ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ- Η.Μ.Μ.Υ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΑ ΣΥΣΤΗΜΑΤΑ Ι

– ΤΗΛ 301

Ώρες που χρειάστηκαν για την συνολική υλοποίηση:35

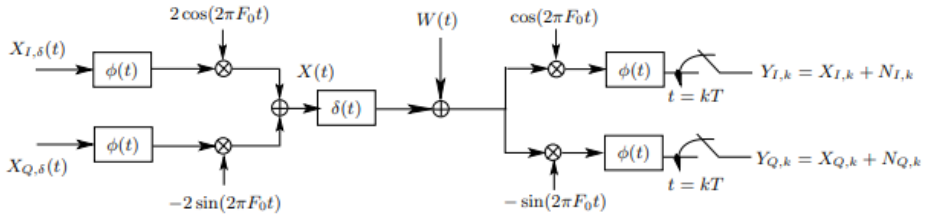
Ομάδα 75:

Χρυσοφάκης Αντώνης 2015030116

Τογρίδης Αλέξανδρος 2019030136

ΑΣΚΗΣΗ 3

Ερώτημα Α



Σε αυτή την άσκηση, θα προσομοιώσουμε το παραπάνω τηλεπικοινωνιακό σύστημα υποθέτοντας ότι χρησιμοποιείται διαμόρφωση 16-PSK, και θα μελετήσουμε την απόδοσή του.

A.1)

```
%% erwthma 1
bit_seq = (sign(randn(N,4))+1)/2;
```

Αρχικά δημιουργήσαμε την δυαδική ακολουθία bit_seq με στοιχεία 4N ισοπίθανα bits.

Ενδεικτικά χρησιμοποιήσαμε N=100.

A.2)

```
%% erwthma 2
Xn=bits_to_PSK_16(bit_seq);
XI=Xn(:,1);
XQ=Xn(:,2);
```

```
function [ x ] = bits_to_PSK_16( bit_seq )
%bits_to_PSK_16 converts a bit sequence into 16-PSK coding
%Picked Gray Code:
%      0000->0001->0011->0010->0110->0111->0101->0100->1100->1101->1111->1110->1010->1011->1001->1000
    for i=1:length(bit_seq)
        if bit_seq(i,1)==0 && bit_seq(i,2)==0 && bit_seq(i,3)==0 && bit_seq(i,4)==0
            x(i,1)=cos(2*pi*0/16);
            x(i,2)=sin(2*pi*0/16);
        elseif bit_seq(i,1)==0 && bit_seq(i,2)==0 && bit_seq(i,3)==0 && bit_seq(i,4)==1
            x(i,1)=cos(2*pi*1/16);
            x(i,2)=sin(2*pi*1/16);
         elseif bit_seq(i,1)==0 && bit_seq(i,2)==0 && bit_seq(i,3)==1 && bit_seq(i,4)==1
            x(i,1)=cos(2*pi*2/16);
            x(i,2)=sin(2*pi*2/16);
         elseif bit_seq(i,1)==0 && bit_seq(i,2)==0 && bit_seq(i,3)==1 && bit_seq(i,4)==0
            x(i,1)=cos(2*pi*3/16);
            x(i,2)=sin(2*pi*3/16);
      elseif bit_seq(i,1)==0 && bit_seq(i,2)==1 && bit_seq(i,3)==1 && bit_seq(i,4)==0
            x(i,1)=cos(2*pi*4/16);
            x(i,2)=sin(2*pi*4/16);
      elseif bit_seq(i,1)==0 && bit_seq(i,2)==1 && bit_seq(i,3)==1 && bit_seq(i,4)==1
            x(i,1)=cos(2*pi*5/16);
            x(i,2)=sin(2*pi*5/16);
      elseif bit_seq(i,1)==0 && bit_seq(i,2)==1 && bit_seq(i,3)==0 && bit_seq(i,4)==1
            x(i,1)=cos(2*pi*6/16);
            x(i,2)=sin(2*pi*6/16);
     elseif bit_seq(i,1)==0 && bit_seq(i,2)==1 && bit_seq(i,3)==0 && bit_seq(i,4)==0
            x(i,1)=cos(2*pi*7/16);
            x(i,2)=sin(2*pi*7/16);
     elseif bit_seq(i,1)==1 && bit_seq(i,2)==1 && bit_seq(i,3)==0 && bit_seq(i,4)==0
```

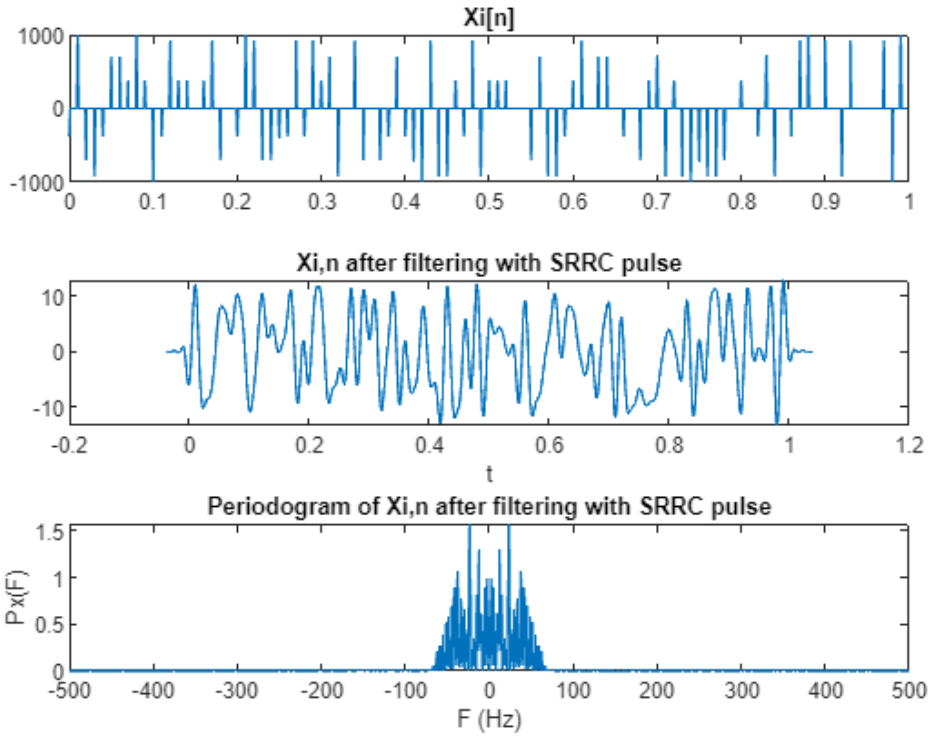
```
            x(i,1)=cos(2*pi*8/16);
            x(i,2)=sin(2*pi*8/16);
        elseif bit_seq(i,1)==1 && bit_seq(i,2)==1 && bit_seq(i,3)==0 && bit_seq(i,4)==1
            x(i,1)=cos(2*pi*9/16);
            x(i,2)=sin(2*pi*9/16);
        elseif bit_seq(i,1)==1 && bit_seq(i,2)==1 && bit_seq(i,3)==1 && bit_seq(i,4)==1
            x(i,1)=cos(2*pi*10/16);
            x(i,2)=sin(2*pi*10/16);
        elseif bit_seq(i,1)==1 && bit_seq(i,2)==1 && bit_seq(i,3)==1 && bit_seq(i,4)==0
            x(i,1)=cos(2*pi*11/16);
            x(i,2)=sin(2*pi*11/16);
        elseif bit_seq(i,1)==1 && bit_seq(i,2)==0 && bit_seq(i,3)==1 && bit_seq(i,4)==0
            x(i,1)=cos(2*pi*12/16);
            x(i,2)=sin(2*pi*12/16);
        elseif bit_seq(i,1)==1 && bit_seq(i,2)==0 && bit_seq(i,3)==1 && bit_seq(i,4)==1

             x(i,1)=cos(2*pi*13/16);
             x(i,2)=sin(2*pi*13/16);
         elseif bit_seq(i,1)==1 && bit_seq(i,2)==0 && bit_seq(i,3)==0 && bit_seq(i,4)==1
            x(i,1)=cos(2*pi*14/16);
            x(i,2)=sin(2*pi*14/16);
         elseif bit_seq(i,1)==1 && bit_seq(i,2)==0 && bit_seq(i,3)==0 && bit_seq(i,4)==0
            x(i,1)=cos(2*pi*15/16);
            x(i,2)=sin(2*pi*15/16);
         end
    end
end
```

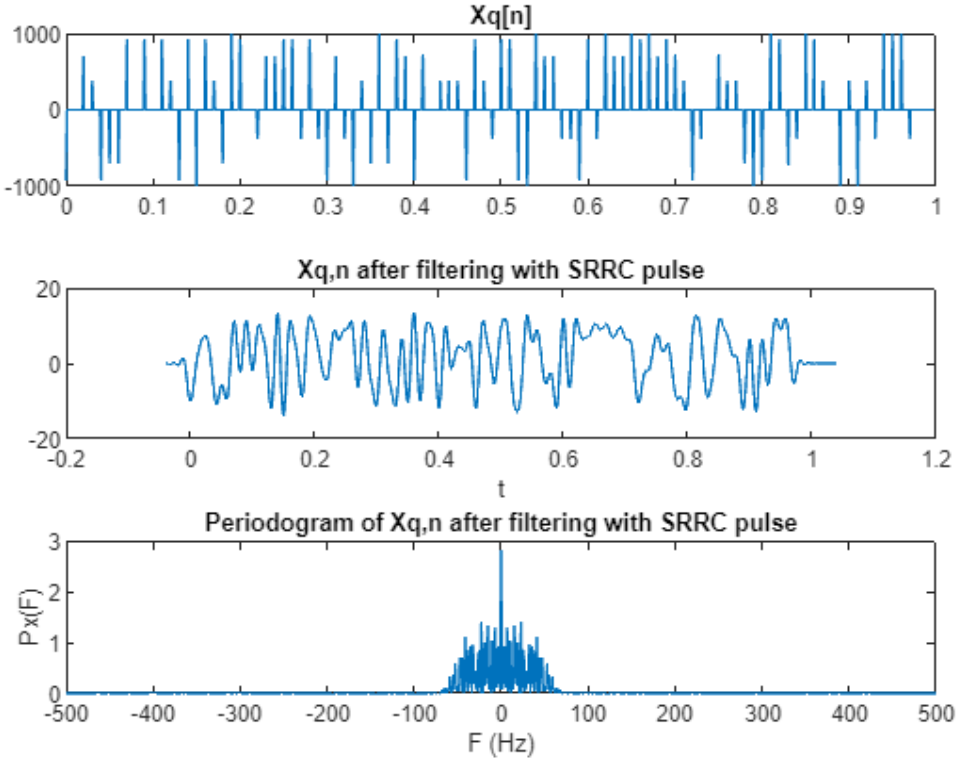
Στη συνέχεια υλοποιήσαμε την συνάρτηση bits_to_PSK_16 η οποία χρησιμοποιεί τον κώδικα Gray:

`0000->0001->0011->0010->0110->0111->0101->0100->1100->1101->1111->1110->1010->1011->1001->1000`

Έτσι μπορέσαμε να κάνουμε την κατάλληλη απεικόνιση της ακολουθίας bits(bit_seq)σε ακολουθία 16-PSK συμβόλων. Ο πίνακας X που επιστρέφεται έχει N γραμμές και 2 στήλες. Από την πρώτη στήλη ορίζουμε το διάνυσμα XI ενώ από την δεύτερη το διάνυσμα XQ όπως φαίνεται παραπάνω.

A.3)


Xi[n]
1000
0
-1000
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Xi,n after filtering with SRRC pulse
10
0
-10
-0.2 0 0.2 0.4 0.6 0.8 1 1.2
t
Periodogram of Xi,n after filtering with SRRC pulse
Px(F)
1.5
1
0.5
0
-500 -400 -300 -200 -100 0 100 200 300 400 500
F (Hz)

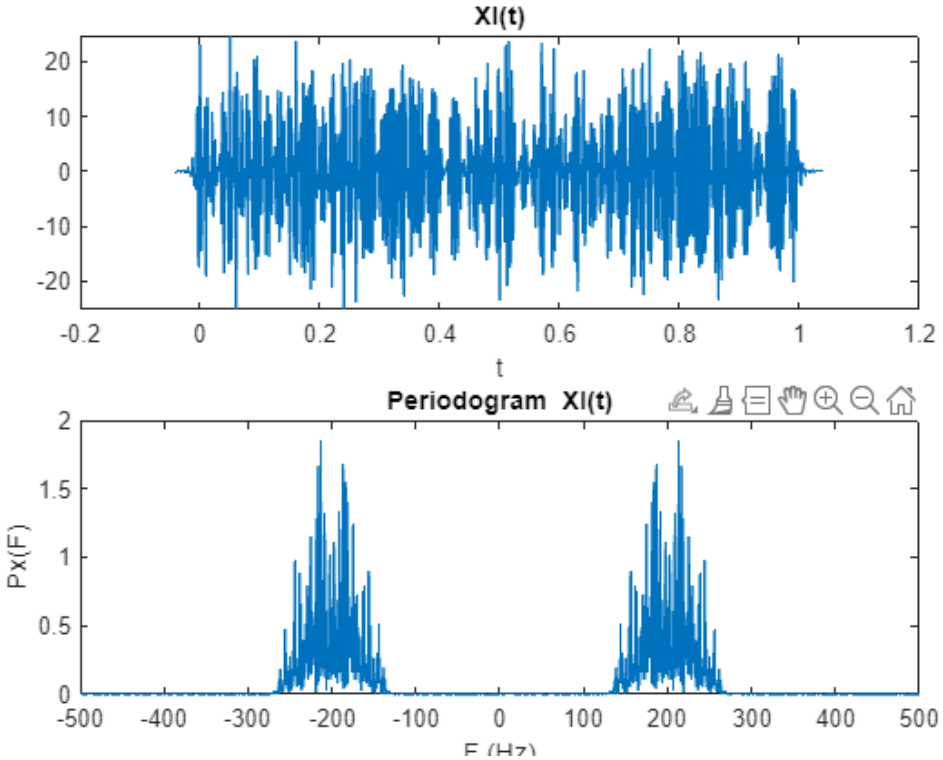

Xq[n]
1000
0
-1000
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Xq,n after filtering with SRRC pulse
20
0
-20
-0.2 0 0.2 0.4 0.6 0.8 1 1.2
t
Periodogram of Xq,n after filtering with SRRC pulse
Px(F)
3
2
1
0
-500 -400 -300 -200 -100 0 100 200 300 400 500
F (Hz)

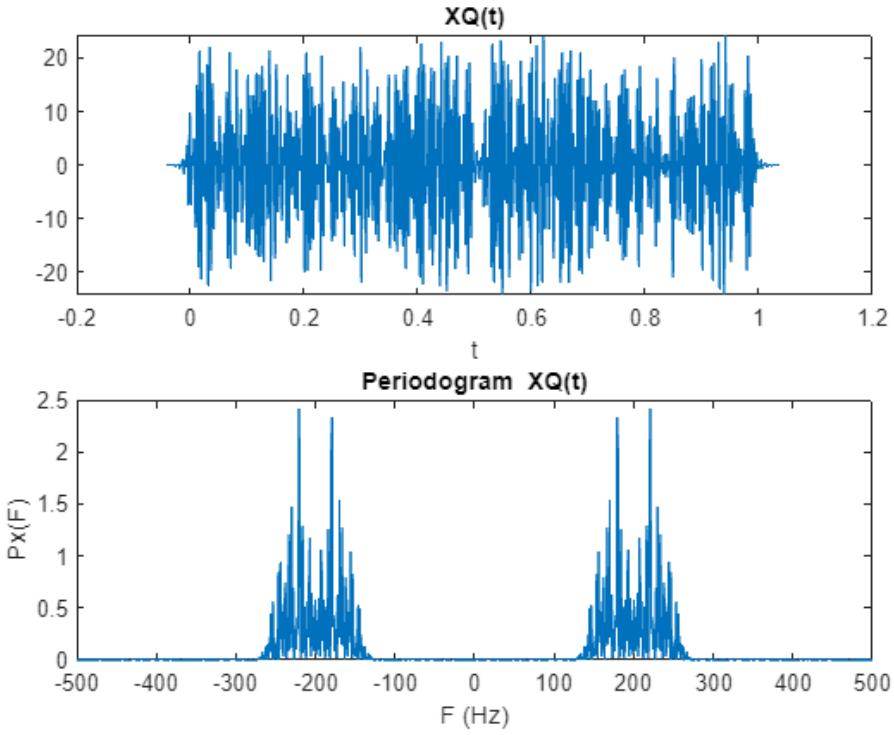
```
%% A.3
%Create SRRC pulse
[ph,t] = srrc_pulse(T,over,A,a);
F = -Fs/2:Fs/Nf:Fs/2-Fs/Nf;
%Create Xd signals for usage in Convolution
Xdi = Fs*upsample(XI,over);
t_Xdi =(0:Ts:N/(1/T)-Ts);
Xdq = Fs*upsample(XQ,over);
t_Xdq = (0:Ts:N/(1/T)-Ts);
%Compute Convolution and create time axis. Also find length of C
t_Xti = (t(1)+t_Xdi(1):Ts:t(end)+t_Xdi(end));
t_Xtq = (t(1)+t_Xdq(1):Ts:t(end)+t_Xdq(end));
Xti = conv(ph,Xdi)*Ts;
Xtq = conv(ph,Xdq)*Ts;
Xti_total=length(t_Xti)*Ts;
Xtq_total = length(t_Xtq)*Ts;
%Fourier Transform and compute Periodogram for both signals
Fxi=fftshift(fft(Xti,Nf)*Ts);
Pxi=(abs(Fxi).^2)/Xti_total;
Fxq=fftshift(fft(Xtq,Nf)*Ts);
Pxq=(abs(Fxq).^2)/Xtq_total;
%Show results for each signal
figure();
subplot(3,1,1);
plot(t_Xdi,Xdi);
title('Xi[n]');
subplot(3,1,2);
plot(t_Xti,Xti);
title('Xi,n after filtering with SRRC pulse');
xlabel('t');
subplot(3,1,3);
plot(F,Pxi);
title('Periodogram of Xi,n after filtering with SRRC pulse');
xlabel('F (Hz)');
ylabel('Px(F)');

figure();
subplot(3,1,1);
plot(t_Xdq,Xdq);
title('Xq[n]');
subplot(3,1,2);
plot(t_Xtq,Xtq);
title('Xq,n after filtering with SRRC pulse');
xlabel('t');
subplot(3,1,3);
plot(F,Pxq);

title('Periodogram of Xq,n after filtering with SRRC pulse');
xlabel('F (Hz)');
ylabel('Px(F)');
```

Σχολιασμός Κώδικα

Αφού θέσαμε το A=4 και φτιάξαμε τον παλμό SRRC από την συνάρτηση η οποία μας δίνεται και τις σταθερές τις οποίες αναφέρει η εκφώνηση. Έπειτα ορίσαμε το διάστημα της συχνότητας ώστε να χρησιμοποιηθεί στον σχεδιασμό των περιοδογραμμάτων. Χρησιμοποιήσαμε την upsample() για την δημιουργία κατάλληλων σημάτων για να χρησιμοποιηθούν στην συνέλιξη. Ουσιαστικά φτιάξαμε τα σήματα $XI_{\delta}(t)$ και $XQ_{\delta}(t)$. Παρακάτω κάναμε την συνέλιξη του κάθε σήματος με τον παλμό(φιλτράρισμα) και δημιουργήσαμε τον κατάλληλο άξονα χρόνου. Έγινε ο μετασχηματισμός Fourier για την κάθε συνέλιξη και υπολογίσαμε το περιοδόγραμμά του. Τέλος, εμφανίζουμε τα φιλτραρισμένα σήματα ,τα περιοδογράμματά τους και τις ακολουθίες XI,n και XQ,n στους κατάλληλους άξονεσ με plot. (Εναλλακτικά χρησιμοποιούμε semilogy).

A.4)






```
%% A.4
%Create XQ(t) and  XI(t) signals
XTq = -2*(Xtq).*(sin(2*pi*Fo*transpose(t_Xtq)));
Xtq_total = length(t_Xtq)*Ts;
XTi = 2*(Xti).*(cos(2*pi*Fo*transpose(t_Xti)));
Xti_total = length(t_Xti)*Ts;


%Fourier Transform and compute Periodogram for both signals
Fxq = fftshift(fft(XTq,Nf)*Ts);
Pxq = (abs(Fxq).^2)/Xtq_total;
Fxi = fftshift(fft(XTi,Nf)*Ts);
Pxi = (abs(Fxi).^2)/Xti_total;

%The results for each signal
figure();
subplot(2,1,1);
plot(t_Xti,XTi);
title('XI(t)');
xlabel('t');
subplot(2,1,2);
plot(F,Pxi);
title('Periodogram  XI(t)');
xlabel('F (Hz)');
ylabel('Px(F)');

figure();
subplot(2,1,1);
plot(t_Xtq,XTq);
title('XQ(t)')
xlabel('t');

 subplot(2,1,2);
 plot(F,Pxq);
 title('Periodogram  XQ(t)');
 xlabel('F (Hz)');
 ylabel('Px(F)');
```

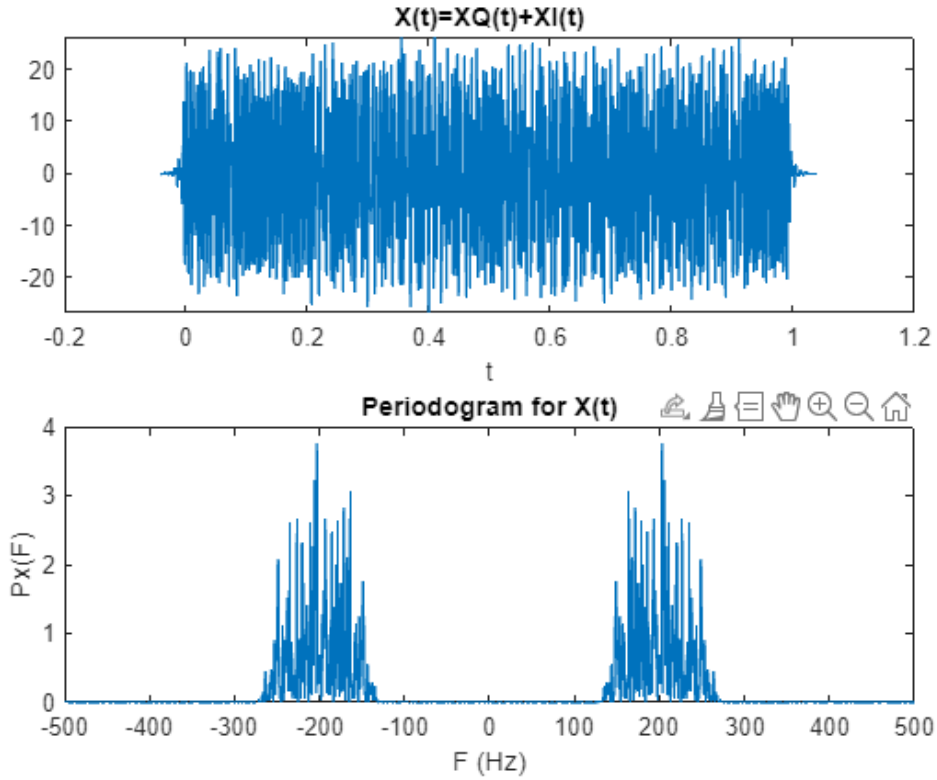
Σχολιασμός Κώδικα

Όπως αναφέρεται στην εκφώνηση της άσκησης έχουμε Fo=200Hz. Έπειτα πολλαπλασιάζουμε τα φιλτραρισμένα σήματα με τους αντίστοιχους φορείς συχνότητας Fo όπως φαίνεται και στην απεικόνισης του συστήματος. Έτσι δημιουργήσαμε τα σήματα $X_I(t)$ και $X_Q(t)$ όπως αυτά περιγράφονται στην άσκηση. Υπολογίσαμε τον μετασχηματισμό Fourier για κάθε σήμα και στη συνέχεια το περιοδόγραμμά του. Οι άξονες του χρόνου και της συχνότητας δεν αλλάζουν, επομένως εμφανίζουμε τα αποτελέσματα με plot. (Εναλλακτικά χρησιμοποιούμε semilogy).

Παρατηρήσεις Αποτελεσμάτων

Από την στιγμή που πολλαπλασιάσαμε με τους φορείς, το σύστημα δεν είναι πλέον βασικής ζώνης, αφού το φάσμα του αρχικού σήματος μετακινείται γύρω από την συχνότητα του φορέα. Αυτό μπορούμε να το συμπεράνουμε παρατηρώντας τα δύο περιοδογράμματα. Παρατηρούμε όντως ότι το φάσμα έχει μετακινηθεί γύρω από τη συχνότητα Fo(200Hz).Επίσης διακρίνουμε πως αυξήθηκε το πλάτος αλλά και η συχνότητα του κάθε σήματος αντίστοιχα , όπως ήταν αναμενόμενο καθώς πολλαπλασιάστηκαν με το φορέα.

Α.5)




```
%% A.5
%Create channel input signal
Xt = XTi+XTq;
T_total = length(t_Xtq)*Ts;
%Fourier Transform and compute Periodogram
Fxt = fftshift(fft(Xt,Nf)*Ts);
Pxt = (abs(Fxt).^2)/T_total;

figure();
subplot(2,1,1);
plot(t_Xtq,Xt);
title('X(t)=XQ(t)+XI(t)');
xlabel('t');
subplot(2,1,2);
plot(F,Pxt);
title('Periodogram for X(t)');
xlabel('F (Hz)');
ylabel('Px(F)');
```

Σχολιασμός Κώδικα

Για να αναλύσουμε την είσοδο του καναλιού X(t), προσθέσαμε τα δύο σήματα XI(t) και XQ(t). Υπολογίσαμε το μετασχηματισμό Fourier και στη συνέχεια δημιουργήσαμε το περιοδόγραμμα του σήματος εισόδου. Παρουσιάσαμε γραφικά τα αποτελέσματα χρησιμοποιώντας κατάλληλους άξονες με τη χρήση της συνάρτησης plot. Εναλλακτικά, μπορούσαμε να χρησιμοποιήσουμε τη συνάρτηση semilogy για την απεικόνιση.

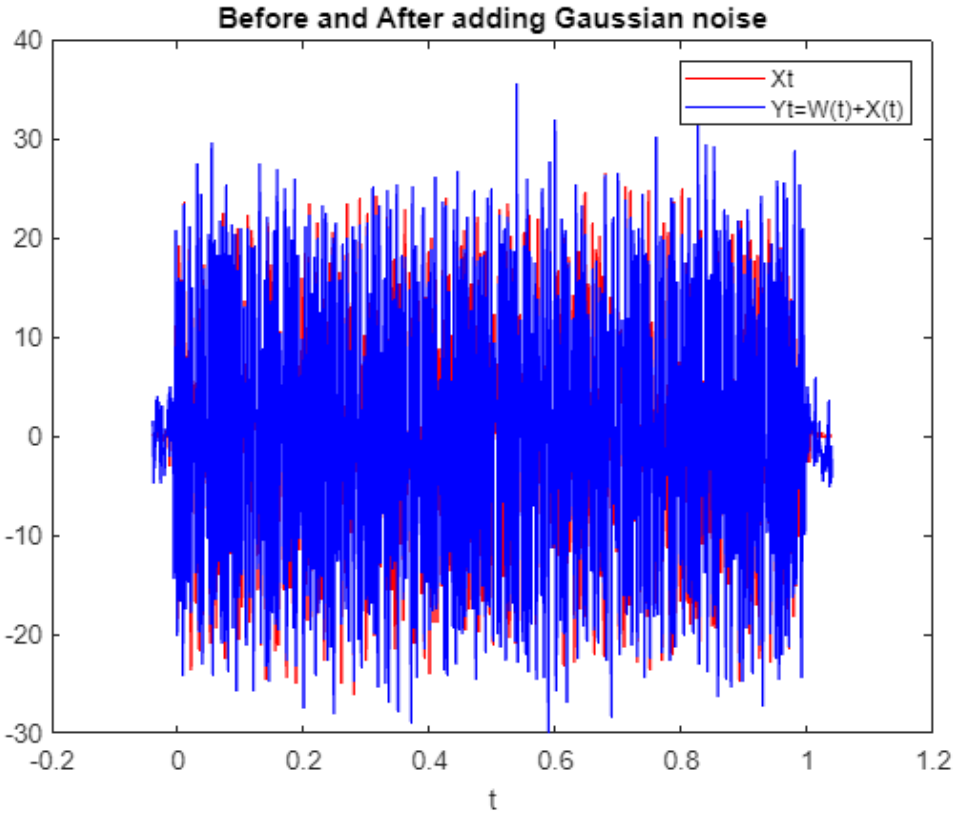
Παρατηρήσεις Αποτελεσμάτων

Βλέποντας το σήμα που τελικά θα είναι η είσοδος του καναλιού, διακρίνουμε ότι το πλάτος έχει αυξηθεί,. Αυτό ήταν αναμενόμενο λόγω της πρόσθεσης των δύο συνιστωσών. Τέλος ξαναβλέπουμε την επίδραση του φορέα στο αρχικό μας σήμα καθώς το κέντρο έχει μετακινηθεί στις συχνότητες Fo και –Fo.

A.6)

Εφόσον υποθέσαμε ότι το κανάλι είναι ιδανικό, το σύστημα έχει κρουστική απόκριση δ(t),άρα η έξοδος του θα είναι το σήμα εισόδου αναλλοίωτο.

A.7)

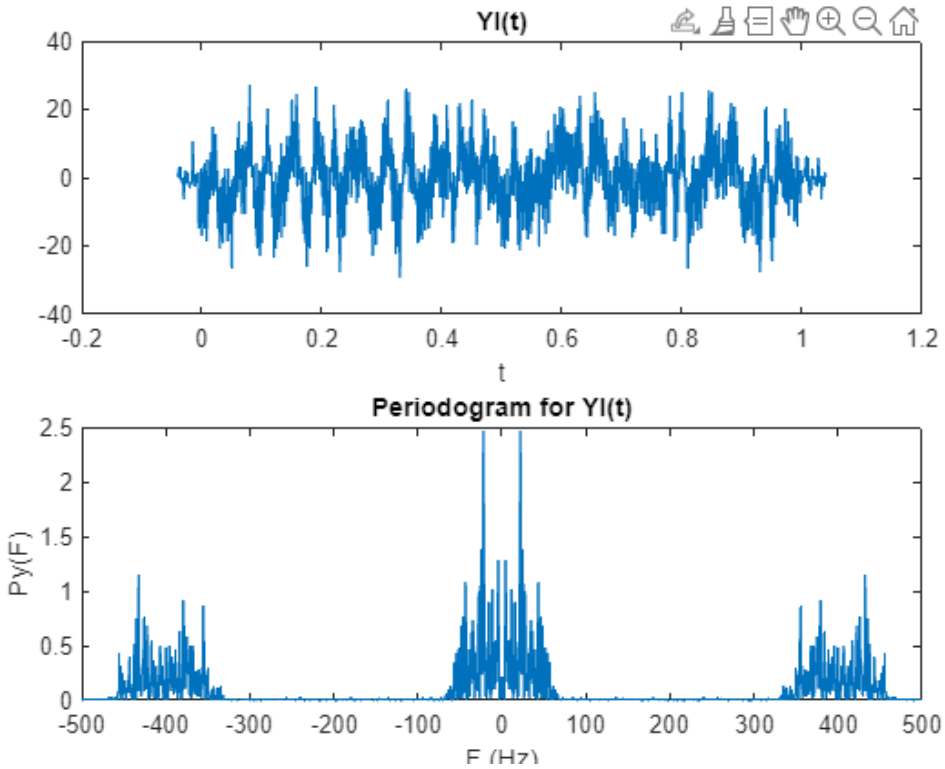


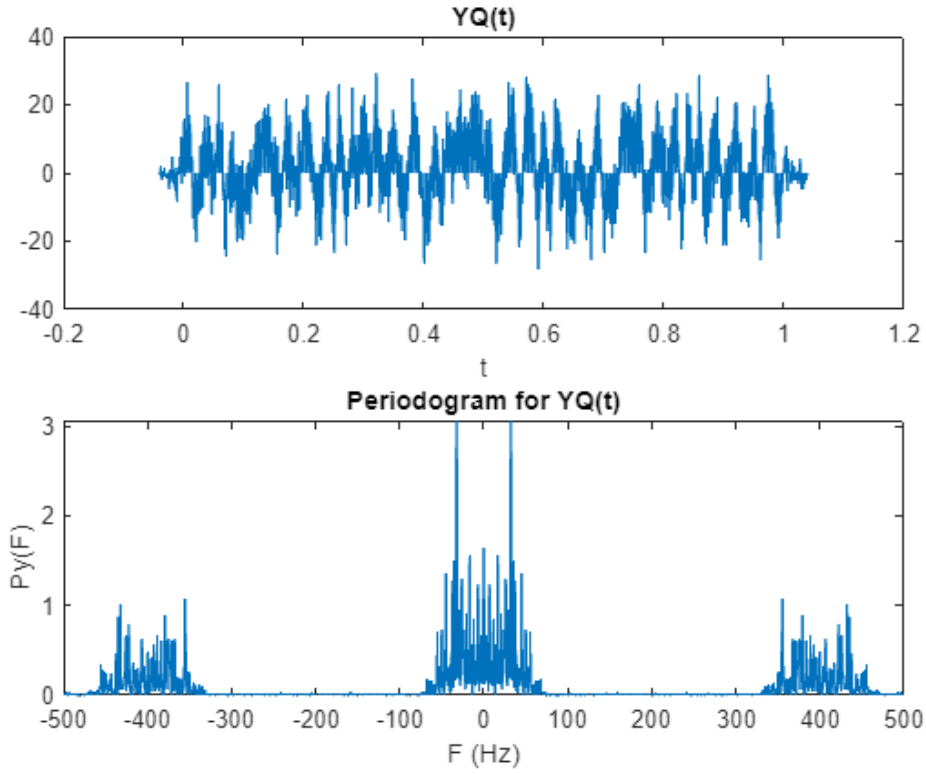
```
%% A.7
%Compute variance and create Gaussian Noise
SNR = 20;
s2w = 1/(Ts*(10^(SNR/10)));
Wt = sqrt(s2w).*randn(length(Xt),1);
%Making the Y(t) signal
Yt = Xt+Wt;
%Here are the results
figure();
plot(t_Xtq,Xt,'red');
hold on;
plot(t_Xtq,Yt,'blue');
hold off;
title('Before and After adding Gaussian noise');
legend('Xt','Yt=W(t)+X(t)');
xlabel('t');
```

<u>Σχολιασμός Κώδικα</u>

Αφού επιλέξαμε SRN=20, ορίσαμε την διασπορά του θορύβου και έπειτα δημιουργήσαμε το σήμα W(t) (Gaussian θόρυβο) . Στη συνέχεια προσθέσαμε την είσοδο του καναλιού X(t) και τον θόρυβο και δημιουργήσαμε την ενθόρυβη κυματομορφή Y(t). Παραπάνω βλέπουμε σχεδιασμένο το σήμα X(t) πριν και μετά την προσθήκη θορύβου.

A.8)






```
%% A.8
%Making YI(t) and YQ(t) signals
Yq_1 = Yt.*(-sin(2*pi*Fo*transpose(t_Xtq)));
Yi_1 = Yt.*(cos(2*pi*Fo*transpose(t_Xtq)));

Tyq1_total = length(t_Xtq)*Ts;
Tyi1_total = length(t_Xtq)*Ts;
%Fourier Transform and compute Periodogram for both signals
Fyi1_x = fftshift(fft(Yi_1,Nf)*Ts);
Pyi1_x = (abs(Fyi1_x).^2)/Tyi1_total;

Fyq1_x = fftshift(fft(Yq_1,Nf)*Ts);
Pyq1_x = (abs(Fyq1_x).^2)/Tyq1_total;
%Here are the results for each signal
figure();
subplot(2,1,1);
plot(t_Xtq,Yi_1);
title('YI(t)');
xlabel('t');
subplot(2,1,2);
plot(F,Pyi1_x);
title('Periodogram for YI(t)');

xlabel('F (Hz)');
ylabel('Py(F)');

figure();
subplot(2,1,1);
plot(t_Xtq,Yq_1);
title('YQ(t)')
xlabel('t');
subplot(2,1,2);
plot(F,Pyq1_x);
title('Periodogram for YQ(t)');
xlabel('F (Hz)');
ylabel('Py(F)');
```

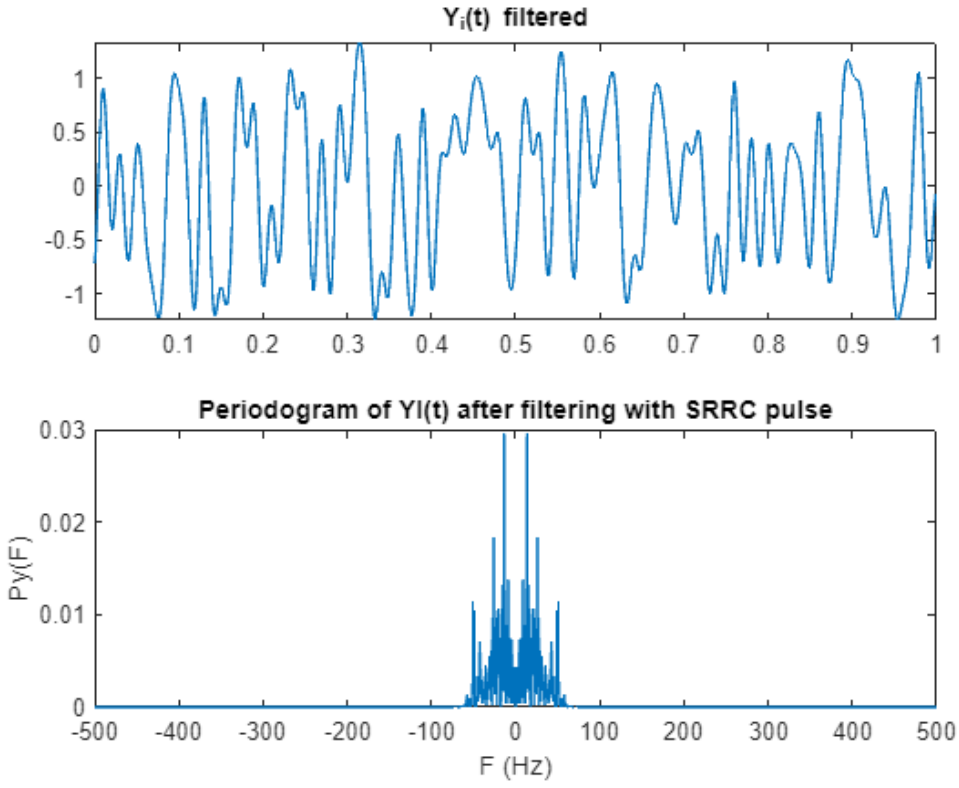
Σχολιασμός Κώδικα

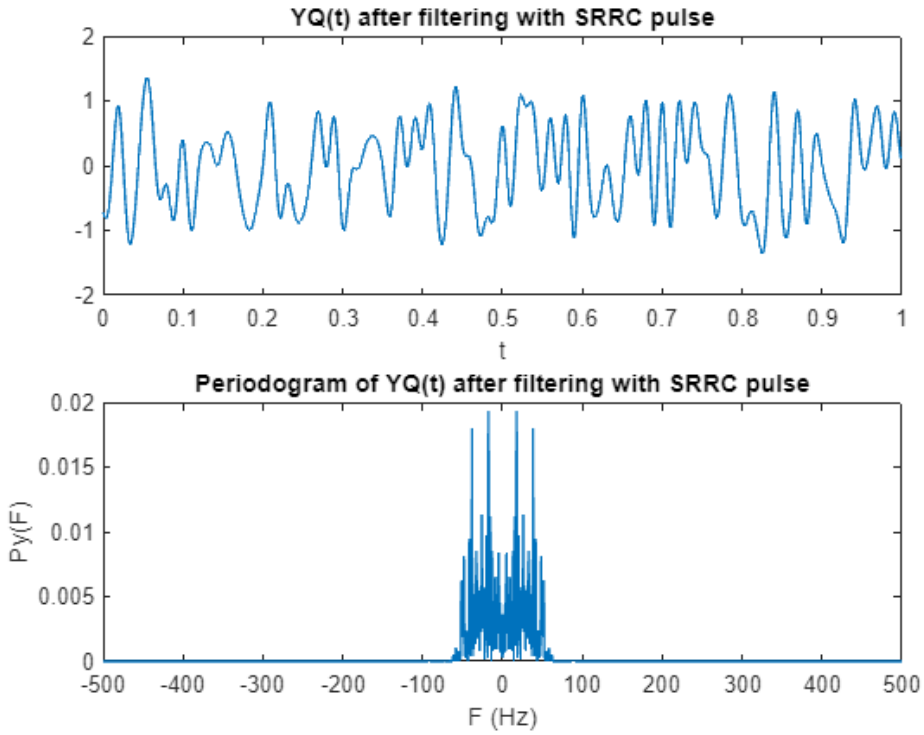
Πρώτα απ' όλα πολλαπλασιάσαμε την κάθε συνιστώσα με τον αντίστοιχο φορέα συχνότητας Fo όπως φαίνεται στην απεικόνιση του συστήματος. Έπειτα κάναμε τον μετασχηματισμό Fourier και υπολογίσαμε το περιοδόγραμμα για τα δύο σήματα που προκύπτουν ξεχωριστά. Τέλος σχεδιάσαμε τα σήματα και τα περιοδόγραμμάτα τους με την χρήση της εντολής plot.(Εναλλακτικά χρησιμοποιούμε semilogy).

Παρατηρήσεις Αποτελεσμάτων

Αρχικά επαναφέρουμε το σήμα στην αρχική ζώνη ώστε να μελετηθεί περαιτέρω. Αυτός είναι και ο λόγος που πολλαπλασιάζουμε με τους αντίστοιχους φορείς(αποδιαμόρφωση). Στα περιοδογράμματα παρατηρούμε πως εμφανίστηκε το φάσμα στη βασική ζώνη, ενώ οι τιμές που αντιστοιχούν στις υψηλές συχνότητες τις οποίες έχουν καθορίσει οι διαμορφωτές, εξακολουθούν να υπάρχουν. Αυτός είναι και ο λόγος που στο επόμενο βήμα θα χρησιμοποιήσουμε τα αντίστοιχα φίλτρα ώστε να περιορίσουμε το πλάτος των λοβών αυτών.

Α.9)






```
%% A.9
%Using SRRC pulse (ph) from A.3
%Compute Convolution and create time axis.
Yi_2 = conv(ph,Yi_1)*Ts;
t_Yi_2 = (t(1)+t_Xtq(1):Ts:t(end)+t_Xtq(end));
Yq_2 = conv(ph,Yq_1)*Ts;
t_Yq_2 = (t(1)+t_Xtq(1):Ts:t(end)+t_Xtq(end));

%Tail cutting
counter = 0;
for n = 1:length(t_Yi_2)
    if t_Yi_2(n)<0
       counter = counter+1;
    end
end
Yi_cut = Yi_2(counter+1:(length(t_Yi_2)-(counter)));
t_Yi_cut = t_Yi_2(counter+1:(length(t_Yi_2)-(counter)));

Yq_cut = Yq_2(counter+1:(length(t_Yi_2)-(counter)));
t_Yq_cut = t_Yq_2(counter+1:(length(t_Yi_2)-(counter)));
%Fourier Transform and compute Periodogram for both cutted signals
Tyi_2_total = length(t_Yi_cut)*Ts;
Fyi_2 = fftshift(fft(Yi_cut,Nf)*Ts);
Pyi_2 = (abs(Fyi_2).^2)/Tyi_2_total;

Tyq_2_total = length(t_Yq_cut)*Ts;
Fyq_2 = fftshift(fft(Yq_cut,Nf)*Ts);
Pyq_2 = (abs(Fyq_2).^2)/Tyq_2_total;
%Here are the results for each signal
figure();
```

```
subplot(2,1,1);
plot(t_Yi_cut,Yi_cut);
title('Y_i(t) filtered');
subplot(2,1,2);
plot(F,Pyi_2);
title('Periodogram of YI(t) after filtering with SRRC pulse');
xlabel('F (Hz)');
ylabel('Py(F)');


figure();
subplot(2,1,1);
plot(t_Yq_cut,Yq_cut);
title('YQ(t) after filtering with SRRC pulse')
xlabel('t');
subplot(2,1,2);
plot(F,Pyq_2);
title('Periodogram of YQ(t) after filtering with SRRC pulse');
xlabel('F (Hz)');
ylabel('Py(F)');
```

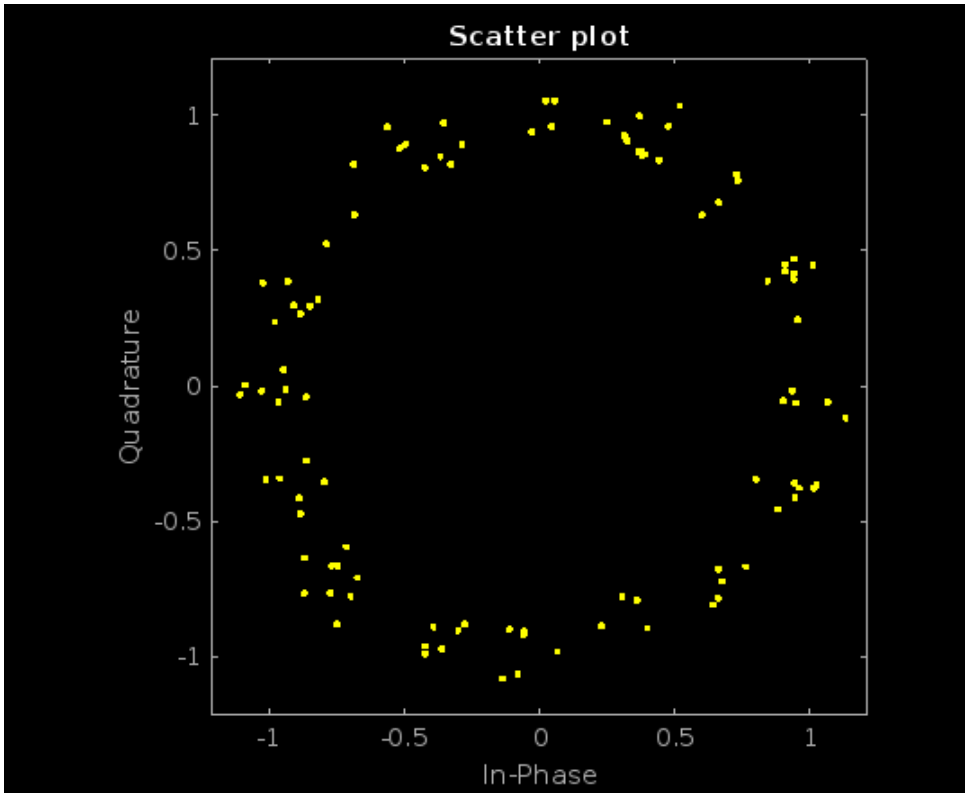
Σχολιασμός Κώδικα

Αφού κάναμε την συνέλιξη μεταξύ του κάθε σήματος και του SRRC παλμού (ph) που δημιουργήθηκε στο ερώτημα Α.3, ορίσαμε το νέο άξονα του χρόνου ώστε να μπορέσουμε να σχεδιάσουμε τη συνέλιξη και μετέπειτα υπολογίσαμε την διάρκεια της συνέλιξης ώστε να χρησιμοποιηθεί στο περιοδόγραμμα. Στη συνέχεια με μια επαναληπτική δομή (for) εργαστήκαμε κατάλληλα ΄ώστε να κόψουμε τις “ουρές” που εμφανίζονται στις δύο συνελίξεις και να δημιουργήσουμε δύο σήματα χωρίς αυτές ώστε να χρησιμοποιηθούν στην δειγματοληψία του επόμενου ερωτήματος. Έπειτα υπολογίσαμε τον μετασχηματισμό Fourier και το περιοδόγραμμα της κάθε συνέλιξης(με αποκομμένη την ουρά). Τέλος σχεδιάσαμε τα φιλτραρισμένα σήματα (συνέλιξη) και το αντίστοιχο περιοδόγραμμα τους με την χρήση της εντολής plot.

Παρατηρήσεις Αποτελεσμάτων

Εξετάζοντας τα περιοδογράμματα επαληθεύουμε ότι οι λοβοί στις υψηλές συχνότητες δεν λαμβάνονται υπόψιν λόγω του φιλτραρίσματος. Επιπλέον διακρίνουμε μια ομοιότητα της κάθε συνιστώσας με την αντίστοιχη που είχε προκύψει στο ερώτημα Α.3.

Α.10)




```
%% A.10
%Downsampling and creating Yi,n and Yq,n
YI = downsample(Yi_cut,over);
YQ = downsample(Yq_cut,over);
%Creating Yn sequence
Yn=[YI YQ];
%Showing the results with the use of Scatterplot
scatterplot(Yn);
```

<u>Σχολιασμός Κώδικα</u>

Καταρχήν χρησιμοποιήσαμε το downsample() ώστε να γίνει η δειγματοληψία στις δύο συνελίξεις με αποκομμένες τις “ουρές”. Στη συνέχεια για κάθε σύμβολο δημιουργήσαμε τον πίνακα Yn με N(επιλέξαμε 100) γραμμές και 2 στήλες.

<u>Παρατηρήσεις Αποτελεσμάτων</u>

Όπως περιμέναμε ο θόρυβος που προστέθηκε έχει δημιουργήσει αρκετές αποκλίσεις οι οποίες είναι ανάλογες του SNR. έτσι προκύπτει και το νέφος γύρω από τα 16 σημεία που θα αναμέναμε να παρουσιάζονται.

Α.11)

```
%% A.11
[est_X, est_bit_seq] = detect_PSK_16(Yn);
```

```
function [est_X, est_bit_seq] = detect_PSK_16(Y)
    % Αρχικοποίηση συμβόλων 16-PSK
    symbols = exp(1i*(0:15)*pi/8);

    % Προετοιμασία αποθήκευσης των εκτιμήσεων
    est_X = zeros(size(Y));
    est_bit_seq = zeros(size(Y, 1), 4);

    % Εύρεση του πλησιέστερου συμβόλου για κάθε δείγμα Y
    for i = 1:size(Y, 1)
        [~, index] = min(abs(Y(i) - symbols));
        est_X(i) = symbols(index);

        % Αντίστροφη απεικόνιση Gray
        est_bit_seq(i, :) = de2bi(index-1, 4, 'left-msb');
    end
end
```

<u>Σχολιασμός Κώδικα</u>

Η συνάρτηση **detect_PSK16** χρησιμοποιείται για την ανίχνευση συμβόλων 16-PSK από μια εισερχόμενη κυματομορφή Y. Πρώτα γίνεται η αρχικοποίηση συμβόλων 16-PSK και η δημιουργία ενός πίνακα με τα 16 σύμβολα του 16-PSK σε μορφή πολικών συντεταγμένων. Έπειτα δημιουργεί δύο κενούς πίνακες (**est_X** και **est_bit_seq**) με το ίδιο μέγεθος με την είσοδο **Y**, για την αποθήκευση των εκτιμήσεων. Στη συνέχεια για την εύρεση του πλησιέστερου συμβόλου, για κάθε δείγμα **Y(i)** στην είσοδο, υπολογίζει την απόστασή του από όλα τα σύμβολα 16-PSK και επιλέγει το πλησιέστερο σύμβολο και αποθηκεύει το εκτιμώμενο σύμβολο στον πίνακα **est_X**. Τέλος για να πετύχουμε την αντίστροφη απεικόνιση Gray κάνουμε μία μετατροπή το εκτιμώμενο σύμβολο σε μια δυαδική ακολουθία(με την χρήση της εντολής de2bi)αντιστοιχίζοντάς το σε μια τετράδα bits και αποθηκεύουμε την εκτιμώμενη δυαδική ακολουθία στον πίνακα **est_bit_seq.** Η συνάρτηση μας επιστρέφει τους πίνακες **est_X** και **est_bit_seq** που περιέχουν τις εκτιμήσεις των συμβόλων 16-PSK και τις αντίστοιχες δυαδικές ακολουθίες.

A.12)

```
%% A.12
Num_Of_Symbol_errors = symbol_errors(est_X,Xn)
```

```
function [ num_of_symbol_errors] = symbol_errors(est_X,X )
%Compares  the symbols at the imput(X) with the estimation of them at the
%output(est_X)
%       est_X:        Output Symbol Sequence Estimation
%       X:                Input Symbol Sequence
%Returns total number of mismatches found
num_of_symbol_errors=0;
N=length(X);
%Posible values [-1, -sqrt(2)/2 , 0 , sqrt(2)/2 , 1]
%Multiply by 2 and make values integer so we can compare
est_X_toInt=round(2.*est_X);
X_toInt=round(2.*X);
%Now possible values are [-2 ,-1, 0, 1, 2]
for i=1:N
    %There is a mismatch.Increase total errors
    if est_X_toInt(i,1)~=X_toInt(i,1) || est_X_toInt(i,2)~=X_toInt(i,2)
        num_of_symbol_errors=num_of_symbol_errors+1;
    end
end
end
```

<u>Σχολιασμός Κώδικα</u>

Η συνάρτηση **symbol_errors** συγκρίνει τα σύμβολα στην είσοδο (**X**) με την εκτίμησή τους στην έξοδο (**est_X**). Υπολογίζει τον συνολικό αριθμό των σφαλμάτων που βρέθηκαν, δηλαδή τον αριθμό των θέσεων στις οποίες τα σύμβολα διαφέρουν. Η μεταβλητή **num_of_symbol_errors** αυξάνεται κάθε φορά που υπάρχει αντιστοίχιση σφάλματος μεταξύ των συμβόλων. Το αποτέλεσμα **num_of_symbol_errors** είναι ο συνολικός αριθμός των λαθών που βρέθηκαν.

A.13)

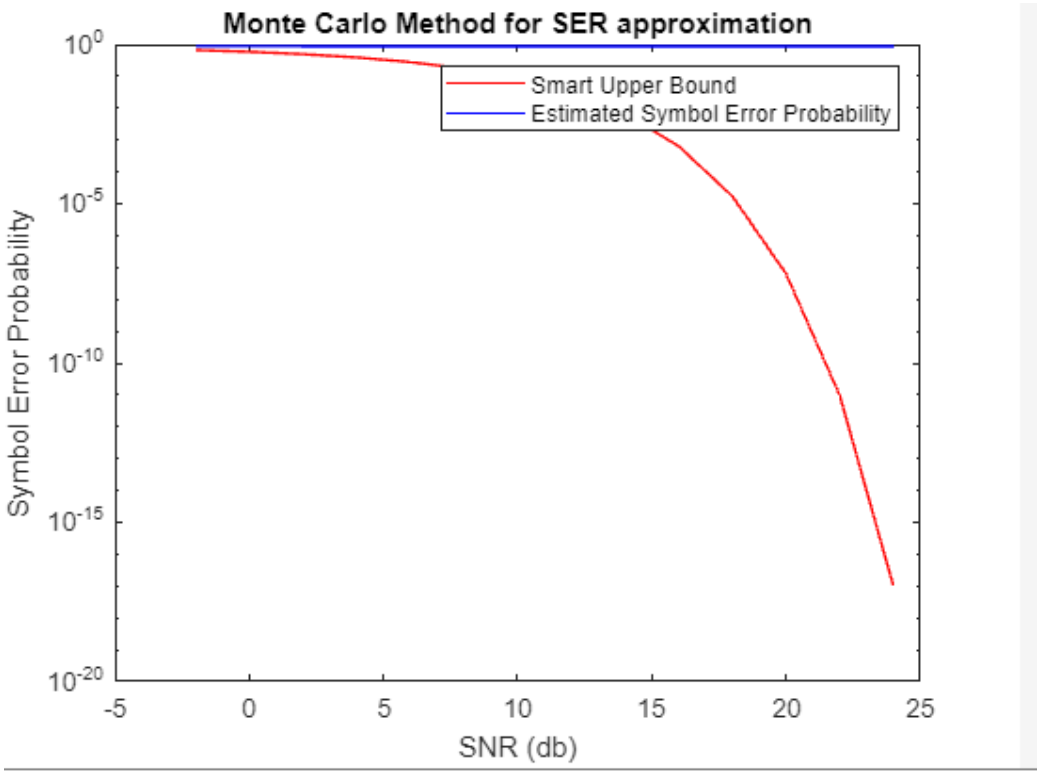
```
%% A.13
Num_Of_Bit_errors = bit_errors(est_bit_seq,bit_seq)
```

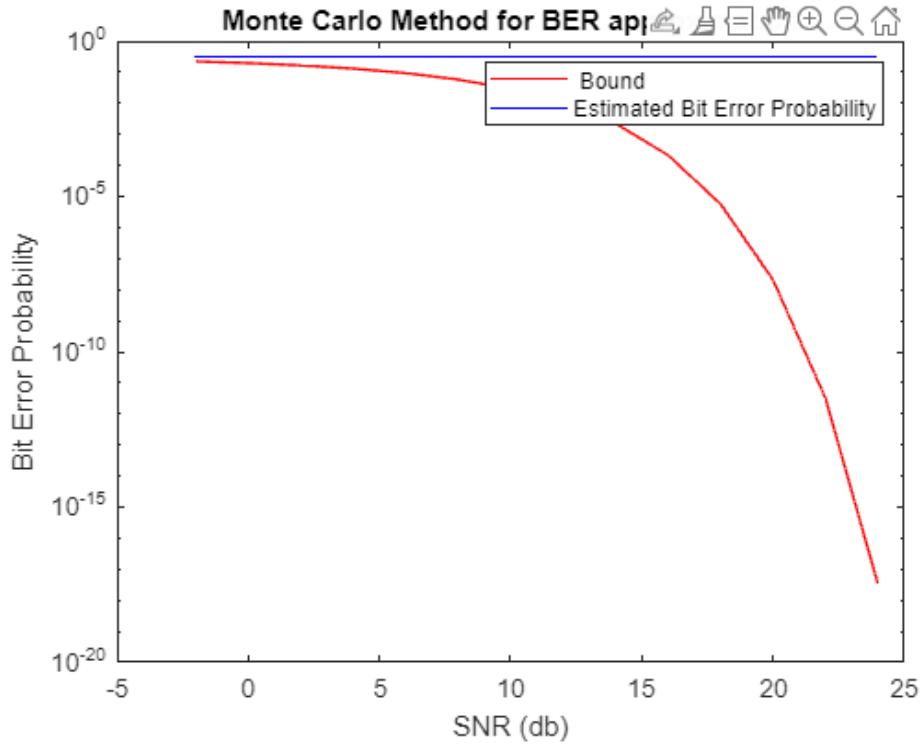
```
function [ num_of_bit_errors ] = bit_errors(est_bit_seq,b )
%Compares the bit sequence of input signal and the estimated bit sequence of output
%       est_bit_seq:        Output Binary 4-bit Sequence
%       b:                          Input Binary 4-bit Sequence
%Returns total number of mismatches found

%Number or triads
N=length(b);
num_of_bit_errors=0;
%Repeat for all triads
for i=1:N
    %There is a mismatch. Increase total errors
    if b(i,1)~=est_bit_seq(i,1) || b(i,2)~=est_bit_seq(i,2) || b(i,3)~=est_bit_seq(i,3) || b(i,4)~=est_bit_seq(i,4)
        num_of_bit_errors=num_of_bit_errors+1;
    end
end
```

<u>Σχολιασμός Κώδικα</u>

Ο κώδικας της συνάρτησης **bit_errors** συγκρίνει την δυαδική ακολουθία εισόδου **b** με την εκτιμώμενη δυαδική ακολουθία εξόδου **est_bit_seq**. Υπολογίζει το πλήθος των αντιστοίχων σφαλμάτων, δηλαδή τον αριθμό των θέσεων στις οποίες οι δύο ακολουθίες διαφέρουν. Το αποτέλεσμα **num_of_bit_errors** είναι ο συνολικός αριθμός των λαθών που βρέθηκαν.

Ερώτημα Β






```
N=100;
SNR=[-2:2:24];
T=10^(-2);
over=10;
Ts=T/over;
Fs=1/Ts;
A=4;
a=1/2;
Fo=200;
K=1000;
%Create SRRC pulse
[ph,t] = srrc_pulse(T,over,A,a);

%For different values of SNR repeat experiment and calculate
%Symbol Error Probability and Bit Error Probability
for test=1:length(SNR)
    %Initalize for each repetition
    Num_Of_Symbol_Errors=0;
    Num_Of_Bits_Errors=0;

    for samples=1:K
        b = (sign (randn(N,4)) + 1)/2;

        Xn=bits_to_PSK_16(b);
        XI=Xn(:,1);
        XQ=Xn(:,2);
        %Create Xd signals for usage in Convolution
        Xdi = Fs*upsample(XI,over);
        t_Xdi =(0:Ts:N/(1/T)-Ts);
        Xdq = Fs*upsample(XQ,over);
        t_Xdq = (0:Ts:N/(1/T)-Ts);
        %Compute Convolution and create time axis. Also find length of Conv.
        t_Xti = (t(1)+t_Xdi(1):Ts:t(end)+t_Xdi(end));
        t_Xtq = (t(1)+t_Xdq(1):Ts:t(end)+t_Xdq(end));
        Xti = conv(ph,Xdi)*Ts;
        Xtq = conv(ph,Xdq)*Ts;
```

```
  %Create XI(t) and XQ(t) signals (modulation)
  XTi = 2*(Xti).*(cos(2*pi*Fo*transpose(t_Xti)));

  XTq = -2*(Xtq).*(sin(2*pi*Fo*transpose(t_Xtq)));

 %Create channel input signal
  Xt = XTi+XTq;
  %Compute variance and create Gaussian Noise
  s2w = 1/(Ts*(10^(SNR(test)/10)));
  s2n=Ts*s2w/2;
  Wt = sqrt(s2w).*randn(length(Xt),1);
  %Create Y(t) signal
  Yt = Xt+Wt;

  %Create ÕI(t) and ÕQ(t) signals( de-modulation)
  Yi_1 = Yt.*(cos(2*pi*Fo*transpose(t_Xtq)));
  Yq_1 = Yt.*(-sin(2*pi*Fo*transpose(t_Xtq)));

  %Compute Convolution and create time axis.
  Yi_2 = conv(ph,Yi_1)*Ts;
  t_Yi_2 = (t(1)+t_Xtq(1):Ts:t(end)+t_Xtq(end));
  Yq_2 = conv(ph,Yq_1)*Ts;

  %Tail cutting
  counter = 0;
  for n = 1:length(t_Yi_2)
    if t_Yi_2(n)<0
      counter = counter+1;
    end
  end
```

```
        Yi_cut = Yi_2(counter+1:(length(t_Yi_2)-(counter)));
        Yq_cut = Yq_2(counter+1:(length(t_Yi_2)-(counter)));

       %Downsampling and creating Yi,n and Yq,n
        YI = downsample(Yi_cut,over);
        YQ = downsample(Yq_cut,over);
        %Creating Yn sequence
        Yn=[YI YQ];

        %Detect 16-PSK Sequence and calculate symbol errors and bit errors
        [est_X,est_bit_seq]=detect_PSK_16(Yn);

        Num_Of_Symbol_Errors=Num_Of_Symbol_Errors+symbol_errors(est_X,Xn);
        Num_Of_Bits_Errors= Num_Of_Bits_Errors+bit_errors(est_bit_seq,b);
    end
    %Store probability of error on each sequence into matrix for every repetition
    Symbol_Errors(1,test)=Num_Of_Symbol_Errors/(N*K);
    Bits_Errors(1,test)=Num_Of_Bits_Errors/(N*K*4);

    symbol_bound(test)=2*Q(1./(sqrt(s2n))*sin(pi/16));
    %Log2(16)=4
    bit_bound(test)=symbol_bound(test)/4;
end

%Show results with semilogy
figure()
semilogy(SNR,symbol_bound,'red');
hold on;
semilogy(SNR,Symbol_Errors,'blue');
hold off;
xlabel('SNR (db)');
ylabel('Symbol Error Probability');
legend('Smart Upper Bound','Estimated Symbol Error Probability');
title('Monte Carlo Method for SER approximation');


figure()
semilogy(SNR,bit_bound,'red');
hold on;
semilogy(SNR,Bits_Errors,'blue');
hold off;
xlabel('SNR (db)');
ylabel('Bit Error Probability');
legend(' Bound','Estimated Bit Error Probability');
title('Monte Carlo Method for BER approximation');
```

Σχολιασμός Κώδικα

Στο μέρος Β, επαναλαμβάνουμε τη διαδικασία του μέρους Α για διάφορες τιμές του SNR. Κάθε επανάληψη είναι για μια συγκεκριμένη τιμή του SNR και επαναλαμβάνουμε το μέρος Α για k=1000 επαναλήψεις, ώστε να εκτιμήσουμε την πιθανότητα σφάλματος συμβόλου και την πιθανότητα σφάλματος bit. Αρχικά, δημιουργούμε μια τυχαία ακολουθία από bits (b) και στη συνέχεια αντιστοιχίζουμε αυτήν την ακολουθία σε ένα 16-PSK σήμα. Παίρνουμε τις δύο συνιστώσες που προκύπτουν και τις φιλτράρουμε με τον παλμό SRRC. Πολλαπλασιάζουμε τα φιλτραρισμένα σήματα με τον αντίστοιχο φορέα συχνότητας Fo και δημιουργούμε το σήμα εισόδου X(t) του καναλιού. Αφού το περάσουμε από τον θόρυβο, ανακτάμε τις δύο συνιστώσες του ενθόρυβου σήματος και κάνουμε αποδιαμόρφωση. Στη συνέχεια, φιλτράρουμε το σήμα με τον SRRC παλμό και αποκόπτουμε τις "ουρές" που εμφανίζονται ώστε να γίνει σωστή η δειγματοληψία. Κάνουμε δειγματοληψία και υπολογίζουμε την ακολουθία εξόδου Yn. Έπειτα, με τη χρήση της συνάρτησης detect_PSK_16, εξάγουμε την ακολουθία από σύμβολα και την ακολουθία από bits και στη συνέχεια υπολογίζουμε τα σφάλματα που προέκυψαν για κάθε ακολουθία και τα προσθέτουμε στα αρχικά ώστε να έχουμε τον συνολικό αριθμό. Υπολογίζουμε την αντίστοιχη πιθανότητα σφάλματος σύμφωνα με τους τύπους που γνωρίζουμε από τη θεωρία:

$$\hat{P}(\mathrm{E}_{\mathrm{symbol}}) = \frac{\text{συνολικό πλήθος σφαλμάτων απόφασης συμβόλου}}{\text{συνολικό πλήθος απεσταλμένων συμβόλων}},$$

$$\hat{P}(\mathrm{E}_{\mathrm{bit}}) = \frac{\text{συνολικό πλήθος σφαλμάτων απόφασης bit}}{\text{συνολικό πλήθος απεσταλμένων bits}}.$$

Τέλος για κάθε τιμή του SNR υπολογίζουμε το έξυπνο άνω φράγμα για την πιθανότητα σφάλματος συμβόλου και το άνω φράγμα για τη πιθανότητα σφάλματος bit. Για τον υπολογισμό χρειάστηκε η συνάρτηση Q() η οποία μας δόθηκε υλοποιημένη. Σχεδιάζουμε σε ένα κοινό semilogy τα αντίστοιχα πειραματικά και θεωρητικά δεδομένα.

<u>Παρατηρήσεις Αποτελεσμάτων</u>

Είναι φανερό πως η απόκλιση στα δύο σήματα είναι τεράστια μεταξύ της πειραματικής και της θεωρητικής καμπύλης. Η θεωρητική μας καμπύλη φαίνεται να κινείται σε φυσιολογικά πλαίσια σε αντίθεση με αυτή της πειραματικής η οποία μοιάζει με ευθεία γραμμή. Αυτό οφείλεται σε κάποιο λάθος του κώδικα κατά των υπολογισμό των λαθών. Να τονίσουμε εδώ πως αν όλα πήγαιναν καλά οι αποκλίσεις μεταξύ των δύο καμπυλών θα έπρεπε να είναι ελάχιστες. Τέλος όπως γνωρίζουμε από την θεωρία, όσο αυξάνεται το SNR θα περιμέναμε μικρότερη πιθανότητα σφάλματος.