NTNU – Trondheim
Norwegian University of
Science and Technology

Department of Electronics and Telecommunications

TTT4115 Communication Theory
Term exercise 1 spring 2015

Helge Langen, Lars-Arne Larsen

Contents

1. Problem 1

1.1. Problem 1a

From the z-transform of the described filter in Problem 1 we get:

$$X(z) = \rho X(z)z^{-1} + \sqrt{1-\rho^2} * E(z) \tag{1.1}$$

If we sort equation 1.1 in regard of $\frac{X(z)}{E(z)}$ we get:

$$\frac{X(z)}{E(z)} = T(z) = \frac{\sqrt{1-\rho^2}}{1-\rho z^{-1}}$$

Make the z-transform back to the diskrete plane

$$t(n) = \sqrt{1-\rho^2}\rho^n u(n)$$

Since the noise is uncorrolated, we may reduce the convolution to find x(n)

$$x(n) = t(n) \circledast e(n) = t(n)\sigma_W^2$$

From Problem 1 it is given that

$$\sigma_W^2 = 1$$

This results in equation 1.2.

$$x(n) = \sqrt{1-\rho^2}\rho^n u(n) \tag{1.2}$$

From the output calculated in equation 1.2 we may use the following equation 1.3 from the compendium to calculate the autocorrelation:

$$R_x(\tau) = \sum_{n=-\infty}^{\infty} E\{x(n)\}E\{x(n+\tau)\} \tag{1.3}$$

By inserting our definition of x(n) we get equation 1.4.

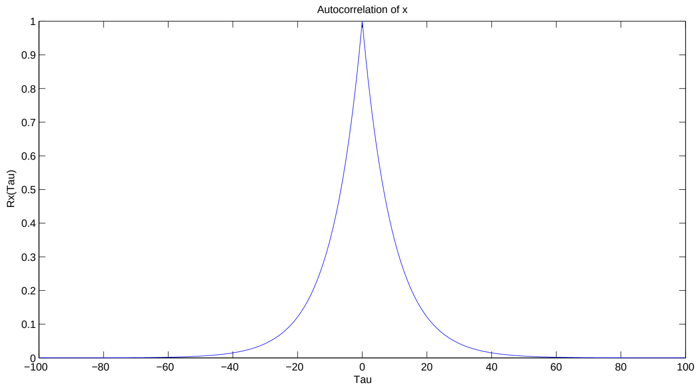
$$R_x(\tau) = (1-\rho^2)\sum_{n=0}^{\infty} \rho^n \rho^{n+\tau} \tag{1.4}$$

By using the formula for geometric series we get equation 1.5

$$R_x(\tau) = (1-\rho^2)\rho^\tau \sum_{n=0}^{\infty} \rho^{2n} = (1-\rho^2)\rho^\tau \frac{1}{1-\rho^2} \tag{1.5}$$

After shorting equation 1.5 we get equation 1.6 that is shown in figure 1.1.

$$R_x(\tau) = \rho^{|\tau|} \tag{1.6}$$



1.2. Problem 1b

The power spectral density may be given by equation 1.7 where T(f) is the AR filter process:

$$[H]Sx(f) = |T(f)|^2 = T(f)T^*(f) \tag{1.7}$$

Insert the expression from equation 1.1 and calculate:

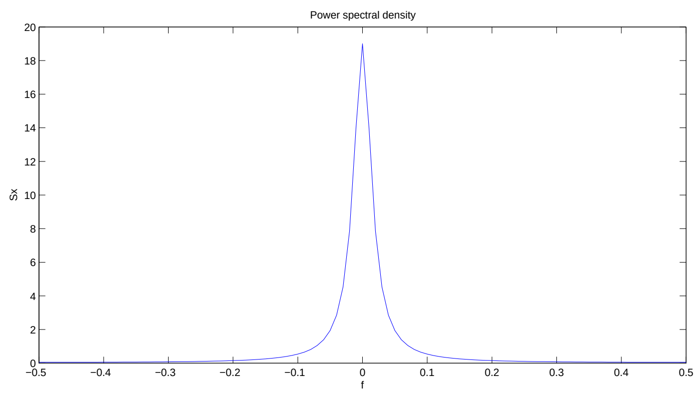
$$[H]T(f) = T(z)|_{z=e^{j2\pi f}} = \frac{\sqrt{1-\rho^2}}{1-\rho e^{-j2\pi f}}$$

Expanding the denominator and cleaning up.

$$[H]Sx(f) = \frac{\sqrt{1-\rho^2}^2}{(1-\rho e^{-j2\pi f})(1-\rho e^{j2\pi f})} = \frac{1-\rho^2}{1-\rho e^{j2\pi f}-\rho e^{-j2\pi f}+\rho^2}$$

Using eulers formula for cosine function to reduce to equation 1.8 and plot the result in figure 1.2.

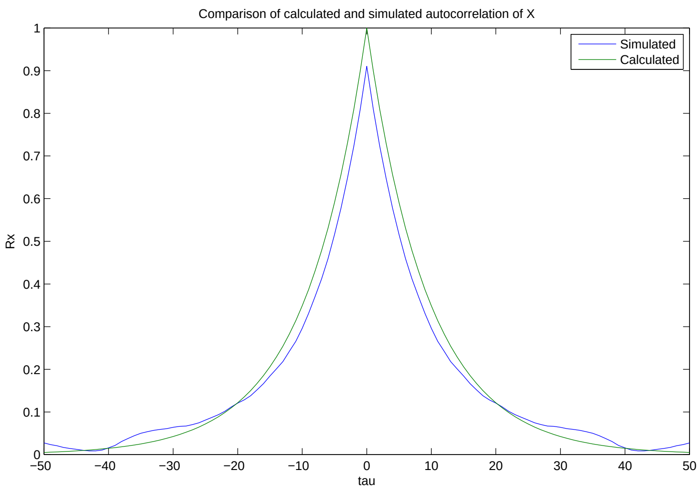
$$[H]Sx(f) = \frac{1-\rho^2}{1+\rho^2-2\rho cos(2\pi f)} \tag{1.8}$$



1.3. Problem 1c

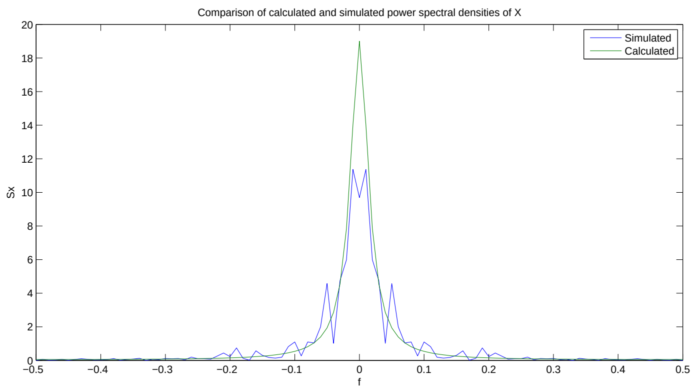
The matlab code for this function is placed in appendix A.3.

1.4. Problem 1d



As seen in figure 1.4, the simulated and calculated autocorrelations are similar. Some variation may be seen between each run of the script.

1.5. Problem 1e



When calculating the fft of the output, we had to use a limited amount of samples to achieve normal results. This was to limit the amount of noise in the plot. Again, there will be some variation between each run of the script, but after some runs it became apparent that the result were approved.

2. Problem 2

To shorten the length of this report, the code for problem 2 have been put together instead of having seperate code for a, b and c. All plots made in normalized frequency have been from 0 to $\frac{1}{2}$ since they are real functions. Since the functions are real, the frequency plot is even, and easier to interpret this way.

2.1. Problem 2a

Equation 2.1 for bitrate was given in the exercise

$$H = \frac{1}{2}log(2\pi e^1 \frac{\sigma_U^2}{\Delta^2}) \tag{2.1}$$

Solved equation 2.1 for Δ^2 for use in later equations

$$\Delta^2 = \frac{2\pi e^1 \sigma_U^2}{2^{2H}} \tag{2.2}$$

Equation 2.3 is equation (3.15) from the compendium and is used in the process of calculating the optimal filters.

$$\sqrt{\lambda} = \frac{\int\limits_{-\infty}^{\infty} \sqrt{S_x(f)S_Q(f)}df}{P + \sigma_Q^2} \tag{2.3}$$

For the calculation of the lagrange multiplier, equation 2.4 is given in the exercise:

$$\sigma_Q^2 = \frac{\Delta^2}{12} \tag{2.4}$$

The noise is constant over the frequency-band, thus applying

$$S_N(f) = \sigma_Q^2$$

All inserted in equation 2.3 gives equation 2.5.

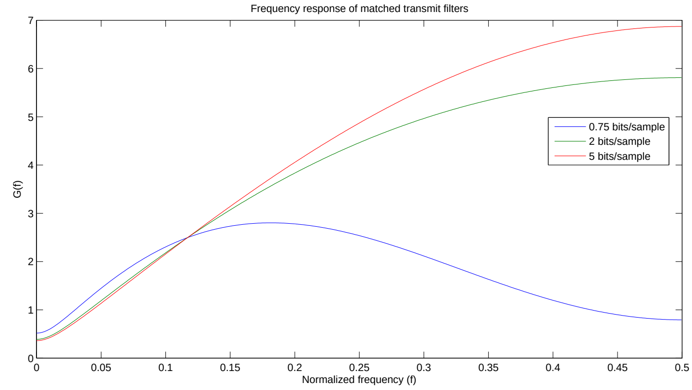
$$\sqrt{\lambda} = \frac{\int\limits_{-\frac{1}{2}}^{\frac{1}{2}} \sqrt{\frac{0.19\sigma_Q^2}{1.81-1.8cos(2\pi f)}}df}{1 + \sigma_Q^2} \tag{2.5}$$

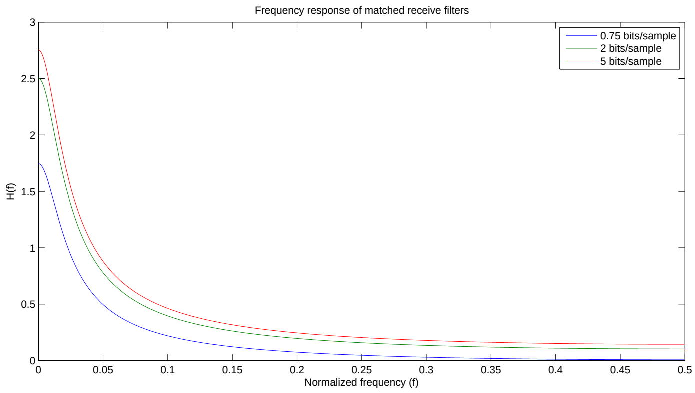
The integral in equation 2.5 had to be solved numerically using matlab. The calculated functions and values are used in equation 2.6 and 2.7 to calculate the optimal transmitter/receiver filters.

$$|H(f)|^2 = \sqrt{\frac{\lambda S_x(f)}{S_N(f)}} - \lambda \tag{2.6}$$

$$|G(f)|^2 = \sqrt{\frac{S_N(f)}{\lambda S_x(f)}} - \frac{S_N(f)}{S_x(f)} \tag{2.7}$$

The resulting frequency responses for the filters are shown in figure 2.1 and 2.1.

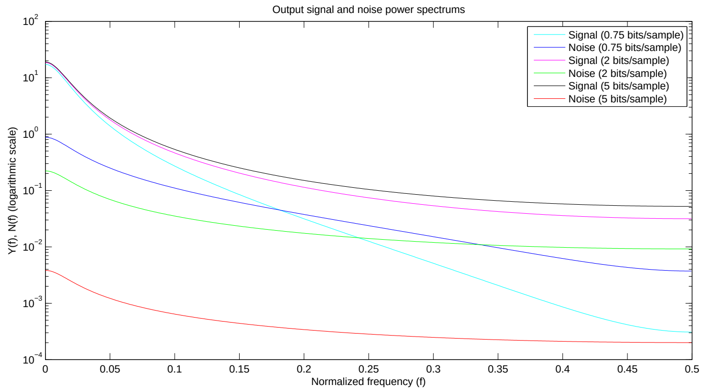




To calculate the power spectral densities of the output noise and signal component, they where first calculated seperatly by using equation 2.8 and 2.9. The results are shown in figure 2.1

$$N(f) = \sigma_Q^2 |H(f)|^2 \tag{2.8}$$

$$Y(f) = S_x(f)|G(f)|^2|H(f)|^2 \tag{2.9}$$



To calculate the SNR, spectras shown in figure 2.1 si used as shown in equation 2.10

$$SNR = 10log\left(\frac{\int_{-\frac{1}{2}}^{\frac{1}{2}} Y(f)df}{\int_{-\frac{1}{2}}^{\frac{1}{2}} N(f)df}\right) \tag{2.10}$$

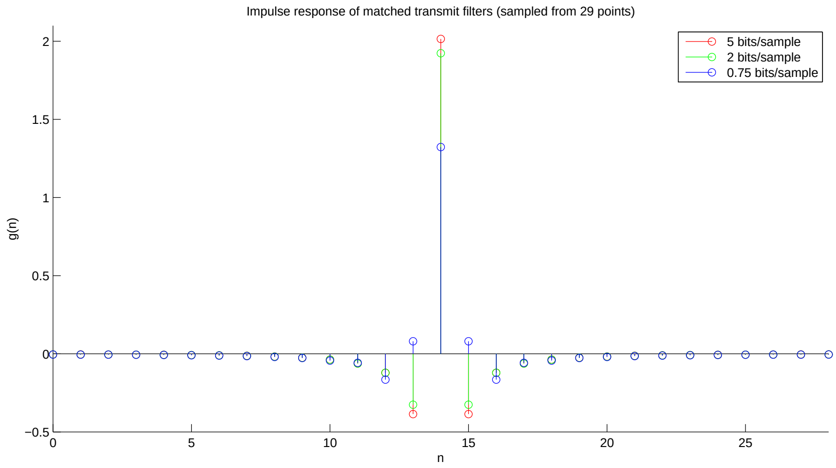
The SNR for the different values of bitrates is shown in table 2.1

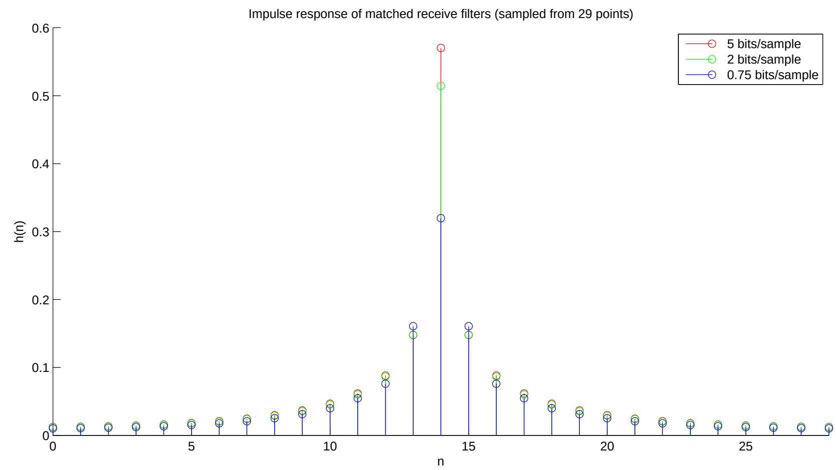
H	SNR[dB]
0.75	9.45
2	14.99
5	32.60

Table 2.1.: Shows SNR values for different bitrates.

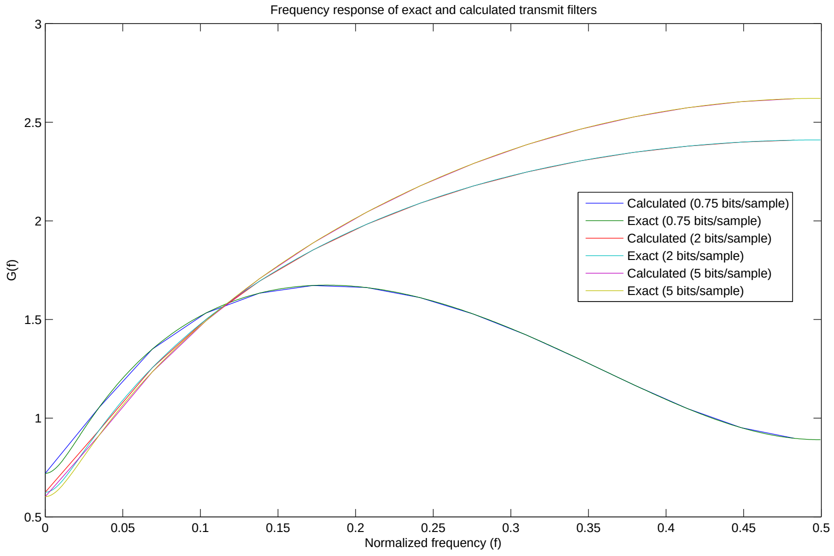
2.2. Problem 2b

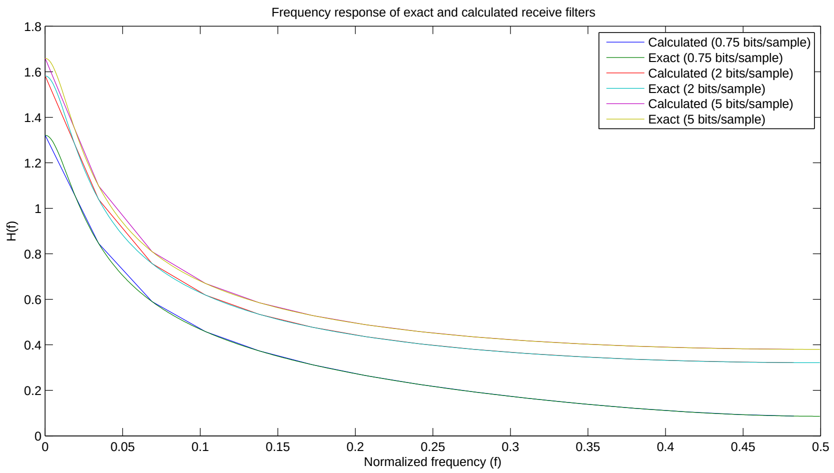
The function FrSamp() that was given in the exercise was used to make an inverse fft of the receive and transmit filters. The inverse fft of a frequency respones will be the same as the impulse response to the given function. Higher resolution of the inverse fft will increase the resolution of the impulse response. The impulse response of the receive and transmit filters are shown in figure 2.2 and 2.2.





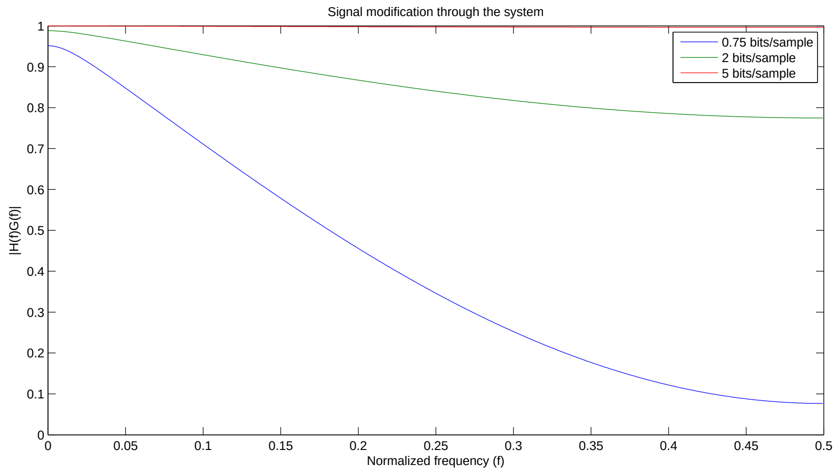
To be able to compare the calculated and the exact filters, an fft is done on the impulse responses shown in figure 2.2 and 2.2. The comparison of exact and calculated filters are shown in figure





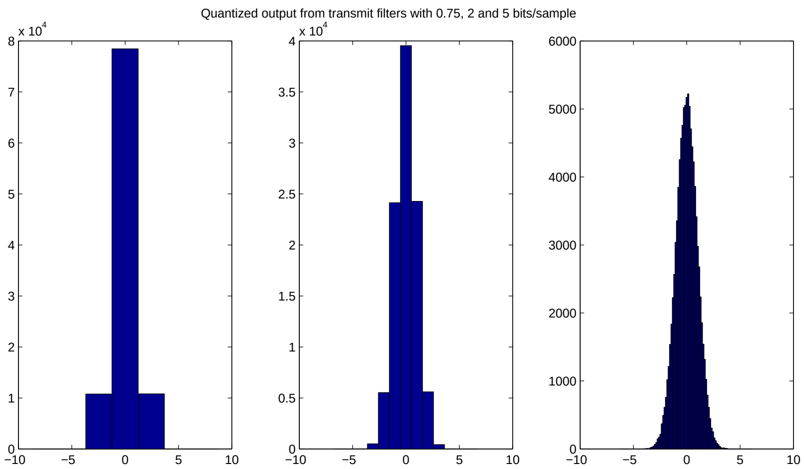
To show how the signal is modulated through the system, equation 3.19 from the compendium is used. The equation is shown in equation 2.11 and the result is plotted in figure 2.2.

$$|H(f)G(f)| = 1 - \sqrt{\lambda \frac{S_N(f)}{S_x(f)}} \tag{2.11}$$



2.3. Problem 2c

To measure the bitrate, we had to decide step size from equation 2.4 and use this to decide decision limits and representation levels. Then the output signal from G(f) where sorted in the different representation levels and shown in figure 2.3.



The upper and lower quantization limits were decided from figure 2.3 so that we did not trunkate any levels.

The definition for bitrate is given in the exercise and is shown in equation 2.12 where P_i is the probability of getting a given value. The results from calculation is given in table 2.3.

$$H = -\sum_{i=1}^{I} P_i log_2(P_i) \tag{2.12}$$

H	SNR calculated[dB]	SNR simulated[dB]	H simulated
0.75	9.45	7.49	0.97
2	14.99	14.78	2.06
5	32.60	29.69	4.99

As seen in table 2.3, the simulated bitrate deviates from the given one for low bitrates. This result complies with the exercise text where it is stated that the equation 2.1 is for high-rate systems.

We noted that the SNR would improve if length of filters are increased.

Appendix A. Appendix

A.1. Exercise 1a

```
clear all

tau=[-100:100];
rho=0.9;

Rx=rho.^(abs(tau));
plot(tau,Rx)
title('Autocorrelation of x')
xlabel('Tau')
ylabel('Rx(Tau)')
```

A.2. Exercise 1b

```
clear all

f=[-1/2:0.01:1/2];
rho=0.9;
Sx=(1-rho^2)./(1+rho^2-2*rho*cos(2*pi*f));
plot(f,Sx)
title('Power spectral density')
ylabel('Sx')
xlabel('f')
```

A.3. Exercise 1c

```
function [ x ] = Oppgave1c( n, rho )
%[ x ] = Oppgave1c( n, rho )
%Generates an AR-function from the filter H(z)=(1-rho^2)/(1-rho*z^-1) of
%length n.

b=sqrt(1-rho^2);
a=[1 -rho];
e=randn(1,n);
x=filter(b,a,e);

end
```

A.4. Exercise 1d

```
clear all

n=10000;
rho=0.9;
tau=[   -50:50];


x=Oppgave1c(n,rho);
Rxsim=xcorr(x)./n;
Rxcalc=rho.^(abs(tau));
plot(tau,Rxsim(n-50:n+50),tau,Rxcalc)
title('Comparison of calculated and simulated autocorrelation of X')
ylabel('Rx')
xlabel('tau')
legend('Simulated','Calculated')
```

A.5. Exercise 1e

```
clear all
n=100;
rho=0.9;

f=[-1/2:1/100:1/2];
x=Oppgave1c(n,rho);
acorr=xcorr(x);
Sxsim=fftshift(fft(acorr(n-50:n+50))./n);
Sxcalc=(1-rho^2)./(1+rho^2-2*rho*cos(2*pi*f));

plot(f,abs(Sxsim),f,Sxcalc)
title('Comparison of calculated and simulated power spectral densities of X')
ylabel('Sx')
xlabel('f')
legend('Simulated','Calculated')
```

A.6. Exercise 2

```
clear all;

NL = 29; % Number of points for low resolution spectrum
NH = 999; % Number of points for high resolution spectrum
fbh = -0.5:1/NH:0.5; % bilateral half spectrum frequency vector
fuh = 0:1/NH:0.5; % unilateral half spectrum frequency vector
flf = 0:1/NL:1-1/NL; % Low resolution unilateral full spectrum frequency vector
fli = 1:ceil(length(flf)/2); % Half spectrum index vector
fhf = 0:1/NH:1-1/NH; % High resolution unilateral full spectrum frequency vector
fhi = 1:ceil(length(fhf)/2); % Half spectrum index vector

% The different bit rates to be used
H1 = 0.75;
H2 = 2;
H3 = 5;
rho = 0.9;
```

```
%------------ Exercise 2a ------------%

% Quantization noise
Deltasq1 = (2*pi*exp(1))/2^(2*H1);
Deltasq2 = (2*pi*exp(1))/2^(2*H2);
Deltasq3 = (2*pi*exp(1))/2^(2*H3);
sigmaq1 = Deltasq1/12;
sigmaq2 = Deltasq2/12;
sigmaq3 = Deltasq3/12;

% Input signal PSD
Sx = @(f) (1-rho^2)./(1+rho^2-(2*rho.*cos(2*pi.*f)));

%Kernel functions for the integrals that go into the Lagrange multiplier
IntSx1 = @(f) sqrt((sigmaq1.*Sx(f)));
IntSx2 = @(f) sqrt((sigmaq2.*Sx(f)));
IntSx3 = @(f) sqrt((sigmaq3.*Sx(f)));

% Calculate Lagrange multipliers
Lagrange1 = (integral(IntSx1,-0.5,0.5)/(1+sigmaq1))^2;
Lagrange2 = (integral(IntSx2,-0.5,0.5)/(1+sigmaq2))^2;
Lagrange3 = (integral(IntSx3,-0.5,0.5)/(1+sigmaq3))^2;

% Matched receive filters
Hfsq1 = @(f) sqrt((Lagrange1.*Sx(f))/sigmaq1)-Lagrange1;
Hfsq2 = @(f) sqrt((Lagrange2.*Sx(f))/sigmaq2)-Lagrange2;
Hfsq3 = @(f) sqrt((Lagrange3.*Sx(f))/sigmaq3)-Lagrange3;

figure(1);
plot(fuh,sqrt(Hfsq1(fuh)),fuh,sqrt(Hfsq2(fuh)),fuh,sqrt(Hfsq3(fuh)));
title('Frequency response of matched receive filters');
xlabel('Normalized frequency (f)');
ylabel('H(f)');
legend('0.75 bits/sample','2 bits/sample','5 bits/sample');

% Matched transmit filters
Gfsq1 = @(f) sqrt(sigmaq1./(Lagrange1.*Sx(f)))-(sigmaq1./Sx(f));
Gfsq2 = @(f) sqrt(sigmaq2./(Lagrange2.*Sx(f)))-(sigmaq2./Sx(f));
Gfsq3 = @(f) sqrt(sigmaq3./(Lagrange3.*Sx(f)))-(sigmaq3./Sx(f));

figure(2);
plot(fuh,sqrt(Gfsq1(fuh)),fuh,sqrt(Gfsq2(fuh)),fuh,sqrt(Gfsq3(fuh)));
title('Frequency response of matched transmit filters');
xlabel('Normalized frequency (f)');
ylabel('G(f)');
legend('0.75 bits/sample','2 bits/sample','5 bits/sample');

% Output noise PSD
Nf1 = sigmaq1.*Hfsq1(fuh);
Nf2 = sigmaq2.*Hfsq2(fuh);
Nf3 = sigmaq3.*Hfsq3(fuh);

% Output signal PSD
Yf1 = Sx(fuh).*Hfsq1(fuh).*Gfsq1(fuh);
Yf2 = Sx(fuh).*Hfsq2(fuh).*Gfsq2(fuh);
Yf3 = Sx(fuh).*Hfsq3(fuh).*Gfsq3(fuh);

figure(3);
semilogy(fuh,Yf1,'c');
hold on;
semilogy(fuh,Nf1);
```

```
semilogy(fuh,Yf2,'m');
semilogy(fuh,Nf2,'g');
semilogy(fuh,Yf3,'k');
semilogy(fuh,Nf3,'r');
title('Output signal and noise power spectrums');
xlabel('Normalized frequency (f)');
ylabel('Y(f), N(f) (logarithmic scale)');
legend('Signal (0.75 bits/sample)','Noise (0.75 bits/sample)','Signal (2 ...
    bits/sample)','Noise (2 bits/sample)','Signal (5 bits/sample)','Noise (5 ...
    bits/sample)');
hold off;

% Signal-to-Noise ratios
SNR1 = 10*log10(sum(Yf1)/sum(Nf1));
SNR2 = 10*log10(sum(Yf2)/sum(Nf2));
SNR3 = 10*log10(sum(Yf3)/sum(Nf3));

%------------ Exercise 2b ------------%

% Frequency sampling of the matched receive filter at low resolution
Fh1 = sqrt(Hfsq1(flf));
Fh2 = sqrt(Hfsq2(flf));
Fh3 = sqrt(Hfsq3(flf));
hn1 = FrSamp(Fh1);
hn2 = FrSamp(Fh2);
hn3 = FrSamp(Fh3);

% Frequency sampling of the matched transmit filter at low resolution
Fg1 = sqrt(Gfsq1(flf));
Fg2 = sqrt(Gfsq2(flf));
Fg3 = sqrt(Gfsq3(flf));
gn1 = FrSamp(Fg1);
gn2 = FrSamp(Fg2);
gn3 = FrSamp(Fg3);

figure(4);
hold on;
stem(0:1:NL-1,hn3,'r');
stem(0:1:NL-1,hn2,'g');
stem(0:1:NL-1,hn1,'b');
title(['Impulse response of matched receive filters (sampled from ' int2str(NL) ...
    ' points)']);
xlabel('n');
ylabel('h(n)');
legend('5 bits/sample','2 bits/sample','0.75 bits/sample');
axis([0 NL-1 0 0.6]);
hold off;

figure(5);
hold on;
stem(0:1:NL-1,gn3,'r');
stem(0:1:NL-1,gn2,'g');
stem(0:1:NL-1,gn1,'b');
title(['Impulse response of matched transmit filters (sampled from ' ...
    int2str(NL) ' points)']);
xlabel('n');
ylabel('g(n)');
legend('5 bits/sample','2 bits/sample','0.75 bits/sample');
axis([0 NL-1 -0.5 2.1]);
hold off;

% Frequency response of calculated receive filters
```

```
Hf1 = fft(hn1);
Hf2 = fft(hn2);
Hf3 = fft(hn3);
% High resolution frequency response of exact receive filters
Hfe1 = sqrt(Hfsq1(fhf(fhi)));
Hfe2 = sqrt(Hfsq2(fhf(fhi)));
Hfe3 = sqrt(Hfsq3(fhf(fhi)));

figure(6);
plot(flf(fli),abs(Hf1(fli)),fhf(fhi),abs(Hfe1),flf(fli),abs(Hf2(fli)), ...
    fhf(fhi),abs(Hfe2),flf(fli),abs(Hf3(fli)),fhf(fhi),abs(Hfe3));
title('Frequency response of exact and calculated receive filters');
xlabel('Normalized frequency (f)');
ylabel('H(f)');
legend('Calculated (0.75 bits/sample)','Exact (0.75 bits/sample)','Calculated ...
    (2 bits/sample)','Exact (2 bits/sample)','Calculated (5 ...
    bits/sample)','Exact (5 bits/sample)');

% Frequency response of calculated transmit filters
Gf1 = fft(gn1);
Gf2 = fft(gn2);
Gf3 = fft(gn3);
% High resolution frequency response of exact receive filters
Gfe1 = sqrt(Gfsq1(fhf(fhi)));
Gfe2 = sqrt(Gfsq2(fhf(fhi)));
Gfe3 = sqrt(Gfsq3(fhf(fhi)));

figure(7);
plot(flf(fli),abs(Gf1(fli)),fhf(fhi),abs(Gfe1),flf(fli),abs(Gf2(fli)), ...
    fhf(fhi),abs(Gfe2),flf(fli),abs(Gf3(fli)),fhf(fhi),abs(Gfe3));
title('Frequency response of exact and calculated transmit filters');
xlabel('Normalized frequency (f)');
ylabel('G(f)');
legend('Calculated (0.75 bits/sample)','Exact (0.75 bits/sample)', 'Calculated ...
    (2 bits/sample)','Exact (2 bits/sample)','Calculated (5 ...
    bits/sample)','Exact (5 bits/sample)');

% Signal modification
Sigmod1 = @(f) 1-sqrt(Lagrange1.*(sigmaq1./Sx(f)));
Sigmod2 = @(f) 1-sqrt(Lagrange2.*(sigmaq2./Sx(f)));
Sigmod3 = @(f) 1-sqrt(Lagrange3.*(sigmaq3./Sx(f)));

figure(8);
plot(fhf(fhi),abs(Sigmod1(fhf(fhi))),fhf(fhi),abs(Sigmod2(fhf(fhi))), ...
    fhf(fhi),abs(Sigmod3(fhf(fhi))));
title('Signal modification through the system');
xlabel('Normalized frequency (f)');
ylabel('|H(f)G(f)|');
legend('0.75 bits/sample','2 bits/sample','5 bits/sample');

%------------ Exercise 2c ------------%

% Generating the input signal
n = 100000;
xn = Oppgave1c(n,rho);
un1 = conv(xn,gn1);
un2 = conv(xn,gn2);
un3 = conv(xn,gn3);

% Quantize the signal
quantlimit = 6;
quantstep1 = sqrt(12*sigmaq1);
```

```
extreme1 = floor(quantlimit/quantstep1)*quantstep1;
quantlevels1 = (-(extreme1+quantstep1):quantstep1/2:(extreme1+quantstep1));
[¬,quants1] = quantiz(un1,quantlevels1(2:2:length(quantlevels1)), ...
    quantlevels1(1:2:length(quantlevels1)));

quantstep2 = sqrt(12*sigmaq2);
extreme2 = floor(quantlimit/quantstep2)*quantstep2;
quantlevels2 = (-(extreme2+quantstep2):quantstep2/2:(extreme2+quantstep2));
[¬,quants2] = quantiz(un2,quantlevels2(2:2:length(quantlevels2)), ...
    quantlevels2(1:2:length(quantlevels2)));

quantstep3 = sqrt(12*sigmaq3);
extreme3 = floor(quantlimit/quantstep3)*quantstep3;
quantlevels3 = -(extreme3+quantstep3):quantstep3/2:(extreme3+quantstep3);
[index,quants3] = quantiz(un3,quantlevels3(2:2:length(quantlevels3)), ...
    quantlevels3(1:2:length(quantlevels3)));

figure(9);
subplot(1,3,1);
hist(quants1,quantlevels1(1:2:length(quantlevels1)));
subplot(1,3,2);
hist(quants2,quantlevels2(1:2:length(quantlevels2)));
subplot(1,3,3);
hist(quants3,quantlevels3(1:2:length(quantlevels3)));
annotation('textbox', [0 0.9 1 0.1],'String', 'Quantized output from transmit ...
    filters with 0.75, 2 and 5 bits/sample','EdgeColor', ...
    'none','HorizontalAlignment', 'center');

% Calculate entropy
quant1 = hist(quants1,quantlevels1(1:2:length(quantlevels1)));
quant2 = hist(quants2,quantlevels2(1:2:length(quantlevels2)));
quant3 = hist(quants3,quantlevels3(1:2:length(quantlevels3)));

entr1 = 0;
entr2 = 0;
entr3 = 0;
for i = 1:length(quant1)
    Pi = quant1(i)/sum(quant1);
    if (Pi>0)
        entr1 = entr1 -(Pi*log2(Pi));
    end
end
for i = 1:length(quant2)
    Pi = quant2(i)/sum(quant2);
    if (Pi>0)
        entr2 = entr2 -(Pi*log2(Pi));
    end
end
for i = 1:length(quant3)
    Pi = quant3(i)/sum(quant3);
    if (Pi>0)
        entr3 = entr3 -(Pi*log2(Pi));
    end
end

% Output signals
yn1 = conv(quants1,hn1);
yn2 = conv(quants2,hn2);
yn3 = conv(quants3,hn3);

% Simulated Signal-Noise Ratios
delay = floor(NL/2);
```

```
SNR_sim1 = ...
    10*log10(sum(xn.^2)/sum((xn-yn1((2*delay)+1:1:length(yn1)-(2*delay))).^2));
SNR_sim2 = ...
    10*log10(sum(xn.^2)/sum((xn-yn2((2*delay)+1:1:length(yn2)-(2*delay))).^2));
SNR_sim3 = ...
    10*log10(sum(xn.^2)/sum((xn-yn3((2*delay)+1:1:length(yn3)-(2*delay))).^2));
```