minimum distance decoding
# Creating four signals s1,s2,s3,s4 as constellation in 4-QAM modulation scheme
s1=np.zeros(275000)
s2=np.zeros(275000)
s3=np.zeros(275000)
s4=np.zeros(275000)
# u1,u2,u3,u4 are distances of r from constellations s1,s2,s3,s4 respectively
u1=np.zeros(5500)
u2=np.zeros(5500)
u3=np.zeros(5500)
u4=np.zeros(5500)
```

```
# This for loop calculate distances u1,u2,u3,u4
# Calcualting distances of 50 samples from both s_i(i=1,2,3,4) and r and␣
↪storing it in u1,u2,u3,u4 respectively

for e in range(0,5500):
        for n in range(50*(e),50*(e+1)):
                s1[n]=((np.cos(2*(np.pi)*(f_c)*(n*(T_s))))+(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s)))))
                s2[n]=(np.cos(2*(np.pi)*(f_c)*(n*(T_s))))-(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s))))
                s3[n]=-(np.cos(2*(np.pi)*(f_c)*(n*(T_s))))+(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s))))
                s4[n]=-(np.cos(2*(np.pi)*(f_c)*(n*(T_s))))-(np.sin(2*(np.
↪pi)*(f_c)*(n*(T_s))))
                u1[e]=u1[e]+(r[n]-s1[n])**2
                u2[e]=u2[e]+(r[n]-s2[n])**2
                u3[e]=u3[e]+(r[n]-s3[n])**2
                u4[e]=u4[e]+(r[n]-s4[n])**2

# Taking the minimum values from [u1[i],u2[i],u3[i],u4[i] ] for i from 0 to 5499
y_1=[]
for o in range(0,5500):
        y=[u1[o],u2[o],u3[o],u4[o]]
        y_1.append(min(y))


# Codes u1,u2,u3,u4 to 1,2,3,4 and stores the values in y_2
y_2=[]
for h in range(0,5500):
        if (y_1[h]== u1[h]):
                y_2.append(1)
        elif (y_1[h]==u2[h]):
                y_2.append(2)
        elif (y_1[h]==u3[h]):
                y_2.append(3)
        elif (y_1[h]==u4[h]):
                y_2.append(4)
c=np.zeros(11000)

# assigning bits to corresponding constellation points and stores it in array c
for p in range(0,5500):
        if (y_2[p]==1):
                c[2*p]=0
                c[2*(p)+1]=0
        elif (y_2[p]==2):
                c[2*p]=0
                c[2*(p)+1]=1
```

```
        elif (y_2[p]==3):
                c[2*p]=1
                c[2*(p)+1]=0
        elif (y_2[p]==4):
                c[2*p]=1
                c[2*(p)+1]=1

# Reshapes the array c to matrix(110,100)
d = c.reshape(110,100)


#Calculates bit error rate and no.of error bits
z_1=np.zeros(11000)
for q in range(0,11000):
        if (c[q]==b[q]):
                z_1[q]=0
        else :
                z_1[q]=1
k_1=np.count_nonzero(z_1)
k=0
for i in range(0,11000):
        k=k+z_1[i]
print('The number of error bits is:',k_1)
print('The bit error rate is:',k/11000)

# plots the final image
plt.imshow(d,'gray')
plt.show()
```

```
('The energy per information bit is :', 5.00000000000075e-07)
enter the value of E_b/N_o :-10
('The number of error bits is:', 3525)
('The bit error rate is:', 0.32045454545454544)
```



0
20
40
60
80
100
0
20
40
60
80


```
[ ]:
```