DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATICS

Digital Telecommunications

EXERCISE 2

Import

This work studies the transmission of information in digital form. form through additive white Gaussian noise communication channels **(AWGN).** The information is represented in analog signal waveforms information, meaning the channels are basically analog.

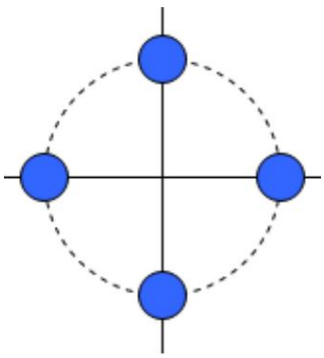
To do this study we compare *4-PSK* and *4-FSK* modulations as towards their performance with the help of the bit error rate **(BER: Bit Error Rate)** which is performed in equivalent bandpass systems with the use of rectangular pulse.

PSK Features

Suppose we have a set of M two-dimensional waveforms of the same type *sm(t)* with *m = 1,2,3…M .* Thus we can create a set of M bandpass waveforms of the form:

$$u_m(t) = s_m(t)\cos 2\pi f_c t$$

If the *M* two-dimensional bandpass waveforms have the same energy then the corresponding points of the signal geometrically represent a circle with radius ÿÿs as shown below:



From this geometric representation for *M* = 4, we observe that the points of the signal are equivalent to a single signal, with the difference that the phase of the signal slides along $\frac{}{4}$ ·

In other words, the bandpass signal is of the form

$$( ) 2 + \qquad \frac{\quad}{2}, \qquad = 1,2, \ldots 4$$

and has the same geometric representation as a set of M = 4 orthogonals signals. We therefore conclude that a simple way to create M bandpass signals with the same energy is to capture and modulate the information in the carrier phase.

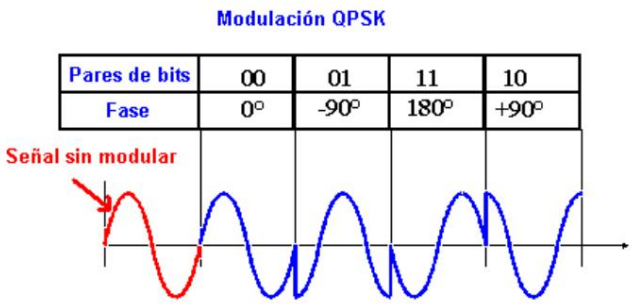
The general representation of an M set of carrier-phase modulated modules waveforms is as follows:

$$( ) = ( ) \cos(2 \qquad 2) + \frac{\quad}{\quad}, \qquad = 0.1, \ldots \quad \ddot{y}\ 1 \quad {}^{\ddot{y}}\ 0\ \ddot{y}\ \ddot{y}$$

with *gT(t)* representing the baseband pulse for the modulation, which determines the spectral characteristics of my signal to be transmitted. If *gT(t)* is a rectangular pulse of the form $( ) = \ddot{y}\ 2 \frac{\quad}{\quad}$, 0 ÿ ÿ then the corresponding transmitted signal waveforms become:

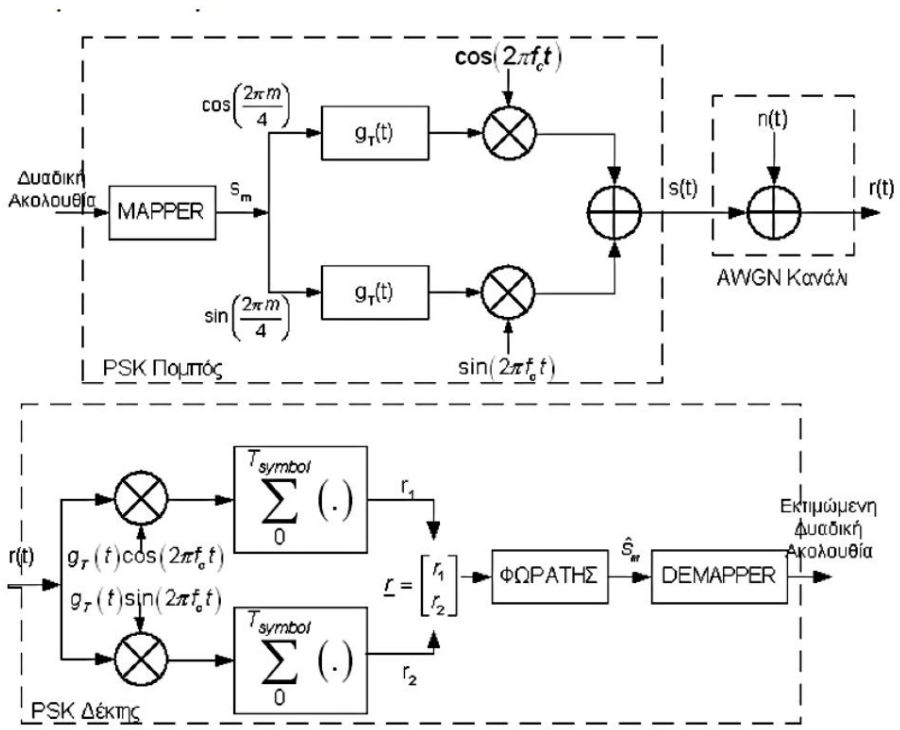
$$( ) = \ddot{y}\ 2 \frac{\quad}{\quad} \cos(2 \qquad 2) + \frac{\quad}{\quad}, \qquad = 0.1, \ldots \quad \ddot{y}\ 1 \quad {}^{\ddot{y}}\ 0\ \ddot{y}\ \ddot{y}$$

In this way they have a fixed envelope and the carrier phase changes abruptly at the beginning of each signal interval. This type of digital modulation is called Phase Shift Keying **(PSK).**



Example of a 4-PSK

In the figure below we describe a PSK **(Quadrature Phase Shift)** rectangle.
Keying (QPSK)) 4 phases:



PSK circuit

The above figure illustrates the design of a PSK circuit.
More specifically, we divided the PSK function into 4 sub-functions. For
For the purposes of this exercise, the functions are described with the help of
MATLAB in the following individual files:

1. **Binary_input.m:** Initially the binary sequence is generated, which is passed to the mapper and the mapping to symbols is done.

2. **Mapper.m:** The mapper has the ability to encode in Gray so that to achieve the mapping of symbols in the low-dimensional space signals into sequences of bits that differ slightly.

3. **Modulator.m:** The output of the mapper in turn feeds the modulator that modulates each component, i.e. multiplies it by the rectangular pulse and modulation around the carrier frequency to obtain the desired bandpass signal described by function described in the previous section.

4. **Demodulator.m:** In turn, the receiver receives the signal and demodulates with the demodulator. To do this, the first thing the receiver to know the phase of the carrier and the time frames of each symbol, or else be synchronized with the transmitter. In addition correlates the received signal with the two components of the carrier, and This results in a vector *r* with two values, which is also the estimated value of the current symbol on the M-ary PSK constellation.

5. **Decision Device.m:** Then the receiver receives the vector *r* from the demodulator and ends up at the symbol that is closest. The vector which will be less than *r* also corresponds to sent symbol.

6. **Demapper.m:** Finally, the mapping is done to the estimated send binary sequence.

Note: It should be noted that between the modulator – demodulator there is also the addition of white Gaussian noise to the signal received by the demodulator. The process of adding noise is done in the file **noise.m.**

FSK Features

In a *4-bit FSK* we can on the other hand transmit a block of *log2M* bits in each signal waveform, and represent these waveforms as follows

$$( ) = ÿ \sqrt{\quad} \cos(2 + 2 \quad), \qquad = 0.1, \ldots 3$$

where

- *ÿs = kEb* (symbol energy)
- *T = 4Tb* (symbol duration)
- *ÿf* (frequency distance between two consecutive frequencies)

Similarly in *M-FSK* the waveforms have the same energy *Es.* H frequency distance *ÿf* determines the way we distinguish between M transmitted signals.

M-ary rectangular FSK waveforms have the representation M M-dimensional vectors of the form

$$s_1 = (\sqrt{E_s}, 0, 0, ..., 0)$$
$$s_2 = (0, 0, 0, ..., \sqrt{E_s})$$
$$...$$
$$s_M = (0, 0, 0, ..., \sqrt{E_s})$$

where the basis functions are $( ) = ÿ (\bar{+})$. Two consecutive vectors have a distance equivalent to the minimum distance of M signals, equal to $\overline{ÿ}$.

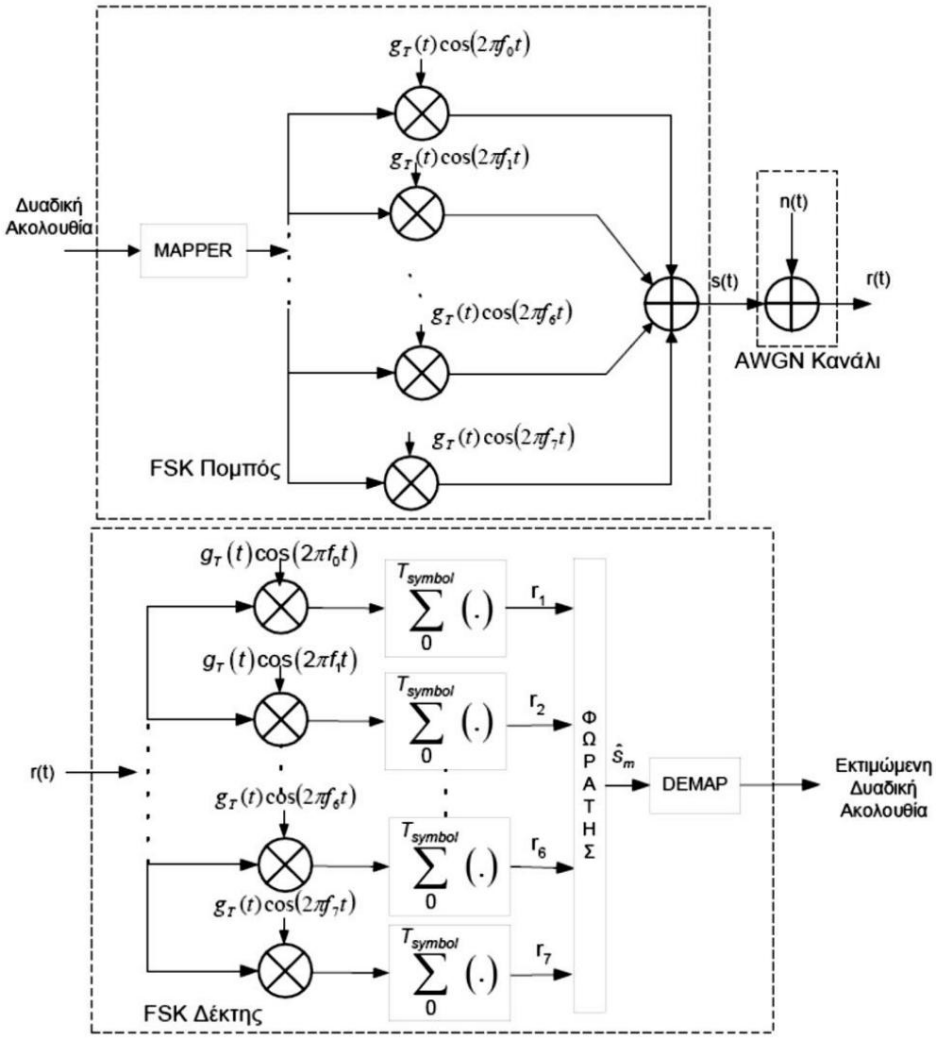
Demodulation of FSK signals and Forasi

The demodulation and decoding of M-ary FSK signals that transmitted through the AWGN channel. It can be achieved by estimating M of slips with phase *ÿm* and perform demodulation and coherent illumination

phase, where the received signal *r(t)* is correlated with each of the M possible signals (+ +) , = 0.1, . .

FSK circuit

The above figure illustrates the design of a PSK circuit.
More specifically, we divided the PSK function into 4 sub-functions.



For the purposes of this exercise, the functions are described using of MATLAB in the following individual files:

1. **Binary_input.m:** Initially the binary sequence is generated, which is passed to the mapper and the mapping to symbols is done.

2. Mapper.m: The mapper has the ability to encode in Gray so that to achieve the mapping of symbols in the two-dimensional space signals into sequences of bits that differ slightly.

3. Modulator.m: The output of the mapper in turn feeds the modulator that modulates each component in the following way. The 4-PSK system uses the following 4 signals for each symbol

$$( ) = ( ) \cos(2 (+)) , \quad = \frac{1}{\quad}, \quad = 0, . .3$$

4. Demodulator.m: In turn, the receiver receives the signal and demodulates with the demodulator.

5. Decision Device.m: Then the receiver receives the vector *r* from the demodulator and ends up at the symbol that is closest. The vector which will be less than *r* also corresponds to sent symbol.

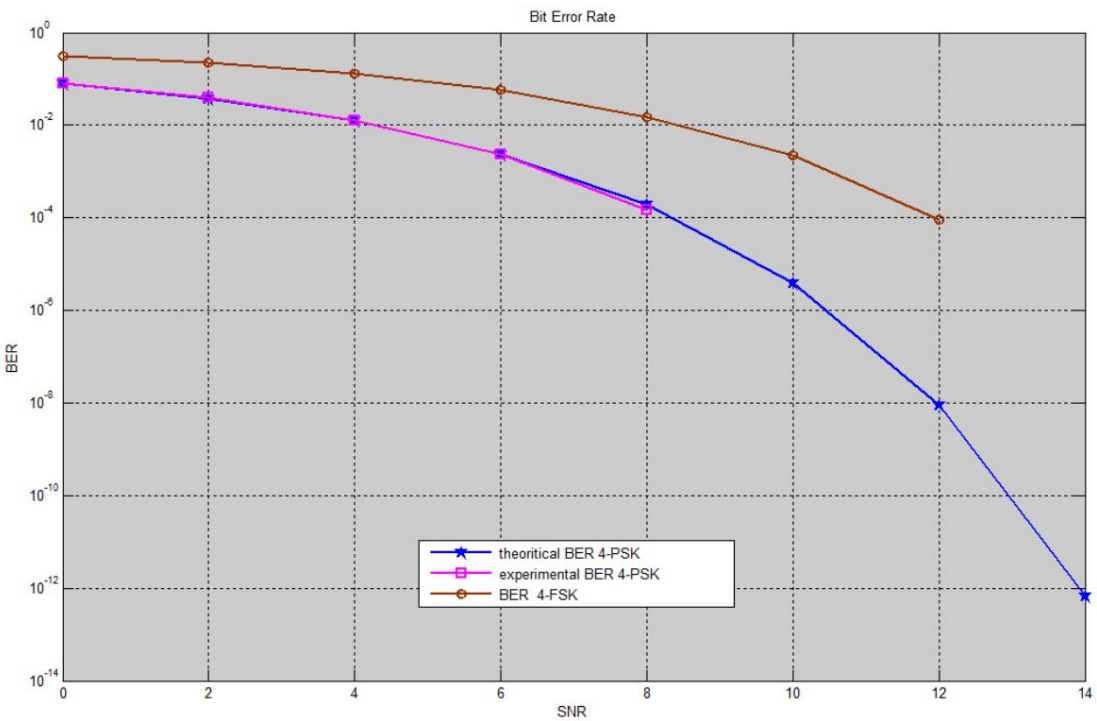
6. Demapper.m: Finally, the mapping is done to the estimated send binary sequence.

Note: It should be noted that between the modulator – demodulator there is also the addition of white Gaussian noise to the signal received by the demodulator. The process of adding noise is done in the file **noise.m.**

Results

Using a few lines of code to create a representation of what was described above we capture in a graphic representation of the results of our study on the **bit error rate,** i.e. the bit error probability refers to the number of bits sent incorrectly to the total number of bits sent.

The measurements were made for *SNR = [0: 2: 12] dB* and for data of the order of 105 bits for greater reliability in the results, as they offer reliability in BER measurements and they exceed our needs.



We observe that for 4-bit PSK the experimental BER "fades" faster from the practical point of view, while the same is true for 4-bit FSK. In general, it is possible to It takes more SNR to "erase" the bit error rate for the theoretical part.

MATLAB codes

binary_input.m

```
function [ binary_sequence ] = binary_input( num_el )
% BINARY_SEQUENCE = BINARY_INPUT( NUM_EL ) %

% INPUT
% NUMEL: number of elements (0, 1) that will be exported in the binary_sequence array

% OUTPUT
% BINARY_SEQUENCE: the binary_sequence array


% initialize the binary_sequence with 0 or 1 as elements, that will have the

% same probability binary_sequence
= randsrc(num_el, 1, [0,1]); end
```

mapper.m

```
function symbols_array = mapper(bin_seq, encoding, gray)
% SYMBOLS_ARRAY = MAPPER(BIN_SEQ, ENCODING, GRAY): %

% INPUT

% BIN_SEQ: argument a binary sequence
% ENCODING: encoding <FSK> or <PSK>
% GRAY: denotes if is to be used gray (1) encoding or not (0)
% OUTPUT
% SYMBOLS_ARRAY: the elements of the transformation into symbols

% the length of input seq_size =
length(bin_seq);

% we group the bits into 2 groups. The remainder of the %sequence is separately converted into one symbol at
% the end

temp = mod(seq_size, 2);

% the sequence which is divisible by 2 div_seq = bin_seq(1 : (seq_size -
temp), :);

% grouping of that sequence reshaped_sequence =
reshape(div_seq, 2, (seq_size - temp) / 2);

% tranform the sequence into binary code for every group of 2 bits for i = 1: (seq_size - temp) / 2

        symbols_array(i) = bin2dec(num2str(reshaped_sequence(:, i)'));
end

% the rest of the bits are separately transformed into a symbol in binary % code if temp ~= 0



        symbols_array(i + 1) = bin2dec(num2str(bin_seq(seq_size - temp + 1 :seq_size, 1)')); end



% if we use gray encoding in order to achieve smaller distance between two

% symbols which are adjacent, we encode the symbols into Gray by using the % following function bin2gray if gray == 1



        symbols_array = bin2gray(symbols_array, encoding, 8);
end

end
```

modulator.m

```
function s_m = modulator(symbols_array, encoding)
% S_M = MODULATOR(SYMBOLS_ARRAY, ENCODING)
%
% INPUT
% SYMBOLS_ARRAY: symbols array that are to be transmitted
% ENCODING:                 encoding <FSK> or <PSK>
% OUTPUT
%S_M:                       the modulated signal

% size of the array that has the sequence converted into symbols
sym_array_size = length(symbols_array);

T_symbol = 40; % period of symbols
f_symbol = 1 / T_symbol; % frequency of symbols
T_sample = 1;                       % period of sample
T_c = 4; % period of ferousa
f_c = 1 / T_c; % frequency of ferousa
E_s = 1;                            % Energy per symbol

% orthogonal pulse
pulse = sqrt(2 *            E_s / T_symbol);

% initialization of the symbols that we send s_m = zeros(sym_array_size,
T_symbol / T_sample);

% computation of the transmitted signal
if encoding == 'PSK'
        for i = 1: sym_array_size
                for t = 1: T_symbol/T_sample
                        s_m(i, t) = pulse * cos( 2*pi*f_c*t -
2*pi*symbols_array(i)/4);
                end
        end
elseif encoding == 'FSK'
        for i = 1: sym_array_size
                for t = 1: T_symbol/T_sample
                        s_m(i, t) = pulse * cos(2 * pi * (f_c + symbols_array(i) * f_symbol) * t);

                end
        end
end

end
```

noise.m

```
function received_signal = noise(s_m, SNR)
% RECEIVED_SIGNAL = NOISE(S_M, SNR) %

% INPUT
% S_M: the modulated signal
% SNR: the SNR
% OUTPUT
% RECEIVED_SIGNAL: signal with AWGN

% given that
E_s = 1;
E_b = E_s / 3;

% we solve the equation % 10 * log_10(E_b /
N_0) = SNR % and we have as a result

N_0 = E_b / (10^(SNR/10));

% We create a gaussian distribution with mean value: m = 0; % and standard deviation


sigma = sqrt(N_0 / 2);

% the noise is added to every sample taken by the modulator % for that reason, the derived array has to have the same
dimensions as the

% array of the samples
 [L_symbol, T_symbol] = size(s_m);

% produce AWGN
noise = m + sigma * randn(L_symbol, T_symbol);

% adds it to the signal received_signal
= s_m + noise;

end
```

demodulator.m

```
function r = demodulator(received_signal, encoding)
% R = DEMODULATOR(RECEIVED_SIGNAL, ENCODING) %

% INPUT
% RECEIVED_SIGNAL: signal with AWGN
% ENCODING: encoding <FSK> or <PSK>
% OUTPUT
% R: components [r1,r2] of each transmitted signal



T_symbol = 40;                          % period of symbol %
f_symbol = 1 / T_symbol;                frequency of symbol % period of
T_sample = 1;                           sample % period of ferousa %
T_c = 4; f_c                            frequency of ferousa
= 1 / T_c;
E_s = 1;                                % Energy per symbol

% orthogonal pulse pulse =
sqrt(2 *                        E_s / T_symbol);

 [L_symbol, T_symbol] = size(received_signal);

% demodulation
if encoding == 'PSK' for t = 1:
        T_symbol y1(t, 1) = pulse *
                cos(2 * pi * f_c * t); y2(t, 1) = pulse * sin(2 * pi * f_c * t);

        end


        % calculation of the 2 components r = [received_signal *
        y1, received_signal * y2];
elseif encoding == 'FSK'
        for i = 1: 4 for t = 1:
                T_symbol y(i, t) = pulse * cos(2 * pi *
                        ( f_c + i * f_symbol) *
t);
                end
        end


        % calculation of the 4 components r = received_signal *
        y';
end

end
```

decision_device.m

```
function symbols = decision_device(r, encoding)
% SYMBOLS = DECISION_DEVICE(R, ENCODING) %

% INPUT
%R:                    components [r1,r2,..] of each transmitted
signal
% ENCODING:            encoding <FSK> or <PSK>
% OUTPUT
% SYMBOLS:             the binary (or gray) symbols that were to be sent

 [line_size, ~] = size(r);

if encoding == 'PSK' % calculates each
        possible received symbol for i = 1: 4 s(i, 1) = cos( 2 * pi * i / 4 ); s(i, 2) = sin( 2 * pi *
        i / 4 );



        end

        % calculates the symbol which presents the greatest probability
to
        % be the sent symbol for j =1: line_size
        for i = 1: 4

                        temp(i, 1) = norm([r(j,1), r(j,2)] - s(i,:));
                end
                 [min_diff, symbols(j, 1)] = min(temp);

        end

        % the 4th symbol is actually the 0th symbol symbols = mod(symbols,4);

elseif encoding == 'FSK'

        T_symbol = 40; % period of symbol f_symbol = 1 / T_symbol; % frequency of symbols


        tmp_symbol = zeros(1, line_size); % calculates the symbol which
        presents the greatest probability
to
        % be the sent symbol tmp_symbol = [0:
        3]; for j =1: line_size index =
        logical(round(abs(r(j, :)))); if
                isempty(tmp_symbol(index)) symbols(j) = 0; else symbols(j) =
                max( tmp_symbol(index) );



        end
end

% the 4th symbol is actually the 0th symbol symbols = mod(symbols,4);


end
```

demapper.m

```
function binary_sequence = demapper(symbols, encoding, gray)
% BINARY_SEQUENCE = DEMMAPER( SYMBOLS, ENCODING, GRAY ) %

% INPUT
% SYMBOLS:                received symbols from detector
% ENCODING: encoding <FSK> or <PSK>
% GRAY: denotes if is to be used gray (1) encoding or not (0)
% OUTPUT
%BINARY_SEQUENCE:              conversion of the received symbols into bits

% if there has been used Gray encoding in the transmitted signal if gray == 1

        symbols = gray2bin(symbols, encoding, 4);
end

binary_sequence = dec2bin(symbols);

% m: number of lines of the received bits matrix
% n: number of columns of the received bits matrix
 [lines, columns] = size(binary_sequence);

% reshape the matrix with the received bits to an array binary_sequence =
reshape(binary_sequence', lines*columns, 1);

% convert to double every character % in order to
recover the value that this character represents
ASCII code % we
substruct 30(hex) = 48(dec) % we have assumed that we deal
only with character wich represents digits % that is a valid hypothesis because we deal only with zero and one binary_sequence =

double(binary_sequence) - 48;


end
```

BER.m

```
function bit_error_rate = BER(input, output)
% BIT_ERROR_RATE = BER( INPUT, OUTPUT ) % calculate the Bit Error
Rate


% find the length of input array input_length = length(input);


% see if it can be divided by 2 modular = mod(input_length, 2);


% if not, that means that we send more bits than the length of input if modular ~= 0

        % calculate the redundant bits of the output further_elements_to_add = 2 - modular;


        % we concatenate at the end of the transmitted string the redundant

        % bits in order to compare the same amount of bits at both ends input(input_length + further_elements_to_add) = input(input_length);
        input(input_length) = 0; %if further_elements_to_add == 2 % input(input_length +
further_elements_to_add - 1) = 0; %end




end

% calculate the bit error rate bit_error_rate = sum(input ~=
output)/length(output);

end
```

script.m

```
clear; clc;
bits = 10^5;

count = 1;
x = [0: 2: 12];

for SNR = 0: 2: 12
        Pb(count, 1) = 5*erfc(sqrt( (10^SNR)/10 ));

        input_binary_sequence = binary_input(bits);
                                                = mapper(input_binary_sequence, 'FSK', symbols_array
0);
        s_m                                     = modulator(symbols_array , = noise(s_m ,SNR);   'FSK');
        received_signal
        r                                       = demodulator(received_signal, 'FSK');
        symbols                                 = decision_device(r, 'FSK');
        output_binary_sequence = demapper(symbols', 'FSK', 0);
        BER_fsk(count, 1)                       = BER(input_binary_sequence,
output_binary_sequence);

        input_binary_sequence = binary_input(bits);
        symbols_array = mapper(input_binary_sequence, 'PSK',
0);
        s_m                                     = modulator(symbols_array, 'PSK');
        received_signal                         = noise(s_m ,SNR);
        r                                       = demodulator(received_signal, 'PSK');
        symbols                                 = decision_device(r, 'PSK');
        output_binary_sequence = demapper(symbols', 'PSK', 0);
        BER_psk(count, 1)                       = BER(input_binary_sequence,
output_binary_sequence);

        count = count + 1;
end

semilogy(x', Pb, '.-');
hold on;
semilogy(x', BER_psk, 'g.-');
semilogy(x', BER_fsk, 'r.-');
legend(theoretical BER 4-PSK','experimental BER 4-PSK', BER 4-FSK');
title('Bit Rate Error');
xlabel('SNR');
ylabel('BER');
hold;

figure;
```