ΗΜΜΥ ΕΜΠ

6ο Εξάμηνο

Ψηφιακές Επικοινωνίες Ι

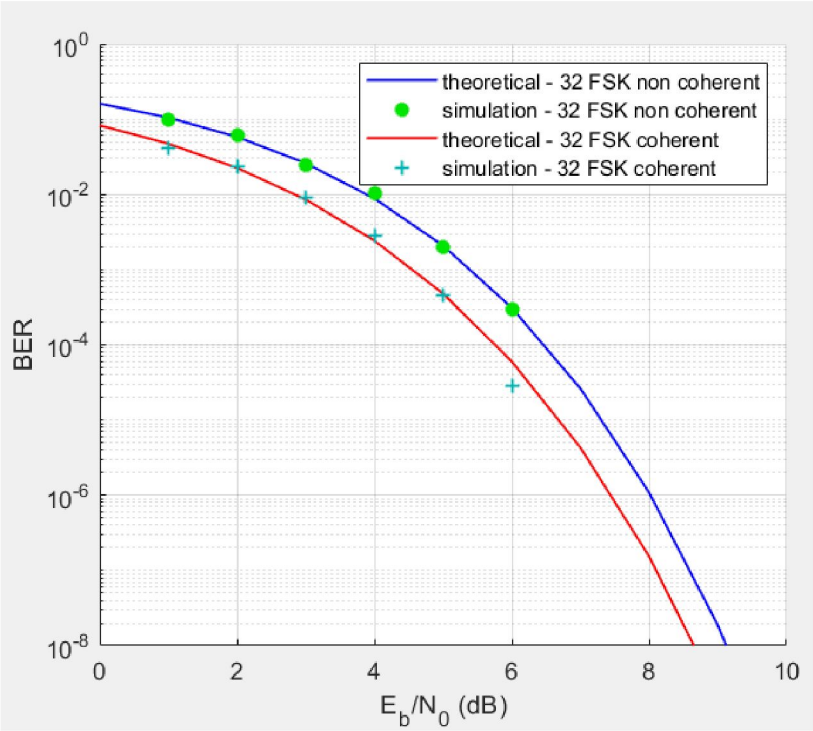
Lab 6

1. Ο δέκτης για την ασύμφωνη FSK εξομοιώνεται ως εξής:

```
%% FSK receiver
% non coherent demodulation
xr=[];
for k=1:length(s)/ns
  tk=(k-1)*T+tks;
  sk=s((k-1)*ns+1:k*ns);
  smi=[];
  for i=1:M
    th=rand()*pi;
    si=sin(2*pi*(f(i)*tk+th));
    sq=cos(2*pi*(f(i)*tk+th));
    smi=sum(sk.*si);
    smq=sum(sk.*sq);
    sm(i)=sqrt(smi^2+smq^2);
  end
  [m,j]=max(sm);
  xr=[xr;de2bi(j-1,bps)];
end
```

Δημιουργούμε τυχαία φάση στο διάστημα (0,π). Αν χρησιμοποιήσουμε απλό συσχετιστή όπως στη σύμφωνη FSK προκύπτουν πολλά λάθη, καθώς η τυχαία φάση επηρεάζει το αποτέλεσμα, οπότε υλοποιούμε ορθογωνικό δέκτη. Αρχικά πολλαπλασιάζουμε το λαμβανόμενο σήμα με ημίτονο και συνημίτονο που έχουν την ίδια φάση και δημιουργούμε τις δύο συνιστώσες `smi` και `smq`, τις οποίες μετά τετραγωνίζουμε και αθροίζουμε. Η ποσότητα `sm(i)=sqrt(smi^2+smq^2)` δεν εξαρτάται από τη φάση, και είναι μέγιστη για τη σωστή συχνότητα λόγω της ορθογωνιότητας.

2. Μέσω bertool σχεδιάζουμε τις καμπύλες Pb<->EbNo και βλέπουμε ότι η σύμφωνη αποδιαμόρφωση είναι καλύτερη, όπως αναμέναμε από τους γνωστούς τύπους για το άνω φράγμα στο Pe. Σημειώνεται ότι η προσομοίωση αργεί αρκετά.



Κώδικας για την άσκηση 2:

Αρχεία *fsk_errors_noncoherent.m* και *fsk_errors_coherent.m* (δίνεται έτοιμο).

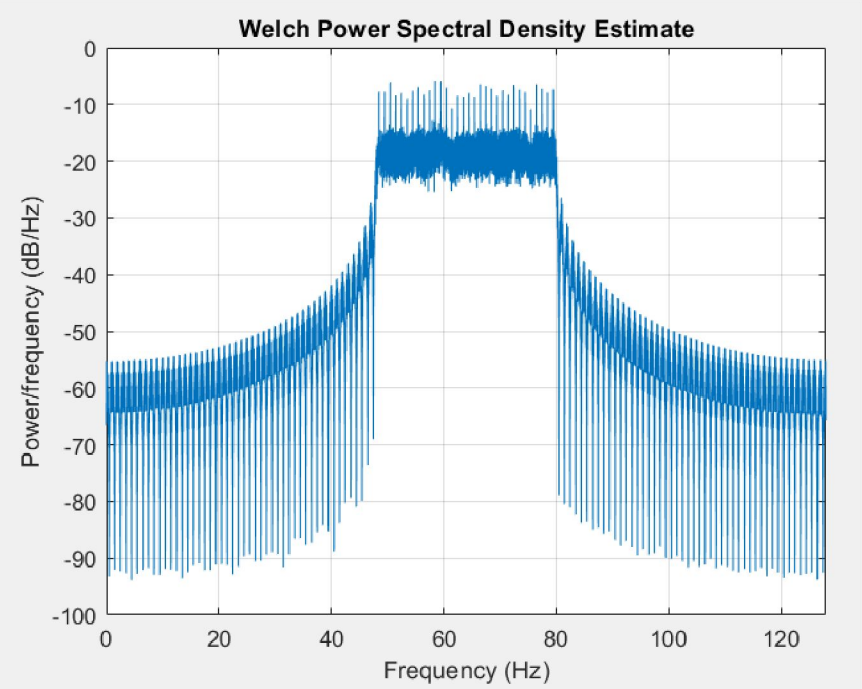
Για την προσομοίωση μέσω bertool καλούμε τα αρχεία *ask_ber_func_fsk_noncoherent.m* και *ask_ber_func_fsk_coherent.m*. Το καθένα από αυτά ρυθμίζει τις παραμέτρους και καλεί την αντίστοιχη συνάρτηση fsk_errors. Παρατήρησα ότι για να βγει σωστό αποτέλεσμα στην ασύμφωνη αποδιαμόρφωση χρειάστηκε αρκετά μεγάλο nsamp.

Συνάρτηση *fsk_errors_noncoherent.m*:

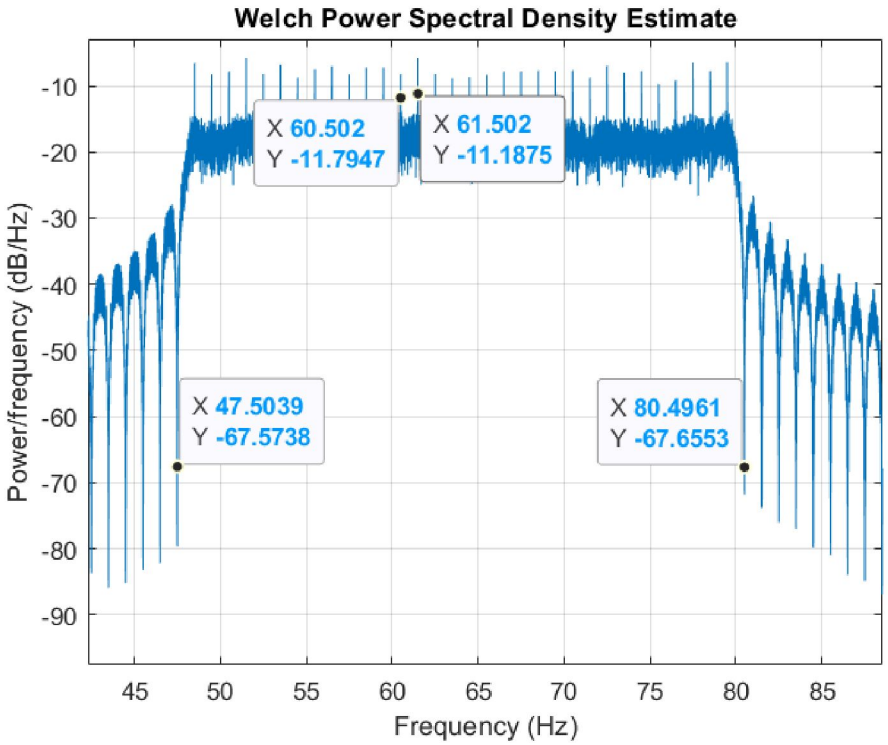
```
function errors=fsk_errors_noncoherent(bps,Nsymb,ns,EbNo)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Input parameters
% bps: bits per symbol, Nsymb: numb of simulated symbols
% ns: number of samples per symbol (oversampling)
% EbNo: normalized signal-to-noise ratio, in db
M=2^bps; % number of different symbols
BR=1; % Baud Rate
fc=2*M*BR; % RF frequency
%% Derived parameters
nb=bps*Nsymb; % number of simulated data bits
T=1/BR; % one symbol period
```

```
Ts=T/ns; % oversampling period
% M frequencies in "non-coherent" distance (BR)
f=fc+BR*((1:M)-(M+1)/2);
% awgn channel
SNR=EbNo+10*log10(bps)-10*log10(ns/2); % in db
% input data bits
y=randi([0,1],1,nb); %
x=reshape(y,bps,length(y)/bps)';
t=[0:T:length(x(:,1))*T]'; % time vector on the T grid
tks=[0:Ts:T-Ts]';
%% FSK signal
s=[];
A=sqrt(2/T/ns);
for k=1:length(x(:,1))
 fk=f(bi2de(x(k,:))+1);
 tk=(k-1)*T+tks;
 s=[s; sin(2*pi*fk*tk)];
end
% figure(1); pwelch(s);
% add noise to the FSK (passband) signal
s=awgn(s,SNR, 'measured');
%% FSK receiver
% non coherent demodulation
xr=[];
for k=1:length(s)/ns
  tk=(k-1)*T+tks;
  sk=s((k-1)*ns+1:k*ns);
  smi=[];
  for i=1:M
    th=rand()*pi;
    si=sin(2*pi*(f(i)*tk+th));
    sq=cos(2*pi*(f(i)*tk+th));
    smi=sum(sk.*si);
    smq=sum(sk.*sq);
    sm(i)=sqrt(smi^2+smq^2);
  end
  [m,j]=max(sm);
  xr=[xr;de2bi(j-1,bps)];
end
% count errors
err=not(x==xr);
errors=sum(sum(err));
end
```

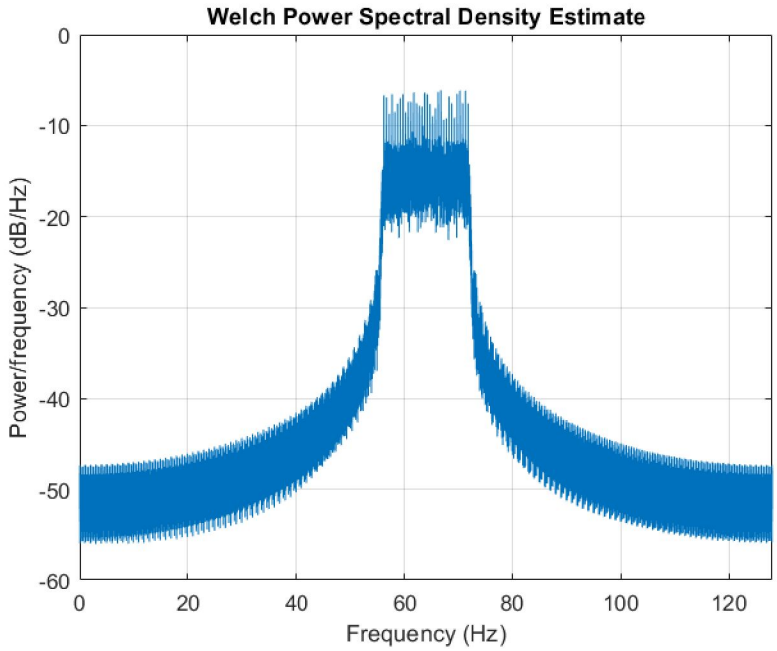
3. Φάσμα ασύμφωνης FSK:



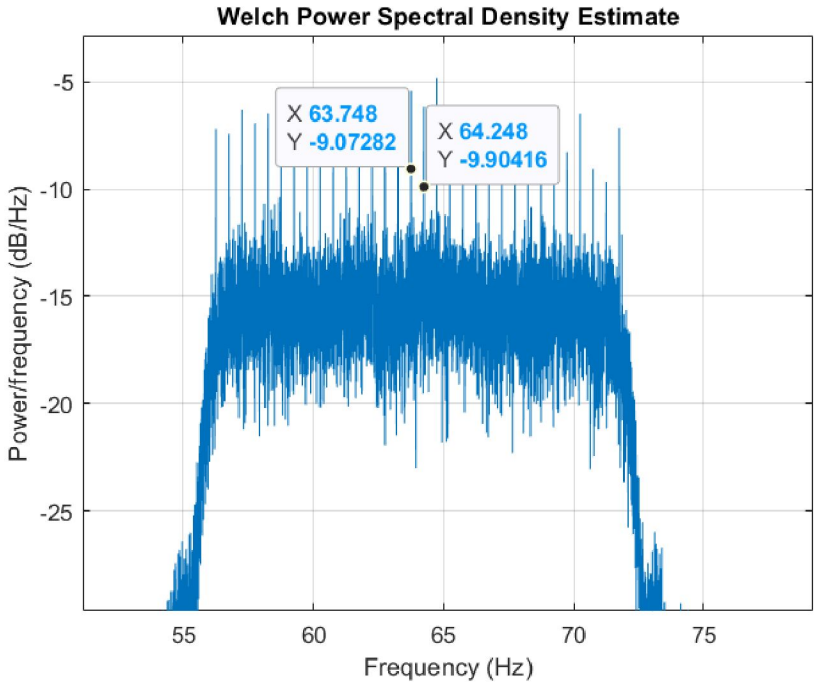
Από το φάσμα του σήματος βλέπουμε τις 32 συχνότητες της 32-FSK. Απέχουν 1/T=1Hz μεταξύ τους και το συνολικό εύρος ζώνης είναι 33Hz, γύρω από τα 64Hz. Zoom in:



Φάσμα σύμφωνης 32-FSK:



Στη σύμφωνη μπορούμε να πετύχουμε το μισό εύρος ζώνης αφού οι συχνότητες μπορούν να απέχουν 1/2T=0.5Hz μεταξύ τους:



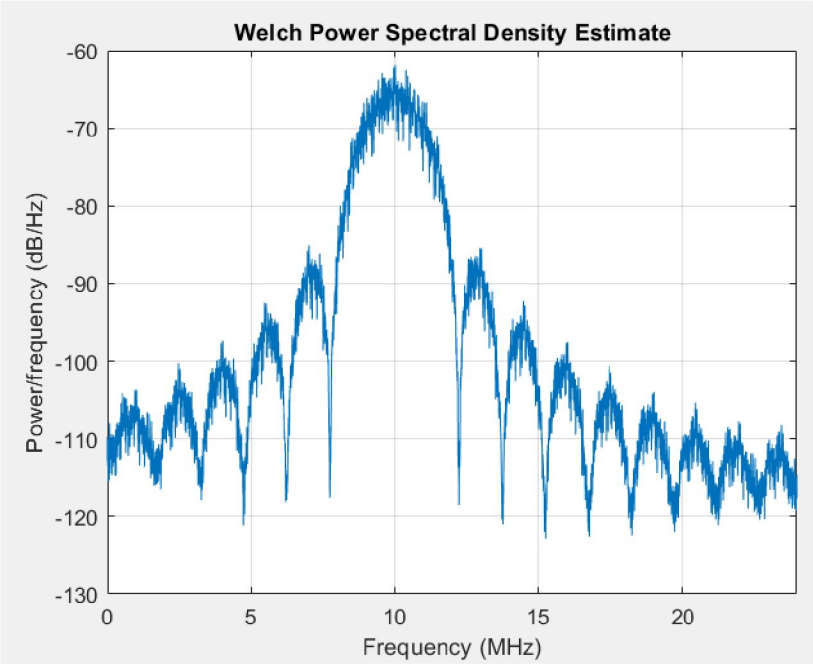
Κώδικας για άσκηση 3: αρχείο *ask_3.m*

```
%% Input parameters
% bps: bits per symbol, Nsymb: numb of simulated symbols
% ns: number of samples per symbol (oversampling)
% EbNo: normalized signal-to-noise ratio, in db
clear all; close all; clc;
bps=5; Nsymb=2000; ns=256; EbNo=6;
M=2^bps; % number of different symbols
BR=1; % Baud Rate
fc=2*M*BR; % RF frequency
%% Derived parameters
nb=bps*Nsymb; % number of simulated data bits
T=1/BR; % one symbol period
Ts=T/ns; % oversampling period
% M frequencies in "non-coherent" distance (BR)
fnon=fc+BR*((1:M)-(M+1)/2);
% M frequencies in "non-coherent" distance (BR)
fnon=fc+BR*((1:M)-(M+1)/2);
% input data bits
y=randi([0,1],1,nb); %
x=reshape(y,bps,length(y)/bps)';
t=[0:T:length(x(:,1))*T]'; % time vector on the T grid
tks=[0:Ts:T-Ts]';
%% FSK signal
% non coherent
s=[];
A=sqrt(2/T/ns);
for k=1:length(x(:,1))
 fk=fnon(bi2de(x(k,:))+1);
 tk=(k-1)*T+tks;
 s=[s; sin(2*pi*fk*tk)];
end
figure(1); pwelch(s,[],[],[],ns);

% coherent
s=[];
A=sqrt(2/T/ns);
for k=1:length(x(:,1))
 fk=fcoh(bi2de(x(k,:))+1);
 tk=(k-1)*T+tks;
 s=[s; sin(2*pi*fk*tk)];
end
figure(2); pwelch(s,[],[],[],ns);
```

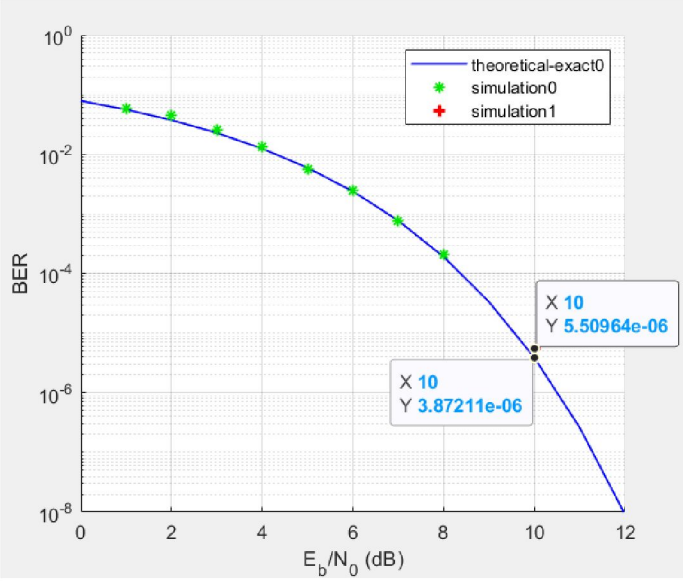
4. Θέτουμε `R=3*10^6` και `fc=10/3*R`.

Από το αρχείο *ask_4_spectrum.m* παίρνουμε τη γραφική παράσταση του φάσματος:



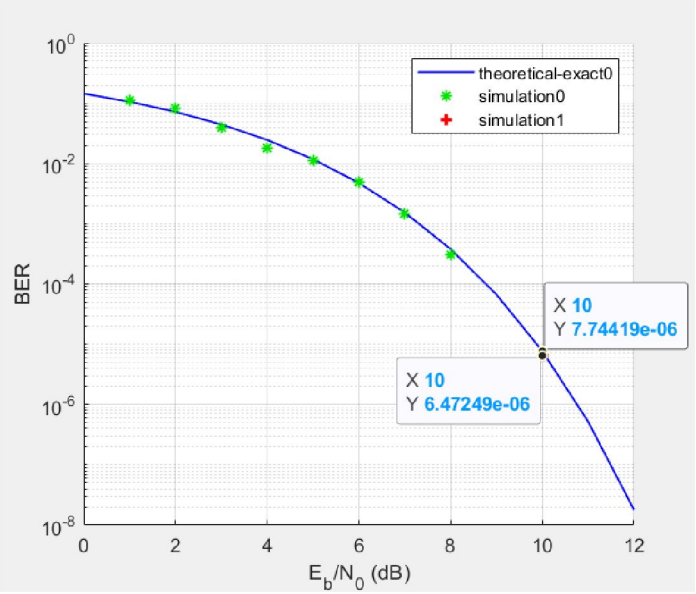
Είναι ζωνοπερατό γύρω από τη συχνότητα 10 MHz.

Εξομοίωση με προκωδικοποίηση:



Επειδή αργούσε πολύ να τρέξει έκανα μια προσομοίωση για μέχρι Eb/No=8dB και μια για να βρω το σημείο με Eb/No=10dB. Το BER προέκυψε θεωρητικά $3.87 \cdot 10^{-6}$ και πειραματικά $5.51 \cdot 10^{-6}$.

Χωρίς προκωδικοποίηση:



Για Eb/No=10dB το BER προέκυψε θεωρητικά $7.74 \cdot 10^{-6}$ και πειραματικά $6.47 \cdot 10^{-6}$.

Η πιθανότητα λάθους είναι διπλάσια στην περίπτωση χωρίς προκωδικοποίηση, καθώς αν γίνει λάθος για ένα ψηφίο γίνεται λάθος και στο επόμενο. Με την προκωδικοποίηση αυτό αποφεύγεται.

Κώδικας για την άσκηση 4:

Όλα τα αρχεία κώδικα για την άσκηση 4 βασίζονται στον κώδικα 6.2 με διαφορετική τιμή στο R και στο fc.

Για τη γραφική παράσταση του φάσματος εκτελούμε το αρχείο *ask_4_spectrum.m* Το αρχείο αυτό περιέχει τον ίδιο κώδικα με τον δοσμένο κώδικα 6.2 μέχρι και τον πομπό, με επιπλέον γραμμή για τη γραφική παράσταση:

```
figure;pwelch(s,[],[],fc,1/Ts);
```

Για την πλήρη εξομοίωση χρησιμοποιούμε το αρχείο *ask_ber_func_msk.m* με το bertool. Στις γραμμές 20 και 21 του αρχείου αυτού, ανάλογα με ποια βρίσκεται σε σχόλιο, μπορούμε να ρυθμίσουμε αν θα καλέσουμε τη συνάρτηση *msk_errors_precoding.m* ή την *msk_errors_no_precoding.m*, για την περίπτωση της προκωδικοποίησης ή όχι αντίστοιχα.

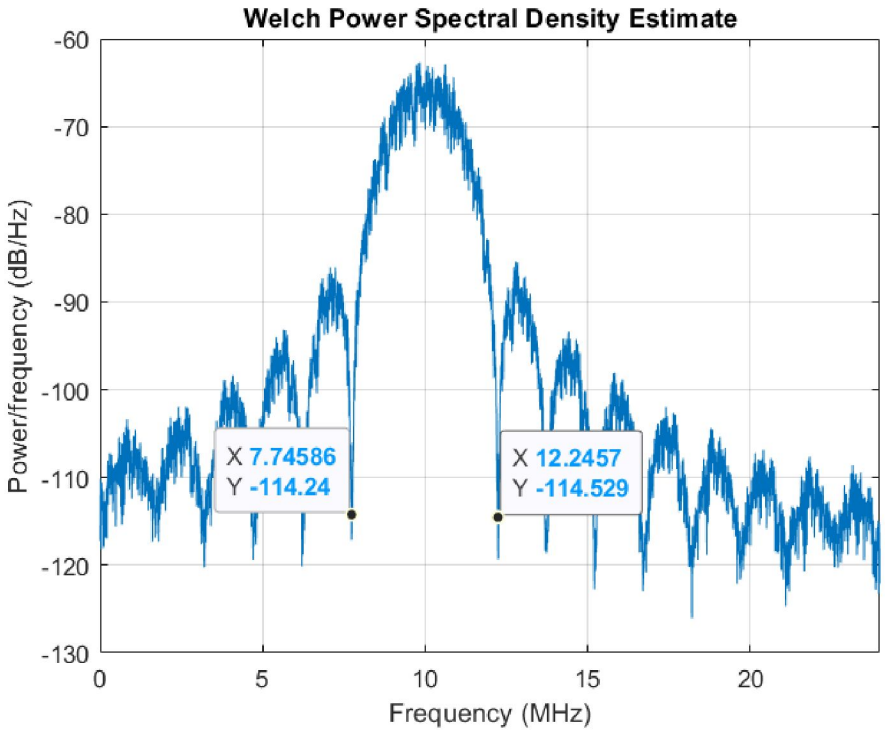
Οι συναρτήσεις *msk_errors_no_precoding.m* και *msk_errors_precoding.m* είναι ίδιες με τον δοσμένο κώδικα 6.2, αλλά έχω διαχωρίσει τις περιπτώσεις κωδικοποίηση ή όχι ρυθμίζοντας τις τελευταίες γραμμές κατάλληλα.

5. Βλέπουμε από το bertool και τους γνωστούς τύπους ότι το MSK με προκωδικοποίηση έχει την ίδια καμπύλη BER με το QPSK.

Η πιθανότητα λάθους για Eb/No=10dB είναι $3.87 \cdot 10^{-6}$.

Βρίσκουμε το Baud rate από τον τύπο $R = \frac{\log_2 M}{T} \Rightarrow \frac{1}{T} = \frac{3}{2} MHz = 1.5MHz.$

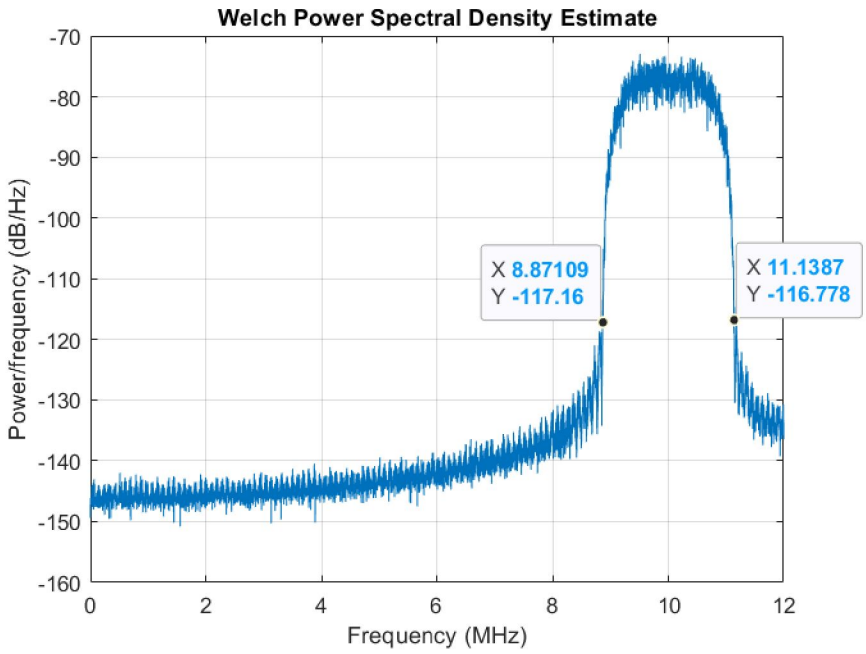
Το σύστημα MSK έχει εύρος ζώνης $\frac{3}{T} = 4.5MHz$ σαν το BFSK:



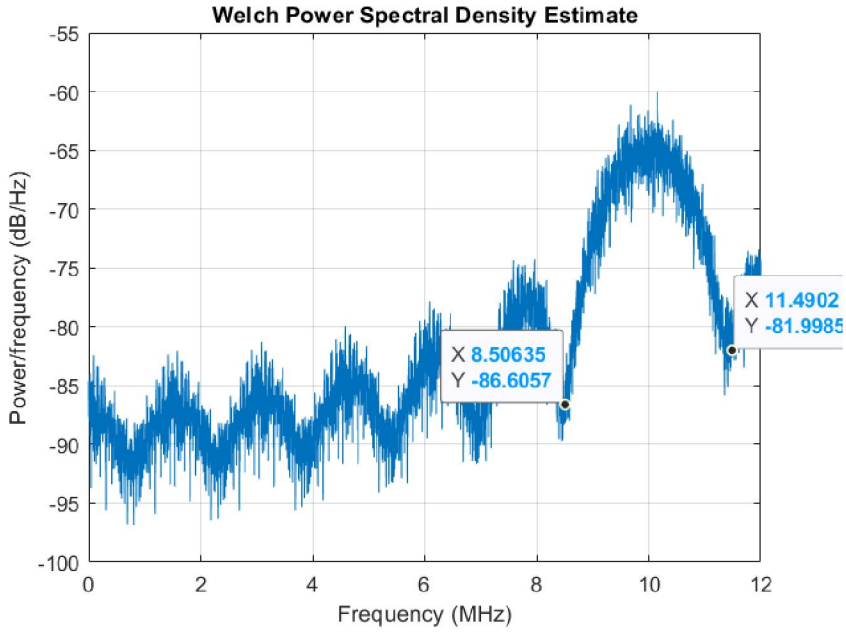
Εξομοιώνουμε σύστημα QPSK με φίλτρο Nyquist $a = 0.5$.

Στο σύστημα QPSK με ζωνοπερατό Nyquist το εύρος ζώνης προκύπτει από τον τύπο:

$$\frac{R}{W} = \frac{\log_2 M}{1+a} \Rightarrow W = \frac{3}{2}(1+0.5) = 2.25MHz$$



Χρησιμοποιώντας ορθογωνικό φίλτρο το εύρος ζώνης είναι $\frac{2}{T} = 3MHz$:



Παρατηρούμε λοιπόν ότι παρόλο που έχουμε την ίδια πιθανότητα λάθους, το QPSK έχει μικρότερες απαιτήσεις σε εύρος ζώνης.

Κώδικας για άσκηση 5: *αρχείο ask_5.m*

Στη γραμμή 33 επιλέγουμε ποιο είδους φίλτρο θα χρησιμοποιήσουμε (ορθογωνικό ή Nyquist)

```
close all; clear all; clc;
k=2; M=2^k; Nsymb=3000; nsamp=16; EbNo=10;

%% Grey encoding vector
ph1=[pi/4];
theta=[ph1; -ph1; pi-ph1; -pi+ph1];
mapping=exp(1j*theta);
if(k>2)
 for j=3:k
 theta=theta/2;
 mapping=exp(1j*theta);
 mapping=[mapping; -conj(mapping)];
 theta=angle(mapping);
 end
end

%% Random bits -> symbols
x=floor(2*rand(k*Nsymb,1));
xsym=bi2de(reshape(x,k,length(x)/k).','left-msb')';
y=[];
for i=1:length(xsym)
 y=[y mapping(xsym(i)+1)];
end

%% Filter parametres
delay=8;
rolloff=1/3;
filtorder = delay*nsamp*2;
rNyquist= rcosine(1,nsamp,'fir/sqrt',rolloff,delay);
rect=ones(1,nsamp);
% % pick filter
filter=rNyquist;
% filter=rect;

%% Transmitter
y=upsample(y,nsamp);
ytx = conv(y,filter);
R=3*10^6; % bit rate
Fs=R/k*nsamp;
fc=10/1.5; % carrier frequency in multiples of baud rate (1/T=1.5MHz)
m=(1:length(ytx));
s=real(ytx.*exp(1j*2*pi*fc*m/nsamp));  % shift to desired frequency
band
%plots
% figure; pwelch(real(ytx),[],[],[],Fs); %before frequency shifting
figure; pwelch(s,[],[],[],Fs);
```