ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ψηφιακές Τηλεπικοινωνίες

1η Εργαστηριακή Άσκηση

Μονοπάτης Δημήτριος

ΑΜ:1040546

(Παλαιός ΑΜ:4776) - Επί πτυχίω

monopatis@ceid.upatras.gr

Επιβλέπων :

Κων/νος Μπερμπερίδης

13 Ιανουαρίου 2016

Εργαλεία ανάπτυξης

Έκδοση Matlab: R2016a 64bit

Λειτουργικό Σύστημα: Debian GNU/Linux 8

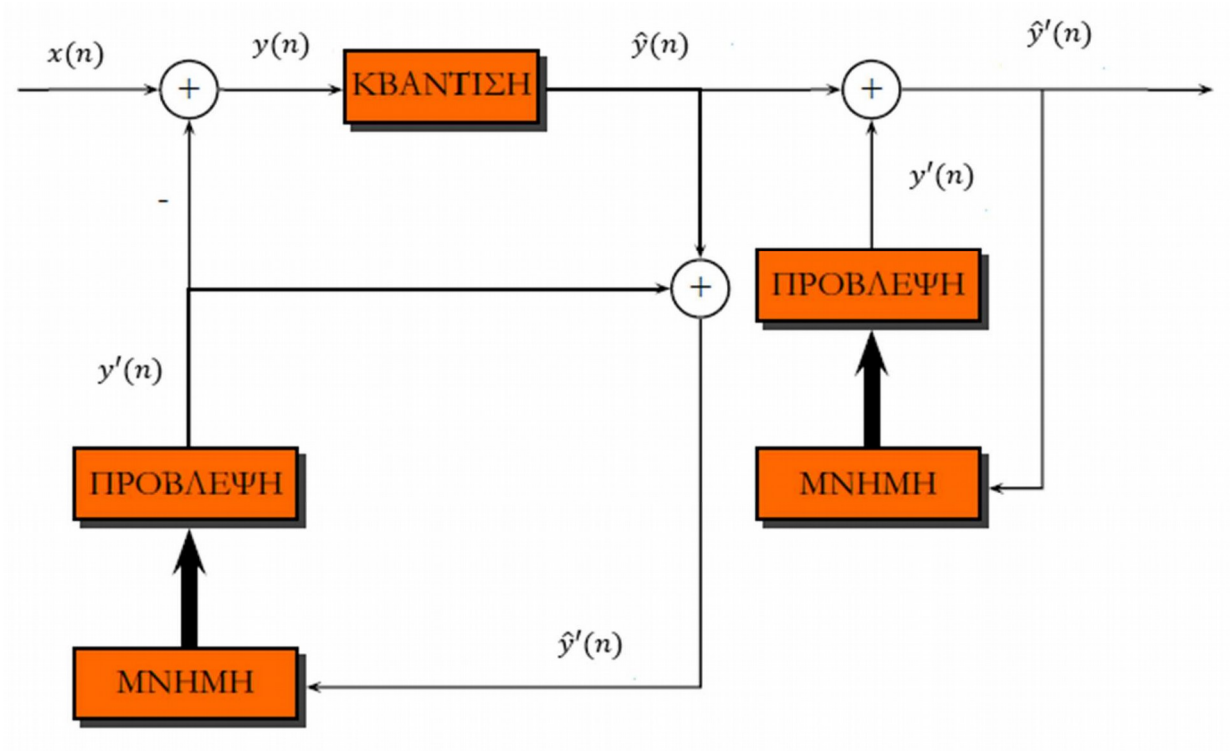
LibreOffice 4.3

Μέρος Α

Συμπίεση Διακριτής Πηγής με Χρήση της Κωδικοποίηση DPCM

Εισαγωγή

Η κωδικοποίηση DPCM (Differential Pulse Code Modulation) μπορεί να θεωρηθεί ως μια γενίκευση της κωδικοποίησης Δέλτα όπου το σήμα που κβαντίζεται και αποστέλλεται στο δέκτη, είναι η διαφορά ανάμεσα στο τρέχον δείγμα (της χρονικής στιγμής n) και σε μία γραμμική πρόβλεψή του. Δηλαδή, στην κωδικοποίηση DPCM, υπολογίζουμε, σε κάθε χρονική στιγμή, μια πρόβλεψη για την τιμή του τρέχοντος δείγματος με βάση τις τιμές προηγούμενων δειγμάτων τα οποία έχουν ήδη κωδικοποιηθεί και στη συνέχεια υπολογίζουμε το λάθος της πρόβλεψης αυτής. Το σήμα σφάλματος πρόβλεψης στη συνέχεια κωδικοποιείται χρησιμοποιώντας ένα ή περισσότερα δυαδικά ψηφία ανά δείγμα.



Κωδικοποιητής και Αποκωδικοποιητής DPCM

Μέρος Β

Προσομοίωση Ομόδυνου Ζωνοπερατού Συστήματος M-PSK

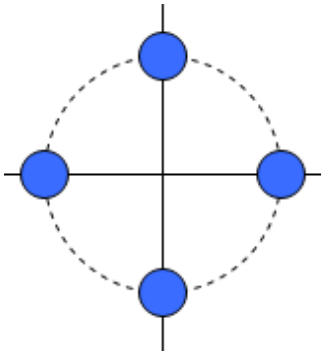
Στην παρούσα άσκηση μελετάμε την διαβίβαση της ψηφιακής πληροφορίας μέσα από κανάλια επικοινωνίας, τα οποία χαρακτηρίζονται ως κανάλια προσθετικού λευκού Gaussian θορύβου (Additive White Gaussian Noise - AWGN). Τα κανάλια αυτά είναι βασικά αναλογικά, με άλλα λόγια η προς μετάδοση ψηφιακή πληροφορία πρέπει να απεικονιστεί σε αναλογικές κυματομορφές σήματος πληροφορίας. Για την μελέτη αυτή, προχωρήσαμε στην σύγκριση των διαμορφώσεων 4-PSK και 8-PSK ως προς την απόδοση τους. Οι συγκρίσεις αυτές βασίστηκαν σε μετρήσεις πιθανότητας σφάλματος bit (Bit Rate Error (BER) που πραγματοποιήθηκαν σε ομόδυνα ζωνοπερατά συστήματα με χρήση ορθογώνιου παλμού.

Χαρακτηριστικά PSK

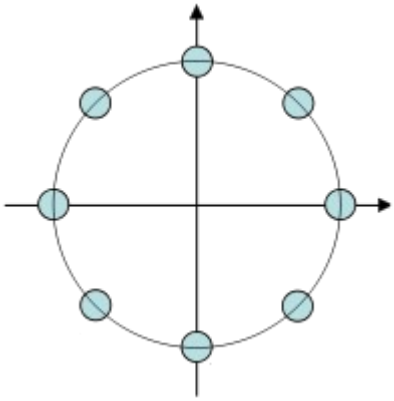
Έστω ότι έχουμε ένα σύνολο από M δισδιάστατες κυματομορφές σήματος $s_m(t)$ με $m = 1,2,3...M$. Έτσι μπορούμε να δημιουργήσουμε ένα σύνολο από M ζωνοπερατές κυματομορφές της μορφής:

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

Εάν οι M δισδιάστατες ζωνοπερατές κυματομορφές έχουν και την ίδια ενέργεια τότε τα αντίστοιχα σημεία του σήματος αναπαριστούν γεωμετρικά ένα κύκλο με ακτίνα $\sqrt{E_s}$ όπως φαίνεται παρακάτω:



Από την γεωμετρική αυτή αναπαράσταση για $M = 4$, παρατηρούμε ότι τα σημεία του σήματος είναι ισοδύναμα με ένα μόνο σήμα, με τη διαφορά ότι η φάση του σήματος ολισθαίνει κατά. $\frac{\pi}{2}$



Από την γεωμετρική αυτή αναπαράσταση για $M = 8$, παρατηρούμε ότι η φάση του σήματος ολισθαίνει κατά $\frac{\pi}{4}$.

Με άλλα λόγια, το ζωνοπερατό σήμα είναι της μορφής

$$s(t)\cos 2\pi f_c t+\frac{\pi m}{2}, m=1,2,\ldots,4 \text{ ή } m=1,2,\ldots 8$$

και έχει την ίδια γεωμετρική αναπαράσταση με ένα σύνολο M διορθογωνίων σημάτων. Συμπεραίνουμε λοιπόν πως ένας απλός τρόπος για την δημιουργία των

Μ ζωνοπερατών σημάτων με ίδια ενέργεια είναι να αποτυπωθεί και να διαμορφωθεί η πληροφορία στη φάση του φέροντος.

Η γενική αναπαράσταση ενός Μ συνόλου διαμορφωμένων κατά φάση φέροντος κυματομορφών είναι η εξής:

$$u_m(t)=g_T(t)\cos\left(2\pi f_c t\right)+\frac{2\pi m}{M}, m=0,1,..M-1 \text{ και } 0\le t\le T$$

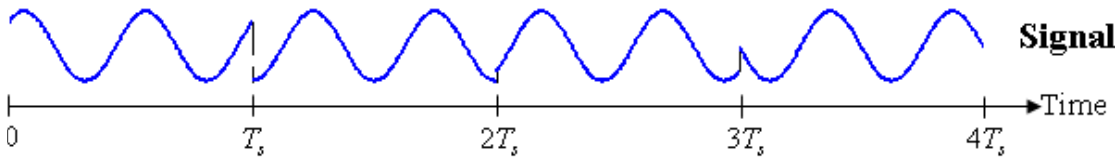
με $g_T(t)$ να αναπαριστά τον παλμό βασικής ζώνης για τη μορφοποίηση, που καθορίζει τα φασματικά χαρακτηριστικά μου προς μετάδοση σήματος.

Εάν ο $g_T(t)$ είναι ορθογώνιος παλμός της μορφής $g_t(T)=\sqrt{\frac{2E_s}{T}}, 0\le t\le T$ τότε οι

αντίστοιχες μεταδιδόμενες κυματομορφές σήματος γίνονται:

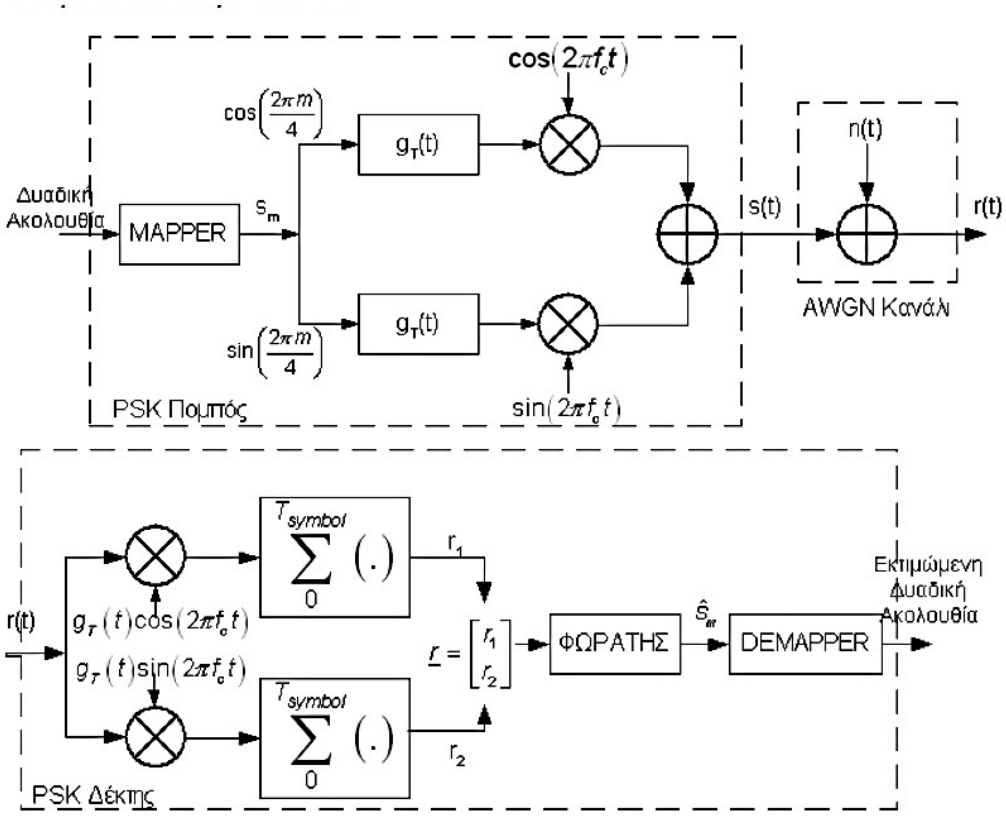
$$u_m(t)=\sqrt{\frac{2E_s}{T}}\cos\left(2\pi f_c t\right)+\frac{2\pi m}{M}, m=0,1,..M-1 \text{ και } 0\le t\le T \quad [1]$$

Με αυτό τον τρόπο έχουν σταθερή περιβάλλουσα και η φάση του φέροντος αλλάζει απότομα στην αρχή κάθε διαστήματος σήματος. Αυτός ο τύπος ψηφιακής διαμόρφωσης καλείται Phase Shift Keying (**PSK**).



Παράδειγμα ενός τετραδικού PSK σήματος με $T=\frac{2}{f_s}$

Στο παρακάτω σχήμα αναπαρίσταται ένα ορθογώνιο PSK (**Quadrature Phase Shift Keying (QPSK)**) 4 φάσεων:



Κύκλωμα PSK

Στο παραπάνω σχήμα απεικονίστηκε ο σχεδιασμός ενός PSK κυκλώματος. Πιο συγκεκριμένα χωρίσαμε την λειτουργία του PSK σε 4 επιμέρους λειτουργίες. Για τους σκοπούς αυτής της άσκησης, οι λειτουργίες περιγράφονται με την βοήθεια της MATLAB στα εξής επιμέρους αρχεία:

1. **binary_input.m**: Αρχικά παράγεται η δυαδική ακολουθία, η οποία διοχετεύεται στο mapper και γίνεται η αντιστοίχιση σε σύμβολα.

2. **mapper.m**: Ο mapper έχει την δυνατότητα να κωδικοποιεί κατά Gray ώστε να επιτυγχάνεται η αντιστοίχιση των συμβόλων στο δισδιάστατο χώρο σημάτων σε ακολουθίες από bits που διαφέρουν κατά λίγο.

3. **modulator.m**: Η έξοδος του mapper τροφοδοτεί με τη σειρά του τον modulator που διαμορφώνει την κάθε συνιστώσα, δηλαδή πολλαπλασιάζει με τον ορθογώνιο παλμό και την διαμορφώνει γύρω από τη φέρουσα συχνότητα ώστε να προκύψει το επιθυμητό ζωνοπερατό σήμα που περιγράφεται από την συνάρτηση [1].

4. **demodulator.m**: Με τη σειρά του ο δέκτης λαμβάνει το σήμα και το αποδιαμορφώνει με τον demodulator. Για να το κάνει, θα πρέπει πρώτα ο δέκτης να γνωρίζει τη φάση της φέρουσας και τα χρονικά πλαίσια καθενός συμβόλου, ή αλλιώς να είναι συγχρονισμένος με τον πομπό. Επιπλέον συσχετίζει το σήμα που έλαβε με τις δύο συνιστώσες της φέρουσας, και προκύπτει έτσι ένα διάνυσμα r με δύο τιμές, που αποτελεί και της εκτιμώμενη τιμή του τρέχοντος συμβόλου πάνω στον αστερισμό του M-αδικού PSK.

5. **foratis.m**: Στη συνέχεια ο φωρατής δέχεται το διάνυσμα r από τον αποδιαμορφωτή και καταλήγει στο πιο σύμβολο βρίσκεται εγγύτερα. Το διάνυσμα s_m που θα απέχει λιγότερο από το r αντιστοιχεί και στο αποστελλόμενο σύμβολο.

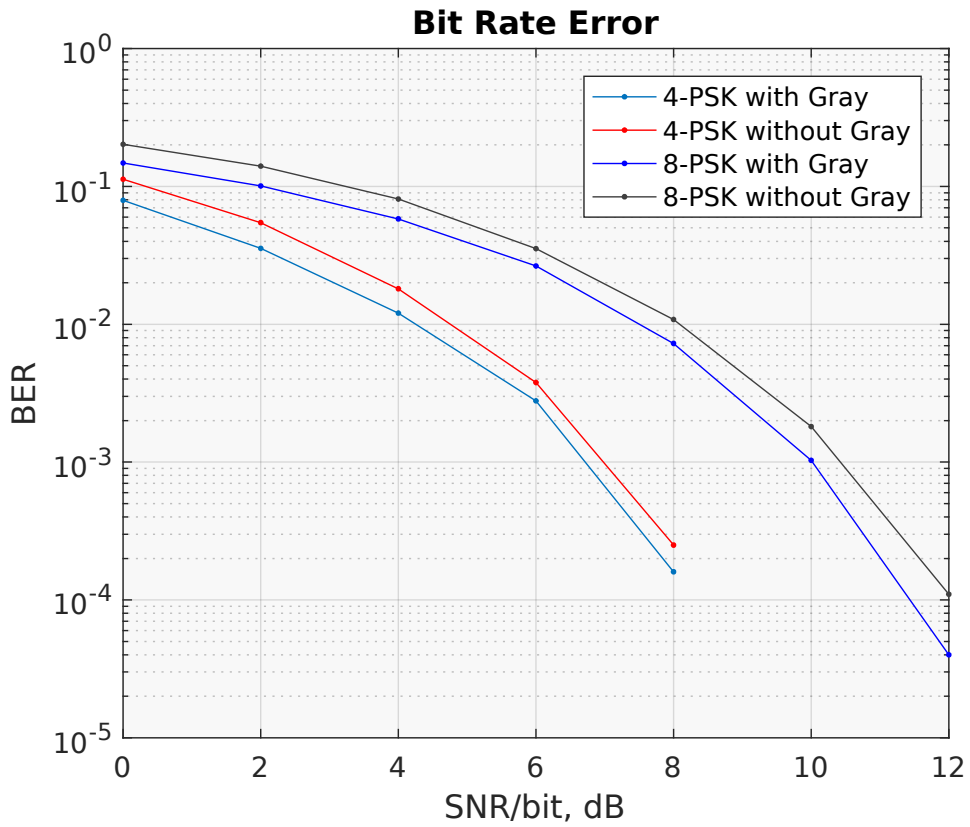
6. **demapper.m**: Στο τέλος γίνεται η αντιστοίχιση στην εκτιμώμενη απεσταλθήσα δυαδική ακολουθία.

Σημείωση: Θα πρέπει να σημειωθεί ότι μεταξύ του διαμορφωτή - αποδιαμορφωτή υπάρχει και πρόσθεση λευκού Gaussian θορύβου στο σήμα που λαμβάνει ο αποδιαμορφωτής. Η διαδικασία της πρόσθεσης του θορύβου γίνεται στο αρχείο **noise.m**.

Αποτελέσματα

Με την χρήση μερικών γραμμών κώδικα για την δημιουργία μιας αναπαράστασης όσων περιγράφηκαν παραπάνω αποτυπώνουμε σε μια γραφική παράσταση τα αποτελέσματά της μελέτης μας για το **bit error rate**, δηλαδή της πιθανότητας σφάλματος bit που αναφέρεται στα λανθασμένα bit που απεστάλησαν προς το συνολικό πλήθος bit που απεστάλησαν.

Οι μετρήσεις έγιναν για *SNR = [0: 2: 16] dB* και για δεδομένα της τάξης των 10^5 bits για πιο μεγάλη αξιοπιστία στα αποτελέσματα, καθώς προσφέρουν αξιοπιστία στις μετρήσεις BER και υπερκαλύπτουν τις ανάγκες μας.



Παρατηρούμε πως για το 4δικό PSK το BER «σβήνει» ταχύτερα όταν εφαρμόζεται κωδικοποίηση Gray ενώ το ίδιο συμβαίνει και με το 8δικό PSK.

Κώδικες MATLAB

binary_input.m

```
function [binary_sequence] = binary_input(number_of_elements)
% binary_sequence = binary_input(number_of_elements):
%
% The binary_input is a function that takes as argument the number of
% elements (0, 1) that will be exported in the binary_sequence array

% initialize the binary_sequence with 0 or 1 as elements, that will have the
% same probability
binary_sequence = randsrc(number_of_elements, 1, [0,1]);

end
```

mapper.m

```
function [ symbols_array ]= mapper(binary_sequence, m, gray)
% symbols_array = mapper(binary_sequence, encoding, gray):
%
% The mapper_modulator is a function that take as argument a binary
% sequence and transforms the elements of this array into symbols
% m is 4 for 4-PSK or 8 for 8-PSK
% The gray argument 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
size_of_binary_sequence = length(binary_sequence);

% we group the bits into groups of log2(m)
% the remainder of the sequence is separately converted into one symbol at
% the end
temp = mod(size_of_binary_sequence, log2(m));

% the sequence which is dividable by log2(m)
new_bin_seq = binary_sequence(1 : (size_of_binary_sequence - temp), :);

% grouping of that sequence
reshaped_sequence = reshape(new_bin_seq, log2(m), (size_of_binary_sequence - temp) /
log2(m));

% transform the sequence into binary code for every group of 3 bits
```

```
for i = 1: (size_of_binary_sequence - temp) / log2(m)
    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(binary_sequence(size_of_binary_sequence - temp +
1 :size_of_binary_sequence, 1)'));
end

% if we use gray encoding in order to achieve smaller distance among 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, 'psk', m);
end

end
```

modulator.m

```
function [s_m] = modulator(symbols_array, m)
% modulator(symbols_array, encoding)
%
% Takes as arguments the symbols array that are to be transmitted and
% encodes it
% Returns the modulated signal

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

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

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

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

% computation of the transmitted signal
for i = 1: size_of_symbols_array
```

```
        for t = 1: T_symbol/T_sample
            s_m(i, t) = g * cos( 2*pi*f_c*t - 2*pi*symbols_array(i)/m );
        end
end

end
```

noise.m

```
function [received_signal] = noise(s_m, SNR, m)
% received_signal = noise(s_m)
% The noise function takes as argument the s_m signal that is to be
% transmitted and the SNR and adds AWGN

% we solve the equation
% 10 * log_10(E_b / N_0) = SNR
% and try to find N_0
% given that
E_s = 1; % and
E_b = E_s / log2(m);
% we have as a result
N_0 = E_b / (10^(SNR/10));

% We create a Gaussian distribution with mean value:
mean = 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 = mean + sigma * randn(L_symbol, T_symbol);

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

end
```

demodulator.m

```
function [r] = demodulator(received_signal, m)
% [r1, r2] = demodulator(received_signal)
% The demodulator function takes as argument the received signal and finds
% the components (r1, ...) of every transmitted signal

% period of symbol
T_symbol = 40;
% frequency of symbol
f_symbol = 1 / T_symbol;
% period of sample
T_sample = 1;
% period of ferousa
T_c = 4;
% frequency of ferousa
f_c = 1 / T_c;
% Energy per symbol
E_s = 1;
% orthogonal pulse
pulse = sqrt(2 * E_s / T_symbol);

[L_symbol, T_symbol] = size(received_signal);
% demodulation
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];
end
```

foratis.m

```
function [symbols] = foratis(r, m)
% symbols = foratis(r1, r2)
% The foratis function takes the r argument and calculates the
% binary (or gray) symbols that was to be send

[r_lines, r_columns] = size(r);

    % calculates each possible received symbol
    for i = 1: m
        s(i, 1) = cos( 2 * pi * i / m );
        s(i, 2) = sin( 2 * pi * i / m );
    end

    % calculates the symbol which presents the greatest probability to
```

```
    % be the sent symbol
    for j =1: r_lines
        for i = 1: m
            temp(i, 1) = norm([r(j,1), r(j,2)] - s(i,:));
        end
        [min_diff, symbols(j, 1)] = min(temp);
    end

% the mth symbol is actually the 0th symbol
symbols = mod(symbols,m);
end
```

demapper.m

```
function [received_bits] = demapper(symbols, m, gray)
% received_bits = demapper(symbols)
% The demapper function converts the received symbols to bits
% The encoding could be with gray encoding (1) or not (0)

% if there has been used Gray encoding in the transmitted signal
if gray == 1
    symbols = gray2bin(symbols, 'psk', m);
end

received_bits = dec2bin(symbols);

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

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

% convert to double every character
% in orde to recover the value that this character represents in 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
received_bits = double(received_bits) - 48;
end
```

ber.m

```
function [BER] = ber(input, output, m)
% BER = ber(input, output)
% calculate the bit error rate

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

% see if it can be divided by log2(m)
modular = mod(length_of_input, log2(m));

% 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 = log2(m) - 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(length_of_input + further_elements_to_add) = input(length_of_input);
    input(length_of_input) = 0;

end

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

script.m

```
bits = 10^5;

i = 1;
x = (0: 2: 16);

for SNR = 0: 2: 16
    A = binary_input(bits);
    B = mapper(A, 4, 1);
    C = modulator(B, 4);
    D = noise(C,SNR, 4);
    r = demodulator(D, 4);
    E = foratis(r, 4);
    F = demapper(E', 4, 1);
    BER_psk4_gray(i, 1) = ber(A,F,4);

    A = binary_input(bits);
    B = mapper(A, 4, 0);
```

```
    C = modulator(B, 4);
    D = noise(C,SNR, 4);
    r = demodulator(D, 4);
    E = foratis(r, 4);
    F = demapper(E', 4, 0);
    BER_psk4_without_gray(i, 1) = ber(A,F,4);

    A = binary_input(bits);
    B = mapper(A, 8, 1);
    C = modulator(B, 8);
    D = noise(C,SNR,8);
    r = demodulator(D, 8);
    E = foratis(r, 8);
    F = demapper(E', 8, 1);
    BER_psk8_gray(i, 1) = ber(A,F,8);

    A = binary_input(bits);
    B = mapper(A, 8, 0);
    C = modulator(B, 8);
    D = noise(C,SNR,8);
    r = demodulator(D, 8);
    E = foratis(r, 8);
    F = demapper(E', 8, 0);
    BER_psk8_without_gray(i, 1) = ber(A,F,8);

    i = i + 1;
end

semilogy(x', BER_psk4_gray, '.-');
hold on;
semilogy(x', BER_psk4_without_gray, 'r.-');
semilogy(x', BER_psk8_gray, 'b.-');
semilogy(x', BER_psk8_without_gray, 'g.-');
legend('4-PSK with Gray','4-PSK without Gray','8-PSK with Gray','8-PSK without Gray');
title('Bit Rate Error');
xlabel('SNR/bit, dB');
ylabel('BER');
hold;

figure;
```