



Cairo University  
Faculty of Engineering

Department of Computer  
Engineering



## ELC 3070 – Spring 2024

### Communications 2

# Project #3

### Submitted to

Dr. Mohamed Khairy

Dr. Nafie

Eng. Mohamed Khaled

### Submitted by

Team: 12

Name	Section	BN
عمرو عبد المتجلي احمد متولي	3	16
علياء عصام الدين نجيب محمد	3	6

## Role of each member:

Name	Role
عمرو عبد المتجلي احمد متولي	<ul style="list-style-type: none"><li>Made the code &amp; some of the report</li></ul>
علياء عصام الدين نجيب محمد	<ul style="list-style-type: none"><li>Made the code and some of the report</li></ul>

## Contents

BPSK .....	
Comment: .....	
QPSK Gray and NOT-Grey Encoded: .....	
Comment: .....	
8PSK .....	
Comment: .....	
16 QAM .....	
Comment: .....	
BFSK .....	
Comment: .....	

## Figures:

Figure 1: Theoretical and practical BER vs EB/No for BPSK.....	
Figure 2 Theoretical and practical BER vs EB/No for QPSK Grey-Encoded.....	
Figure 3 Theoretical and practical BER vs EB/No for QPSK Not-Grey-Encoded.....	
Figure 4: Theoretical and practical BER vs EB/No for 8PSK .....	
Figure 5: Theoretical and practical BER vs EB/No for 16QAM.....	
Figure 6: Theoretical and practical BER vs EB/No for all four modulation schemes. ....	
Figure 7: Theoretical and practical BER vs EB/No for QPSK Gray-Encoded and Not-Grey-Encoded. ....	
Figure 8: Theoretical and practical BER vs EB/No for BFSK. ....	
Figure 8: PSD of BFSK .....	

## 1. Tasks:

- 1.1. Theoretical vs practical BER graph of the BPSK:

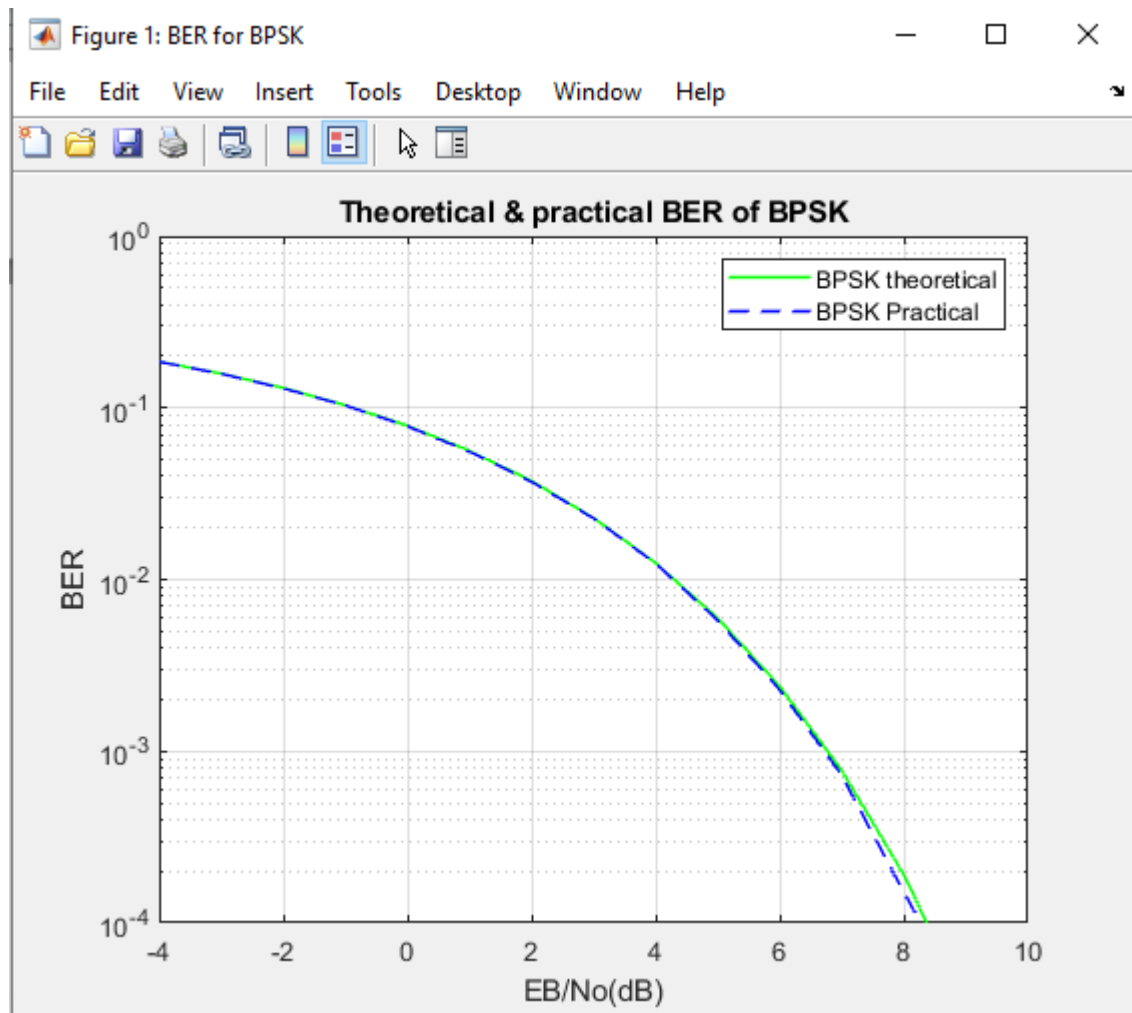


Figure 1: Theoretical and practical BER vs  $E_b/N_0$  for BPSK

**Comment:** As we can see they are almost the same and they become more similar when the number of binary data used increases, we used number of bits equals  $48 \times 100000$ .

- **1.2. Theoretical vs practical BER graph of the QPSK Gray-Encoded:**

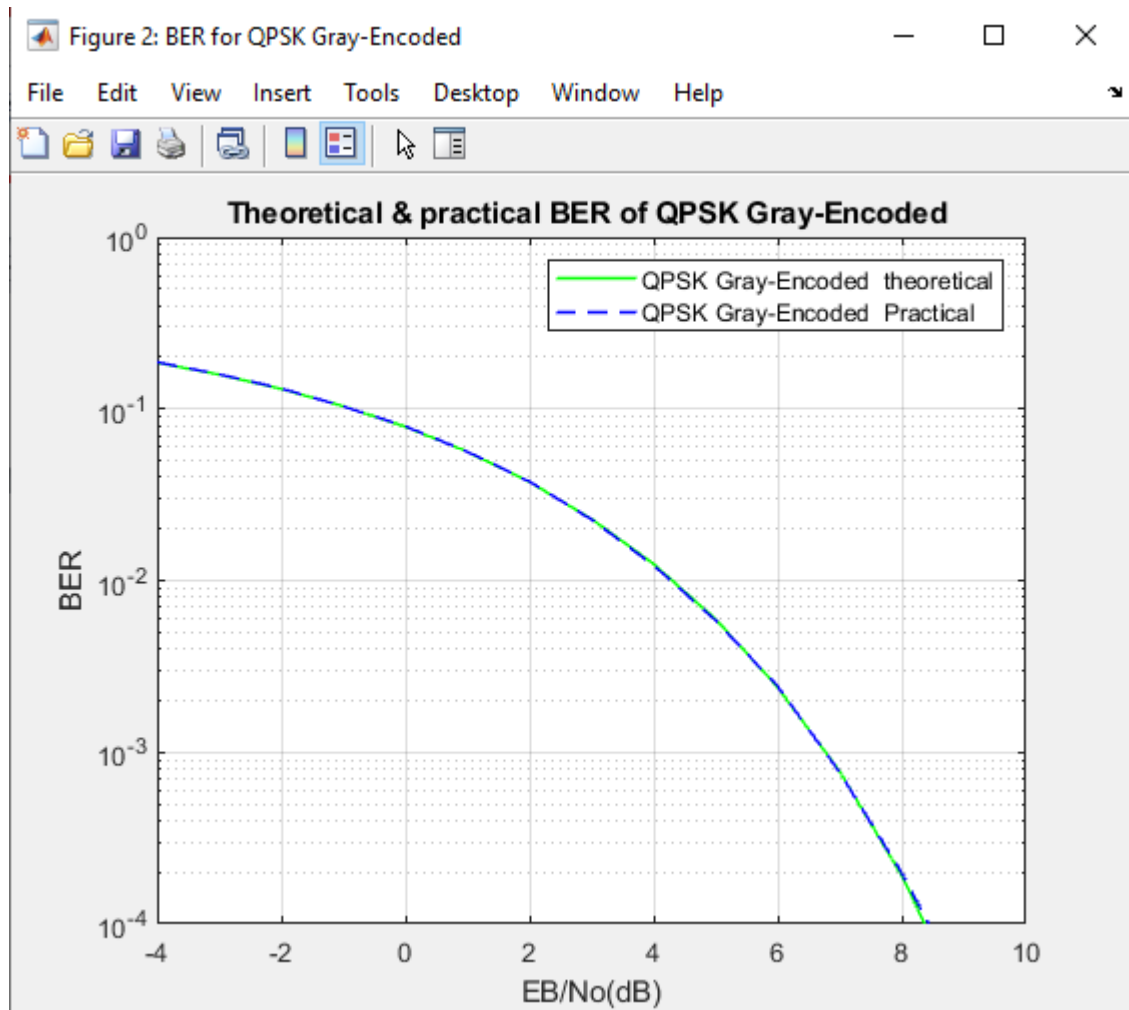


Figure 2: Theoretical and practical BER vs  $E_b/N_0$  for QPSK Grey-Encoded

**Comment:** As we can see they are almost the same.

- **1.3. Theoretical vs practical BER graph of the QPSK Not-Grey-Encoded:**

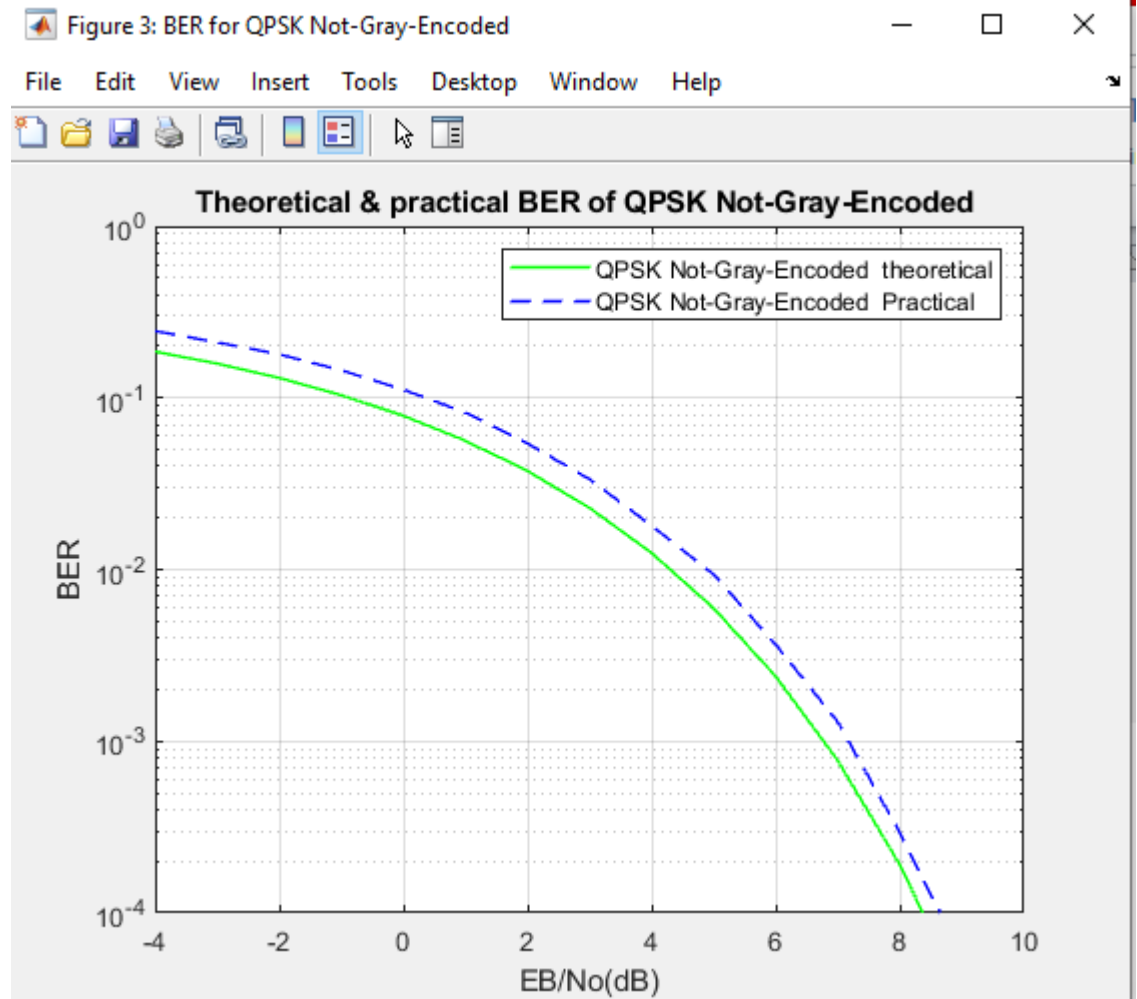


Figure 3: Theoretical and practical BER vs  $E_b/N_0$  for QPSK Not-Grey-Encoded

**Comment:** As we can see the practical BER of QPSK Not-Grey-Encoded is above the theoretical with a slightly larger value indicating there is more error as there will be two bits changes in some cases.

- **1.4. Theoretical vs practical BER graph of the 8PSK:**

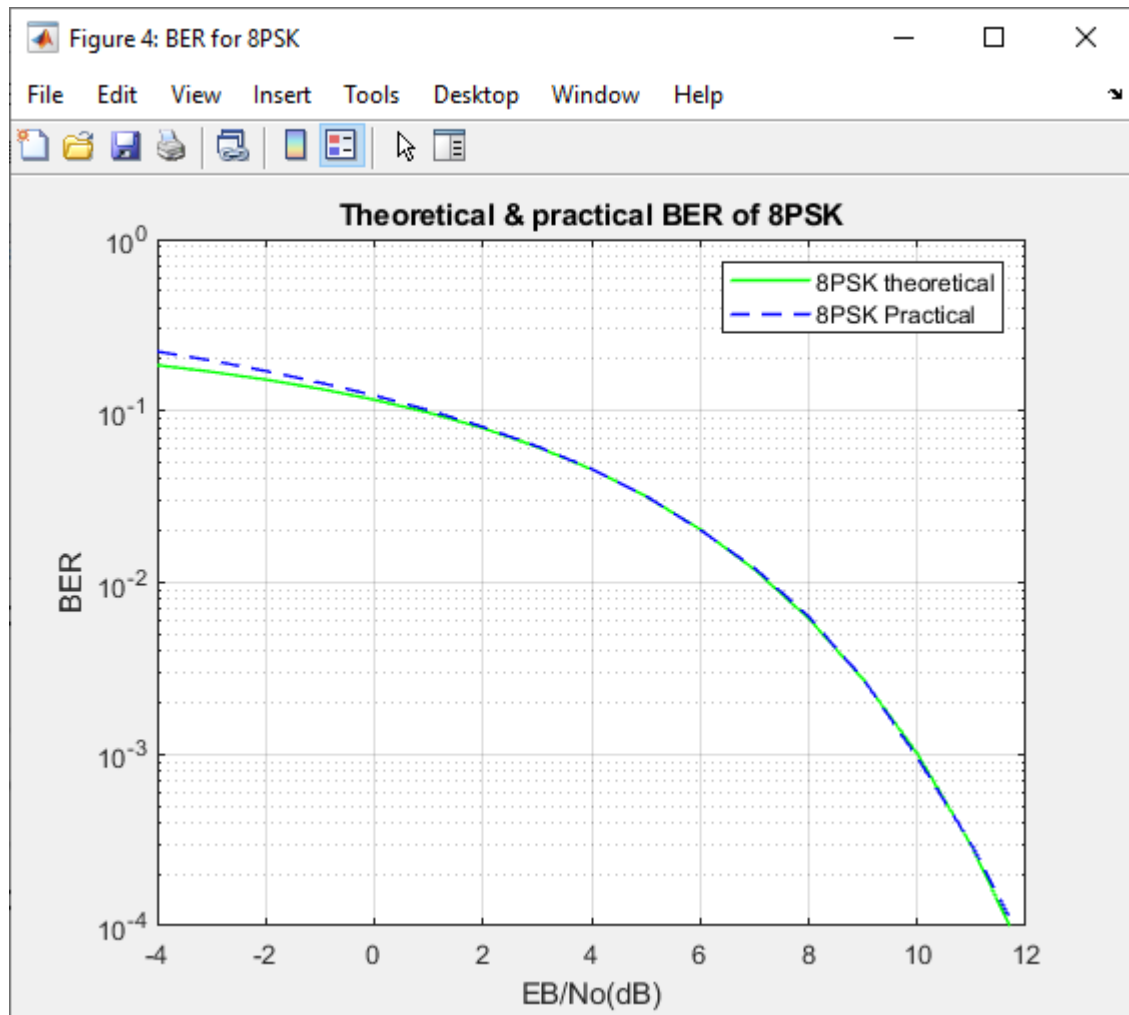


Figure 4: Theoretical and practical BER vs  $E_b/N_0$  for 8PSK

**Comment:** As we can see they are almost the same and they become more similar when the number of binary data used increases.

- **1.5. Theoretical vs practical BER graph of the 16QAM:**

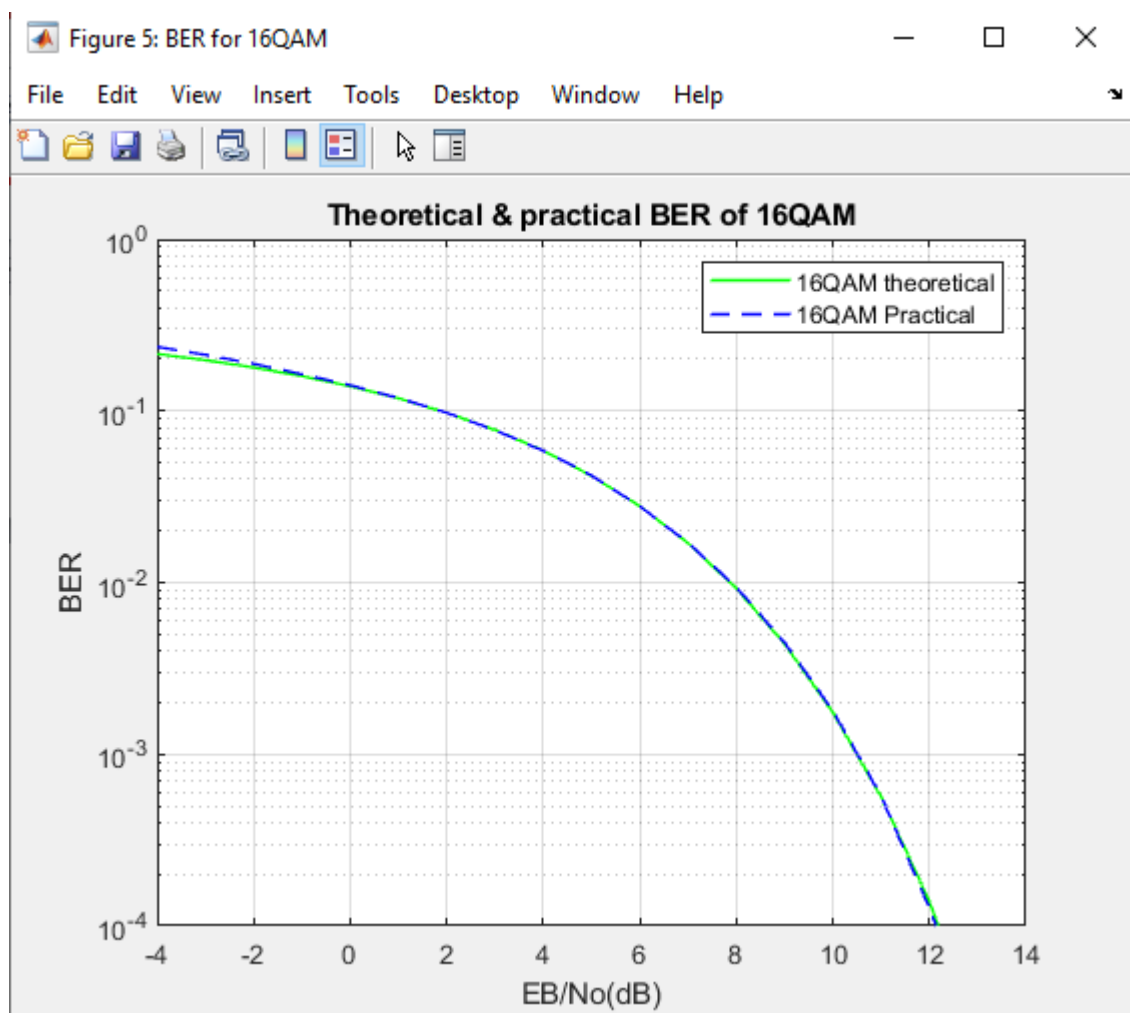


Figure 5: Theoretical and practical BER vs  $E_b/N_0$  for 16QAM

**Comment:** As we can see they are almost the same.

- **1.6. Theoretical and practical BER graph for all the four modulation schemes:**

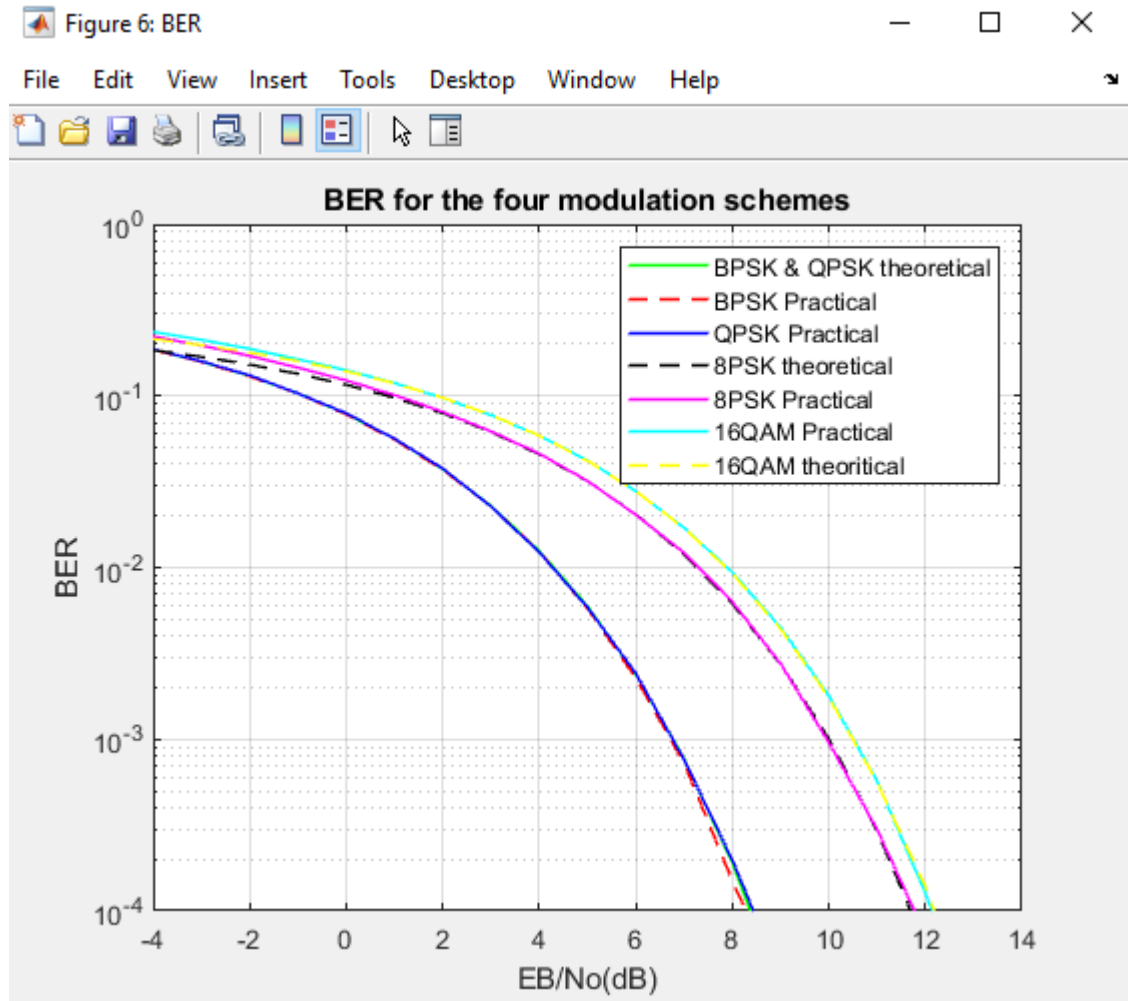


Figure 6: Theoretical and practical BER vs  $EB/No$  for all four modulation schemes

**Comment:**

- 1) The power of the signal in case of 16QAM must be higher than that of BPSK by 8.5dB to have the same BER for both cases.
- 2) As we can see the BER of the BPSK and QPSK are the same and also their practical BER are also the same as there will be a change for one bit at a time from one symbol to another one beside it.



- 3) The BER of the BPSK and QPSK is smaller than that of 16QAM and 8PSK for the same energy per bit so they need more energy to have the same BER.
- 4) In case of 16QAM, we used more bits than the previous types of modulation and also used grey coding for the used bits resulting in 16 symbols being used, so this increases the BER as the region for each symbols decreases significantly.

- **1.7. Theoretical and practical BER graph for QPSK Gray-Encoded and Not-Gray-Encoded:**

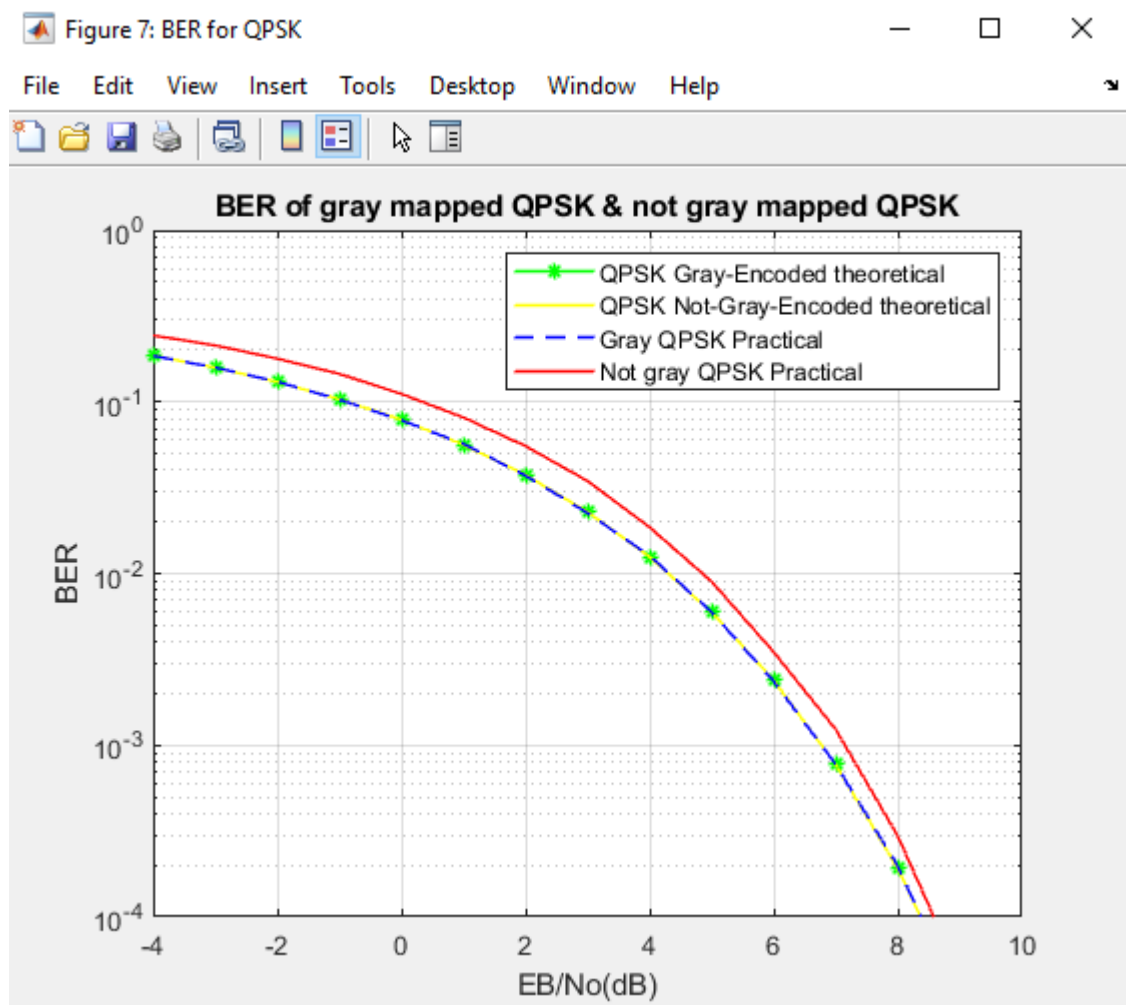


Figure 7: Theoretical and practical BER vs EB/No for QPSK Gray-Encoded and Not-Gray-Encoded

### **Comment:**

As we can see the BER of QPSK Not-Grey-Encoded is above the QPSK Grey-Encoded with a slightly larger value indicating there is more error as in QPSK Not-Grey-Encoded there will be two bits changes in some cases when we go from one symbol to the other symbol beside it, while in the QPSK Grey-Encoded there will be only one bit changes at a time when we go from one symbol to the other symbol beside it, so we concluded that QPSK Gray-Encoding decreases the BER.

## **2. BFSK:**

- **2.1. This is the basis functions of the signal set:**

$$\phi_i(t) = \sqrt{\frac{2}{T_b}} * \cos(2\pi f_i t)$$

$$\text{where } T_b \text{ frequency is } f_i = \frac{n_c + i}{T_b}, i = 1, 2$$

***This is the 2 basis functions of the signal set:***

$$\phi_1(t) = \sqrt{\frac{2}{T_b}} * \cos(2\pi f_1 t) ,$$

$$\phi_2(t) = \sqrt{\frac{2}{T_b}} * \cos(2\pi f_2 t)$$

- **2.2. An expression for the baseband equivalent signals for this set, indicating the carrier frequency used:**

$$S(t) = \sqrt{\frac{2E_b}{T_b}} * \cos(2\pi f_1 t) \quad \text{or} \quad \sqrt{\frac{2E_b}{T_b}} * \cos(2\pi f_2 t)$$

**The carrier frequency used is  $f_c = f_1$**

$$S_1(t) = \sqrt{\frac{2E_b}{T_b}} * \cos(2\pi f_c t)$$

$$S_2(t) = \sqrt{\frac{2E_b}{T_b}} * \cos(2\pi(f_c + \Delta f)t)$$

$$S_2(t) = \sqrt{\frac{2E_b}{T_b}} * (\cos(2\pi f_c t) * \cos(2\pi \Delta f t) - \sin(2\pi f_c t) * \sin(2\pi \Delta f t))$$

**The two baseband equivalent signals used for this set:**

$$S_{1BB}(t) = \sqrt{\frac{2E_b}{T_b}} \quad , \text{ If the bit transmitted is } 0$$

$$S_{2BB}(t) = \sqrt{\frac{2E_b}{T_b}} (\cos(2\pi \Delta f t) + j * \sin(2\pi \Delta f t)) \quad ,$$

*If the bit transmitted is 1*

- **2.3. Theoretical vs practical BER graph of the BFSK:**

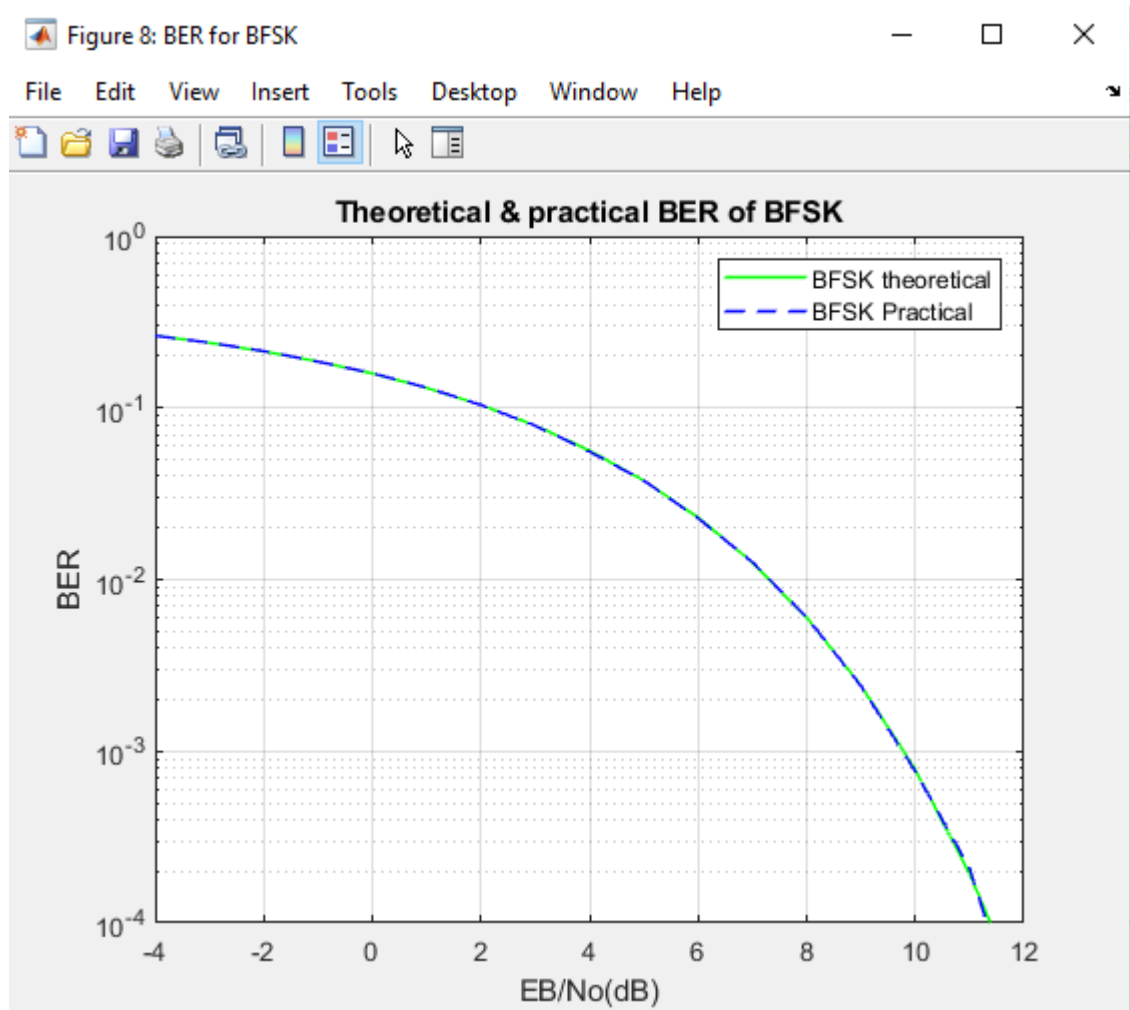


Figure 8: Theoretical and practical BER vs  $E_b/N_0$  for BFSK

**Comment:** As we can see they are almost the same and they become more similar when the number of binary data used increases, also the BER of BFSK is good.

- **2.4. PSD graph of BFSK using the baseband equivalent signal:**

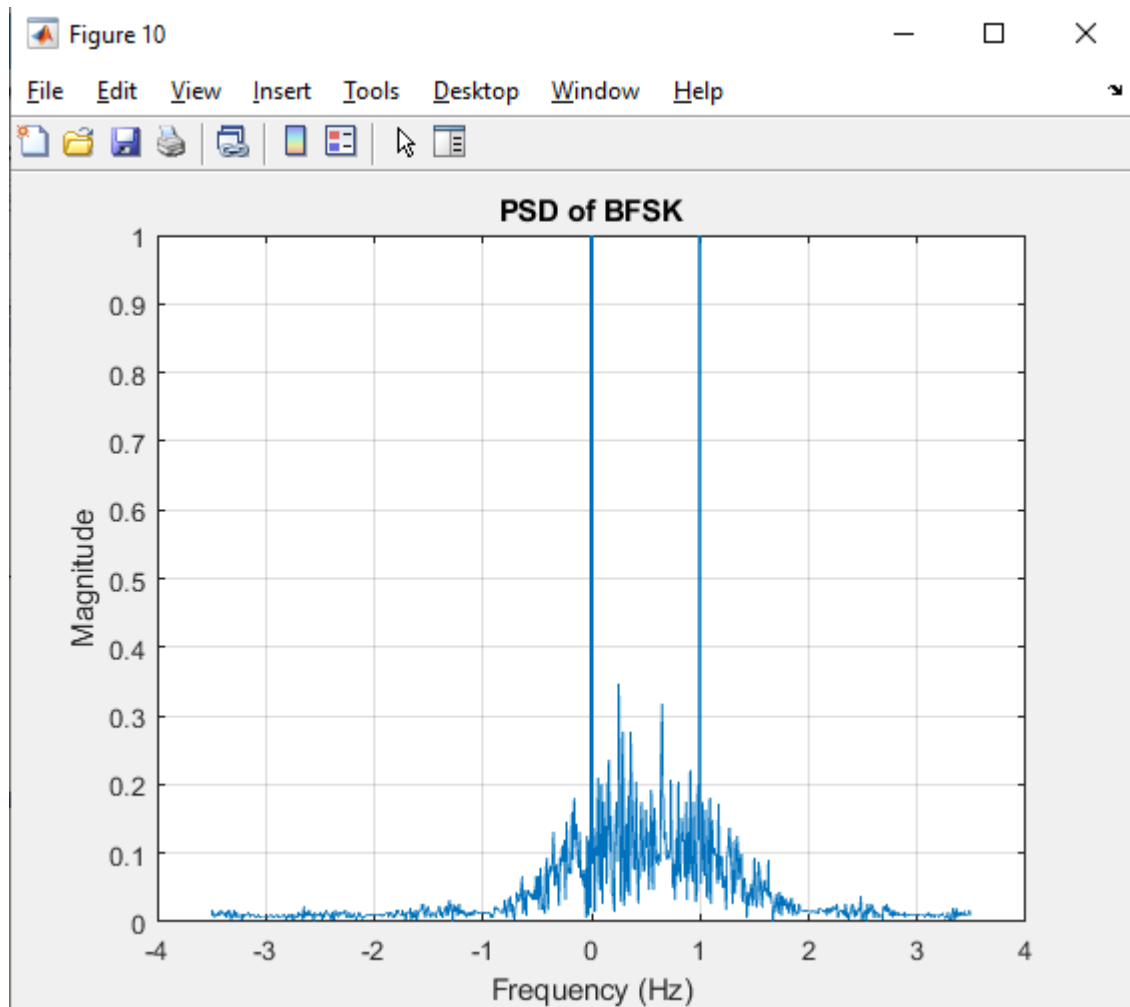


Figure 9: PSD of BFSK

### **3. The MATLAB Code:**

```

4. clc;
5. close all;
6. clear all;
7. Number_of_bits = 48*10000;
8. SNR_dB= -4:1:14;
9. % Generate a random bits
10.     Original_bits = randi([0, 1], 1, Number_of_bits);
11.

```

```

12.      %Calculating No considering Eb(bit energy) = 1 for all modulation
schemes
13.      No = zeros(size(SNR_dB));
14.      for x = 1:length(SNR_dB)
15.          No(x) = 10^(-SNR_dB(x)/10);
16.      end
17.
18.      %% BPSK
19.
20.      % mapper
21.      BPSK_mapped = (2*Original_bits)-1;
22.
23.      % Chanel
24.
25.      BPSK_AWGN = (randn(1,Number_of_bits))./sqrt(2);
26.      BPSK_BER = zeros(size(SNR_dB));
27.
28.      for x = 1:length(SNR_dB)
29.
30.          recieved_BPSK = BPSK_mapped + sqrt(No(x))*BPSK_AWGN;
31.
32.          % demapper
33.
34.          BPSK_demapped = zeros(1,Number_of_bits);
35.
36.          for k = 1:Number_of_bits
37.              if (recieved_BPSK(k) < 0)
38.                  BPSK_demapped(k) = 0;
39.              else
40.                  BPSK_demapped(k) = 1;
41.              end
42.
43.              % BPSK practical BER
44.
45.              if (BPSK_demapped(k)~=Original_bits(k))
46.                  BPSK_BER(x)=BPSK_BER(x)+1;
47.              end
48.          end
49.      end
50.      %Theoretical BER of BPSK
51.      BPSK_BER = BPSK_BER./Number_of_bits;
52.      BPSK_theoretical_BER = 0.5*erfc(sqrt(10.^(SNR_dB/10)));
53.
54.      %plotting BER of BPSK theoretical vs practical
55.      figure('Name' , 'BER for BPSK');
56.      semilogy(SNR_dB , BPSK_theoretical_BER,'g','linewidth',1);
57.      hold on;
58.      semilogy(SNR_dB , BPSK_BER,'b --','linewidth',1);
59.      hold off;

```

```

60.     title('Theoretical & practical BER of BPSK');
61.     xlabel('EB/No(dB)'); ylabel('BER'); grid on;
62.     legend('BPSK theoretical','BPSK
    Practical','Location','NorthEast')
63.     ylim([10^-4, 10^0]);
64.     grid on;
65.
66.     %% QPSK & 8PSK
67.
68.     % Convert the numbers to strings
69.     str_bits = arrayfun(@num2str, Original_bits, 'UniformOutput',
    false);
70.
71.     % Concatenate adjacent strings
72.     QPSK_binary_bits_str = strcat(str_bits(1:2:end),
    str_bits(2:2:end));
73.     M8PSK_binary_bits_str = strcat(str_bits(1:3:end),
    str_bits(2:3:end), str_bits(3:3:end));
74.
75.     % Convert the binary number to its decimal equivalent
76.     QPSK_bits_in_decimal = bin2dec(QPSK_binary_bits_str);
77.     QPSK_bits_in_binary = dec2bin(QPSK_bits_in_decimal);
78.     M8PSK_bits_in_decimal = bin2dec(M8PSK_binary_bits_str);
79.     M8PSK_bits_in_binary = dec2bin(M8PSK_bits_in_decimal);
80.
81.     % Shift the decimal number to the right
82.     QPSK_shifted_bits_in_decimal = bitshift(QPSK_bits_in_decimal, -
    1); % Shift one position to the right
83.     M8PSK_shifted_bits_in_decimal = bitshift(M8PSK_bits_in_decimal, -
    1); % Shift one position to the right
84.
85.     % Perform XOR operation
86.     QPSK_decimal_gray_encoded = bitxor(QPSK_bits_in_decimal,
    QPSK_shifted_bits_in_decimal);
87.     M8PSK_decimal_gray_encoded = bitxor(M8PSK_bits_in_decimal,
    M8PSK_shifted_bits_in_decimal);
88.
89.     % Converting to binary
90.     QPSK_binary_gray_encoded = dec2bin(QPSK_decimal_gray_encoded);
91.     M8PSK_binary_gray_encoded = dec2bin(M8PSK_decimal_gray_encoded);
92.
93.     % mapping gray QPSK
94.
95.     QPSK_gray_mapped = zeros(1,Number_of_bits/2);
96.     QPSK_gray_energy_summation = 0;
97.     for k = 1:Number_of_bits/2
98.         if(QPSK_binary_gray_encoded(k,:) == '00')
99.             QPSK_gray_mapped(k) = -1-1i;
100.        elseif(QPSK_binary_gray_encoded(k,:) == '01')
101.            QPSK_gray_mapped(k) = -1+1i;

```

```

102.         elseif(QPSK_binary_gray_encoded(k,:) == '11')
103.             QPSK_gray_mapped(k) = 1+1i;
104.         elseif(QPSK_binary_gray_encoded(k,:) == '10')
105.             QPSK_gray_mapped(k) = 1-1i;
106.         end
107.         QPSK_gray_energy_summation = QPSK_gray_energy_summation +
abs(QPSK_gray_mapped(k))^2;
108.     end
109.     Eavg_QPSK_gray = QPSK_gray_energy_summation/(Number_of_bits/2);
110.
111.     % mapping not gray QPSK
112.
113.     QPSK_not_gray_mapped = zeros(1,Number_of_bits/2);
114.     QPSK_not_gray_energy_summation = 0;
115.     for k = 1:Number_of_bits/2
116.         if(QPSK_bits_in_binary(k,:) == '00')
117.             QPSK_not_gray_mapped(k) = -1-1i;
118.         elseif(QPSK_bits_in_binary(k,:) == '01')
119.             QPSK_not_gray_mapped(k) = -1+1i;
120.         elseif(QPSK_bits_in_binary(k,:) == '11')
121.             QPSK_not_gray_mapped(k) = 1-1i;
122.         elseif(QPSK_bits_in_binary(k,:) == '10')
123.             QPSK_not_gray_mapped(k) = 1+1i;
124.         end
125.         QPSK_not_gray_energy_summation =
QPSK_not_gray_energy_summation + abs(QPSK_not_gray_mapped(k))^2;
126.     end
127.     Eavg_QPSK_not_gray =
QPSK_not_gray_energy_summation/(Number_of_bits/2);
128.
129.     % mapping 8PSK
130.
131.     M8PSK_mapped = zeros(1,Number_of_bits/3);
132.     M8PSK_energy_summation = 0;
133.     for k = 1:(Number_of_bits/3)
134.         if(M8PSK_binary_gray_encoded(k,:) == '000')
135.             M8PSK_mapped(k) = 1;
136.         elseif(M8PSK_binary_gray_encoded(k,:) == '001')
137.             M8PSK_mapped(k) = 0.7071+0.7071i;
138.         elseif(M8PSK_binary_gray_encoded(k,:) == '011')
139.             M8PSK_mapped(k) = 1i;
140.         elseif(M8PSK_binary_gray_encoded(k,:) == '111')
141.             M8PSK_mapped(k) = -0.7071-0.7071i;
142.         elseif(M8PSK_binary_gray_encoded(k,:) == '101')
143.             M8PSK_mapped(k) = -1i;
144.         elseif(M8PSK_binary_gray_encoded(k,:) == '100')
145.             M8PSK_mapped(k) = 0.7071-0.7071i;
146.         elseif(M8PSK_binary_gray_encoded(k,:) == '110')
147.             M8PSK_mapped(k) = -1;
148.         elseif(M8PSK_binary_gray_encoded(k,:) == '010')

```



```

149.         M8PSK_mapped(k) = -0.7071+0.7071i;
150.     end
151.     M8PSK_energy_summation = M8PSK_energy_summation +
        abs(M8PSK_mapped(k))^2;
152. end
153. Eavg_M8PSK = M8PSK_energy_summation/(Number_of_bits/3);
154.
155. % Initializeing BER vectors
156. QPSK_gray_BER = zeros(size(SNR_dB));
157. QPSK_not_gray_BER = zeros(size(SNR_dB));
158. QAM16_BER = zeros(size(SNR_dB));
159. M8PSK_BER = zeros(size(SNR_dB));
160.
161. for i = 1:length(SNR_dB)
162.
163.     % QPSK gray channel
164.     VARI_QPSK_gray = sqrt((No(i)*Eavg_QPSK_gray)/(2*2));
165.     real_noise_QPSK_gray = VARI_QPSK_gray.* randn(1 ,
        Number_of_bits/2);
166.     img_noise_QPSK_gray = VARI_QPSK_gray .* randn(1 ,
        Number_of_bits/2);
167.     recieved_QPSK_Gray = QPSK_gray_mapped +
        (real_noise_QPSK_gray+(1i*img_noise_QPSK_gray));
168.
169.     % QPSK gray demapper
170.     QPSK_gray_demapped = zeros(1,Number_of_bits/2);
171.
172.     for k=1:Number_of_bits/2
173.
174.         im = imag(recieved_QPSK_Gray(k));
175.         re = real(recieved_QPSK_Gray(k));
176.         if((re<=0) && (im<=0))
177.             QPSK_gray_demapped(k) = 0;
178.         elseif((re<=0) && (im>=0))
179.             QPSK_gray_demapped(k) = 1;
180.         elseif((re>=0) && (im<=0))
181.             QPSK_gray_demapped(k) = 2;
182.         elseif((re>=0) && (im>=0))
183.             QPSK_gray_demapped(k) = 3;
184.         end
185.     end
186.
187.     QPSK_gray_demapped = dec2bin(QPSK_gray_demapped);
188.
189.     % gray QPSK BER
190.
191.     for k=1:Number_of_bits/2
192.         if
193.             ~((QPSK_binary_gray_encoded(k,1)==QPSK_gray_demapped(k,1)))
                QPSK_gray_BER(i) = QPSK_gray_BER(i) + 1;

```

```

194.         end
195.         if
196.             ~((QPSK_binary_gray_encoded(k,2)==QPSK_gray_demapped(k,2)))
197.             QPSK_gray_BER(i) = QPSK_gray_BER(i) + 1;
198.         end
199.     end
200.     % QPSK not gray channel
201.     VARI_QPSK_not_gray = sqrt((No(i)*Eavg_QPSK_not_gray)/(2*2));
202.     real_noise_QPSK_not_gray = VARI_QPSK_not_gray.* randn(1 ,
        Number_of_bits/2);
203.     img_noise_QPSK_not_gray = VARI_QPSK_not_gray .* randn(1 ,
        Number_of_bits/2);
204.     recieved_QPSK_not_gray = QPSK_not_gray_mapped +
        (real_noise_QPSK_not_gray +(1i*img_noise_QPSK_not_gray));
205.
206.     % QPSK_not_gray demapper
207.     QPSK_not_gray_demapped = zeros(1,Number_of_bits/2);
208.
209.     for k=1:Number_of_bits/2
210.
211.         im = imag(recieved_QPSK_not_gray(k));
212.         re = real(recieved_QPSK_not_gray(k));
213.         if (re<0)
214.             if(im<0)
215.                 QPSK_not_gray_demapped(k) = 0;
216.             else
217.                 QPSK_not_gray_demapped(k) = 1;
218.             end
219.         else
220.             if(im<0)
221.                 QPSK_not_gray_demapped(k) = 3;
222.             else
223.                 QPSK_not_gray_demapped(k) = 2;
224.             end
225.         end
226.     end
227.     QPSK_not_gray_demapped = dec2bin(QPSK_not_gray_demapped);
228.
229.     % not gray QPSK BER
230.
231.     for k=1:Number_of_bits/2
232.         if
233.             ~((QPSK_bits_in_binary(k,1)==QPSK_not_gray_demapped(k,1)))
234.             QPSK_not_gray_BER(i) = QPSK_not_gray_BER(i) + 1;
235.         end
236.         if
237.             ~((QPSK_bits_in_binary(k,2)==QPSK_not_gray_demapped(k,2)))
238.             QPSK_not_gray_BER(i) = QPSK_not_gray_BER(i) + 1;
239.         end
240.     end

```

```

238.         end
239.
240.         % 8PSK channel
241.
242.         VARI_M8PSK = sqrt((No(i)*Eavg_M8PSK)/(2*3));
243.         real_noise_M8PSK = VARI_M8PSK.* randn(1 , Number_of_bits/3);
244.         img_noise_M8PSK = VARI_M8PSK .* randn(1 , Number_of_bits/3);
245.         recieved_M8PSK_Gray = M8PSK_mapped + (real_noise_M8PSK +
        (1i*img_noise_M8PSK));
246.
247.         % M8PSK demapper
248.
249.         M8PSK_demapped = zeros(1,Number_of_bits/3);
250.
251.         for k=1:Number_of_bits/3
252.
253.             ang= angle(recieved_M8PSK_Gray(k));
254.             if (ang>2.7489)
255.                 M8PSK_demapped(k) = 6;
256.             elseif (ang>1.9635)
257.                 M8PSK_demapped(k) = 2;
258.             elseif (ang>1.1781)
259.                 M8PSK_demapped(k) = 3;
260.             elseif (ang>0.3927)
261.                 M8PSK_demapped(k) = 1;
262.             elseif (ang>-0.3927)
263.                 M8PSK_demapped(k) = 0;
264.             elseif (ang>-1.1781)
265.                 M8PSK_demapped(k) = 4;
266.             elseif (ang>-1.9635)
267.                 M8PSK_demapped(k) = 5;
268.             elseif (ang>-2.7489)
269.                 M8PSK_demapped(k) = 7;
270.             else
271.                 M8PSK_demapped(k) = 6;
272.             end
273.         end
274.         M8PSK_demapped = dec2bin(M8PSK_demapped);
275.
276.         % 8PSK practical BER
277.
278.         for k=1:Number_of_bits/3
279.             if
                ~((M8PSK_binary_gray_encoded(k,1)==M8PSK_demapped(k,1)))
280.                 M8PSK_BER(i) = M8PSK_BER(i) + 1;
281.             end
282.             if
                ~((M8PSK_binary_gray_encoded(k,2)==M8PSK_demapped(k,2)))
283.                 M8PSK_BER(i) = M8PSK_BER(i) + 1;

```

```

284.         end
285.         if
~((M8PSK_binary_gray_encoded(k,3)==M8PSK_demapped(k,3)))
286.             M8PSK_BER(i) = M8PSK_BER(i) + 1;
287.         end
288.     end
289.
290. end
291. %Theoretical BER of QPSK Gray-Encoded
292. QPSK_gray_BER = QPSK_gray_BER./Number_of_bits;
293. QPSK_gray_theoretical_BER = 0.5*erfc(sqrt(10.^(SNR_dB/10)));
294. %plotting BER of QPSK Gray-Encoded theoretical vs practical
295. figure('Name' , 'BER for QPSK Gray-Encoded ');
296. semilogy(SNR_dB , QPSK_gray_theoretical_BER,'g','linewidth',1);
297. hold on;
298. semilogy(SNR_dB , QPSK_gray_BER,'b --','linewidth',1);
299. hold off;
300. title('Theoretical & practical BER of QPSK Gray-Encoded ');
301. xlabel('EB/No(dB)'); ylabel('BER'); grid on;
302. legend('QPSK Gray-Encoded theoretical','QPSK Gray-Encoded
Practical','Location','NorthEast')
303. ylim([10^-4, 10^0]);
304. grid on;
305.
306.
307.
308. %Theoretical BER of QPSK Not-Gray-Encoded
309. QPSK_not_gray_BER = QPSK_not_gray_BER./Number_of_bits;
310. QPSK_not_gray_theoretical_BER = 0.5*erfc(sqrt(10.^(SNR_dB/10)));
311. %plotting BER of QPSK Not-Gray-Encoded theoretical vs practical
312. figure('Name' , 'BER for QPSK Not-Gray-Encoded ');
313. semilogy(SNR_dB ,
QPSK_not_gray_theoretical_BER,'g','linewidth',1);
314. hold on;
315. semilogy(SNR_dB , QPSK_not_gray_BER,'b --','linewidth',1);
316. hold off;
317. title('Theoretical & practical BER of QPSK Not-Gray-Encoded ');
318. xlabel('EB/No(dB)'); ylabel('BER'); grid on;
319. legend('QPSK Not-Gray-Encoded theoretical','QPSK Not-Gray-
Encoded Practical','Location','NorthEast')
320. ylim([10^-4, 10^0]);
321. grid on;
322.
323.
324.
325.
326. %Theoretical BER of 8PSK
327. M8PSK_BER = M8PSK_BER./Number_of_bits;
328. M8PSK_theoretical_BER =
(1/3)*erfc(sqrt(3*10.^(SNR_dB/10))*sin(22/(8*7)));

```

```

329.    %plotting BER of 8PSK therotical vs practical
330.    figure('Name' , 'BER for 8PSK');
331.    semilogy(SNR_dB , M8PSK_theoretical_BER,'g','linewidth',1);
332.    hold on;
333.    semilogy(SNR_dB , M8PSK_BER,'b --','linewidth',1);
334.    hold off;
335.    title('Theoretical & practical BER of 8PSK');
336.    xlabel('EB/No(dB)'); ylabel('BER'); grid on;
337.    legend('8PSK theoretical','8PSK
    Practical','Location','NorthEast')
338.    ylim([10^-4, 10^0]);
339.    grid on;
340.
341.    %% 16QAM
342.
343.    % Mapper
344.
345.    QAM16_binary_bits_str = strcat(str_bits(1:4:end),
    str_bits(2:4:end), str_bits(3:4:end), str_bits(4:4:end));
346.    QAM16_bits_in_decimal = bin2dec(QAM16_binary_bits_str);
347.    QAM16_shifted_bits_in_decimal = bitshift(QAM16_bits_in_decimal, -
    1); % Shift one position to the right
348.    QAM16_decimal_gray_encoded = bitxor(QAM16_bits_in_decimal,
    QAM16_shifted_bits_in_decimal);
349.    QAM16_binary_gray_encoded = dec2bin(QAM16_decimal_gray_encoded);
350.
351.    QAM16_mapped = zeros(1,Number_of_bits/4);
352.    QAM16_energy_summation = 0;
353.
354.    for k = 1:Number_of_bits/4
355.
356.        if(QAM16_binary_gray_encoded(k,:) == '0000')
357.            QAM16_mapped(k) = -3-3i;
358.        elseif(QAM16_binary_gray_encoded(k,:) == '0001')
359.            QAM16_mapped(k) = -3-1i;
360.        elseif(QAM16_binary_gray_encoded(k,:) == '0010')
361.            QAM16_mapped(k) = -3+3i;
362.        elseif(QAM16_binary_gray_encoded(k,:) == '0011')
363.            QAM16_mapped(k) = -3+1i;
364.        elseif(QAM16_binary_gray_encoded(k,:) == '0100')
365.            QAM16_mapped(k) = -1-3i;
366.        elseif(QAM16_binary_gray_encoded(k,:) == '0101')
367.            QAM16_mapped(k) = -1-1i;
368.        elseif(QAM16_binary_gray_encoded(k,:) == '0110')
369.            QAM16_mapped(k) = -1+3i;
370.        elseif(QAM16_binary_gray_encoded(k,:) == '0111')
371.            QAM16_mapped(k) = -1+1i;
372.        elseif(QAM16_binary_gray_encoded(k,:) == '1000')
373.            QAM16_mapped(k) = 3-3i;
374.        elseif(QAM16_binary_gray_encoded(k,:) == '1001')

```

```

375.         QAM16_mapped(k) = 3-1i;
376.     elseif(QAM16_binary_gray_encoded(k,:) == '1010')
377.         QAM16_mapped(k) = 3+3i;
378.     elseif(QAM16_binary_gray_encoded(k,:) == '1011')
379.         QAM16_mapped(k) = 3+1i;
380.     elseif(QAM16_binary_gray_encoded(k,:) == '1100')
381.         QAM16_mapped(k) = 1-3i;
382.     elseif(QAM16_binary_gray_encoded(k,:) == '1101')
383.         QAM16_mapped(k) = 1-1i;
384.     elseif(QAM16_binary_gray_encoded(k,:) == '1110')
385.         QAM16_mapped(k) = 1+3i;
386.     elseif(QAM16_binary_gray_encoded(k,:) == '1111')
387.         QAM16_mapped(k) = 1+1i;
388.     end
389.     QAM16_energy_summation = QAM16_energy_summation +
    abs(QAM16_mapped(k))^2;
390. end
391. QAM16_energy_avg = QAM16_energy_summation/(Number_of_bits/4);
392.
393. Eavg_16QAM = QAM16_energy_avg;
394.
395. for i = 1:length(SNR_dB)
396.
397.     % QAM16 Channel
398.
399.     VARI_16QAM = sqrt((No(i)/2)*(Eavg_16QAM/4));
400.     real_noise_16QAM = VARI_16QAM .* randn(1 , Number_of_bits/4);
401.     img_noise_16QAM = VARI_16QAM .* randn(1 , Number_of_bits/4);
402.     received_16QAM = QAM16_mapped + (real_noise_16QAM +
    (1i*img_noise_16QAM));
403.
404.     %%% QAM16 demapper
405.
406.     QAM16_demapped = zeros(1,Number_of_bits/4);
407.
408.     for k = 1:Number_of_bits/4
409.
410.         im = imag(received_16QAM(k));
411.         re = real(received_16QAM(k));
412.         if(re<=-2)
413.             if(im<=-2)
414.                 QAM16_demapped(k) = 0;
415.             elseif (im<=0)
416.                 QAM16_demapped(k) = 1;
417.             elseif (im<=2)
418.                 QAM16_demapped(k) = 3;
419.             else
420.                 QAM16_demapped(k) = 2;
421.             end

```

```

422.         elseif(re<=0)
423.             if(im<=-2)
424.                 QAM16_demapped(k) = 4;
425.             elseif (im<=0)
426.                 QAM16_demapped(k) = 5;
427.             elseif (im<=2)
428.                 QAM16_demapped(k) = 7;
429.             else
430.                 QAM16_demapped(k) = 6;
431.             end
432.         elseif (re<=2)
433.             if(im<=-2)
434.                 QAM16_demapped(k) = 12;
435.             elseif (im<=0)
436.                 QAM16_demapped(k) = 13;
437.             elseif (im<=2)
438.                 QAM16_demapped(k) = 15;
439.             else
440.                 QAM16_demapped(k) = 14;
441.             end
442.         else
443.             if(im<=-2)
444.                 QAM16_demapped(k) = 8;
445.             elseif (im<=0)
446.                 QAM16_demapped(k) = 9;
447.             elseif (im<=2)
448.                 QAM16_demapped(k) = 11;
449.             else
450.                 QAM16_demapped(k) = 10;
451.             end
452.         end
453.     end
454.     QAM16_demapped = dec2bin(QAM16_demapped);
455.
456.     % QAM16 practical BER
457.
458.     for k=1:Number_of_bits/4
459.
460.         if
461.             ~((QAM16_binary_gray_encoded(k,1)==QAM16_demapped(k,1)))
462.                 QAM16_BER(i) = QAM16_BER(i) + 1;
463.             end
464.         if
465.             ~((QAM16_binary_gray_encoded(k,2)==QAM16_demapped(k,2)))
466.                 QAM16_BER(i) = QAM16_BER(i) + 1;
467.             end
468.         if
469.             ~((QAM16_binary_gray_encoded(k,3)==QAM16_demapped(k,3)))
470.                 QAM16_BER(i) = QAM16_BER(i) + 1;
471.             end
472.         if
473.             ~((QAM16_binary_gray_encoded(k,4)==QAM16_demapped(k,4)))
474.                 QAM16_BER(i) = QAM16_BER(i) + 1;
475.             end
476.     end

```

```

469.         if
470.             ~((QAM16_binary_gray_encoded(k,4)==QAM16_demapped(k,4)))
471.                 QAM16_BER(i) = QAM16_BER(i) + 1;
472.             end
473.         end
474.     end
475.     %Theoretical BER of 16QAM
476.     QAM16_BER = QAM16_BER./Number_of_bits;
477.     QAM16_theoretical_BER =
478.         (3/8)*erfc(sqrt(10.^(SNR_dB/10))/sqrt(2.5));
479.     %plotting BER of 16QAM theoretical vs practical
480.     figure('Name' , 'BER for 16QAM');
481.     semilogy(SNR_dB , QAM16_theoretical_BER,'g','linewidth',1);
482.     hold on;
483.     semilogy(SNR_dB , QAM16_BER,'b --','linewidth',1);
484.     hold off;
485.     title('Theoretical & practical BER of 16QAM');
486.     xlabel('EB/No(dB)'); ylabel('BER'); grid on;
487.     legend('16QAM theoretical','16QAM
488.         Practical','Location','NorthEast')
489.     ylim([10^-4, 10^0]);
490.     grid on;
491.
492.     %%
493.     % plotting
494.     figure('Name' , 'BER');
495.     semilogy(SNR_dB , BPSK_theoretical_BER,'g','linewidth',1);
496.     hold on;
497.     semilogy(SNR_dB , BPSK_BER,'r --','linewidth',1);
498.     hold on;
499.     semilogy(SNR_dB , QPSK_gray_BER,'b','linewidth',1);
500.     hold on;
501.     semilogy(SNR_dB , M8PSK_theoretical_BER,'k --','linewidth',1);
502.     hold on;
503.     semilogy(SNR_dB , M8PSK_BER,'m','linewidth',1);
504.     hold on;
505.     semilogy(SNR_dB , QAM16_BER,'c','linewidth',1);
506.     hold on;
507.     semilogy(SNR_dB , QAM16_theoretical_BER,'y --','linewidth',1);
508.     hold off;
509.     title('BER for the four modulation schemes');
510.     xlabel('EB/No(dB)'); ylabel('BER'); grid on;
511.     legend('BPSK & QPSK theoretical','BPSK Practical','QPSK
512.         Practical','8PSK theoretical','8PSK Practical','16QAM Practical','16QAM
513.         theoritical','Location','NorthEast')
514.     ylim([10^-4, 10^0]);
515.     grid on;

```



```

514.
515.     figure('Name' , 'BER for QPSK');
516.     semilogy(SNR_dB , QPSK_gray_theoretical_BER,'g-*','linewidth',1);
517.     hold on;
518.     semilogy(SNR_dB ,
519.         QPSK_not_gray_theoretical_BER,'y','linewidth',1);
520.     hold on;
521.     semilogy(SNR_dB , QPSK_gray_BER,'b --','linewidth',1);
522.     hold on;
523.     semilogy(SNR_dB , QPSK_not_gray_BER,'r','linewidth',1);
524.     hold off;
525.     title('BER of gray mapped QPSK & not gray mapped QPSK');
526.     xlabel('Eb/No(dB)'); ylabel('BER'); grid on;
527.     legend('QPSK Gray-Encoded theoretical','QPSK Not-Gray-Encoded
528.         theoretical','Gray QPSK Practical','Not gray QPSK
529.         Practical','Location','NorthEast')
530.     ylim([10^-4, 10^0]);
531.     grid on;
532.
533. %% BFSK
534.
535. % mapping
536.
537. BFSK_mapped = zeros(1,Number_of_bits);
538.
539. for k=1:Number_of_bits
540.
541.     if(Original_bits(k) == 0)
542.         BFSK_mapped(k) = 1;
543.     else
544.         BFSK_mapped(k) = 1i;
545.     end
546. end
547.
548. BFSK_BER = zeros(size(SNR_dB));
549.
550. for i = 1:length(SNR_dB)
551.
552.     % BFSK channel
553.
554.     VARI_BFSK = sqrt(No(i)/2);
555.     real_noise_BFSK = VARI_BFSK .* randn(1 , Number_of_bits);
556.     img_noise_BFSK = VARI_BFSK .* randn(1 , Number_of_bits);
557.     received_BFSK = BFSK_mapped + (real_noise_BFSK +
558.         (1i*img_noise_BFSK));
559.
560.
561. % BFSK demapper
562.
563. BFSK_demapped = zeros(1,Number_of_bits);

```

```

559.
560.     for k=1:Number_of_bits
561.
562.         im = imag(received_BFSK(k));
563.         re = real(received_BFSK(k));
564.         if(re>im)
565.             BFSK_demapped(1,k) = 0;
566.         else
567.             BFSK_demapped(1,k) = 1;
568.         end
569.     end
570.
571.     % BFSK practical BER
572.
573.     for k=1:Number_of_bits
574.
575.         if ~(BFSK_demapped(1,k)==Original_bits(1,k))
576.             BFSK_BER(i) = BFSK_BER(i) + 1;
577.         end
578.     end
579. end
580. %Theoretical BER of BFSK
581. BFSK_BER = BFSK_BER./Number_of_bits;
582. BFSK_theoretical_BER = 0.5*erfc(sqrt(10.^((SNR_dB/10))/2));
583.
584. %plotting BER of BFSK theoretical vs practical
585.
586. figure('Name' , 'BER for BFSK');
587. semilogy(SNR_dB , BFSK_theoretical_BER,'g','linewidth',1);
588. hold on;
589. semilogy(SNR_dB , BFSK_BER,'b --','linewidth',1);
590. hold off;
591. title('Theoretical & practical BER of BFSK');
592. xlabel('Eb/No(dB)'); ylabel('BER'); grid on;
593. legend('BFSK theoretical','BFSK
    Practical','Location','NorthEast')
594. ylim([10^-4, 10^0]);
595. grid on;
596.
597.
598. %%
599. Number_of_waveforms = 500;
600. Number_of_bits = 100;
601.
602. % Generating an ensemble consists of 500 realization , each 101
    bits
603. Data = randi([0 1], Number_of_waveforms, Number_of_bits+1) ;
604.
605. %Activating the DAC every 10ms

```

```

606.     Transmitter_ensemble = zeros( Number_of_waveforms ,
    (Number_of_bits+1)*7 );
607.
608.     bit_duration = 0.07;
609.     signal_mag = sqrt( 2*1/bit_duration ) ; %
610.     time = 0.01 : 0.01 : 0.07;
611.
612.     S_Baseband = signal_mag * cos(2*pi*(1/bit_duration) * time) + 1i
    * signal_mag * sin( 2*pi*(1/bit_duration) * time) ;
613.
614.     %construct the ensemble
615.     lag = 1;
616.     for k = 1 : Number_of_bits+1
617.         for i = 1 : Number_of_bits+1
618.             if (Data(k , i) ~= 0)
619.                 Transmitter_ensemble(k , lag) = S_Baseband(1);
    Transmitter_ensemble(k , lag+1) = S_Baseband(2); Transmitter_ensemble(k
    , lag+2) = S_Baseband(3); Transmitter_ensemble(k , lag+3) =
    S_Baseband(4);
620.                 Transmitter_ensemble(k , lag+4) = S_Baseband(5);
    Transmitter_ensemble(k , lag+5) =S_Baseband(6) ; Transmitter_ensemble(k
    , lag+6) = S_Baseband(7);
621.
622.                 elseif (Data(k , i) == 0)
623.                     Transmitter_ensemble(k , lag) = signal_mag;
    Transmitter_ensemble(k , lag+1) = signal_mag; Transmitter_ensemble(k ,
    lag+2) = signal_mag; Transmitter_ensemble(k , lag+3) = signal_mag;
624.                     Transmitter_ensemble(k , lag+4) = signal_mag;
    Transmitter_ensemble(k , lag+5) = signal_mag; Transmitter_ensemble(k ,
    lag+6) = signal_mag;
625.                 end
626.                 lag = lag + 7;
627.             end
628.             lag = 1;
629.         end
630.
631.     %delay
632.     delayed_for_BFSK_ensemble = randi ([0 6], Number_of_waveforms, 1)
    ; %delay array for polar_nrz
633.     BFSK = zeros (Number_of_waveforms, Number_of_bits*7) ; %
    initialization
634.
635.     % adding the delay
636.     for i = 1:Number_of_waveforms
637.         BFSK(i, :) = Transmitter_ensemble(i,
    delayed_for_BFSK_ensemble(i)+1 : (Number_of_bits*7 +
    delayed_for_BFSK_ensemble(i)));
638.     end
639.
640.     % calculating the autocorrelation of the ensemble

```

```

641.     BFSK_avg_auto = zeros( 1 ,Number_of_bits*7); % initialization of
the autocorrelation matrix for polar_rz
642.
643.     for lag = 0 : (Number_of_bits*7)-1
644.         autocorrelation = 0;
645.
646.         for i = 1 : Number_of_waveforms
647.             autoCorrelation = conj(BFSK(i , 1))*BFSK(i, lag + 1);
648.             autocorrelation = autocorrelation + autoCorrelation;
649.
650.         end
651.         BFSK_avg_auto(lag+1) = autocorrelation/500;
652.     end
653.
654.
655.     PSD = abs(fftshift(fft(BFSK_avg_auto)));
656.
657.     fs = 100;
658.     frequency_axis = -fs/2 : fs/700 : fs/2 - fs/700;
659.
660.     figure;
661.     plot(frequency_axis*bit_duration, PSD/(fs*2));
662.     title("PSD of BFSK");
663.     xlabel("Frequency (Hz)");
664.     ylabel("Magnitute");
665.     ylim([0,1]);
666.     grid on;

```