Digital Communications - HW2 - MATLAB Code

Jacopo Pegoraro, Edoardo Vanin

24/04/2018

```
clc; close all; clear global; clearvars;

%% COMPUTE r(k)
Lvect=[7 15 31 63 127 255];
Nbound=20;
%additive noise
sigdB=-8;
sigmaw=10^(sigdB/10);

load('Noise_try.mat','w')
sw_collectLS=zeros(Nbound,1);

for n=1:length(Lvect)

    L=Lvect(n);

    for Ncurrent=1:Nbound

        %PN sequence
        x=[PN(L); PN(L)];
        %map all zeros to -1
        x=2*(x-0.5);

        a1=-0.9635;
        a2=0.4642;
        h=impz(1, [1 a1 a2]);
        %h=h(1:Ncurrent);
        h_even=h(1:2:end);
        h_odd=h(2:2:end);
%         if (Ncurrent<L)
%             h_even=h_even(1:ceil(Ncurrent/2));
%             h_even=h_even(1:floor(Ncurrent/2));
%         end

        wcut=w(1:2*length(x));
        %to save the previous results using the same noise
        w_even=wcut(1:2:end);
```

```
w_odd=wcut(2:2:end);
r_even=filter(h_even,1,x)+w_even;
r_odd=filter(h_odd,1,x)+w_odd;
%total output with a P/S converter
d_true=zeros(length(x),1);
for i=1:length(r_even)
    d_true(2*i-1)=r_even(i);
    d_true(2*i)=r_odd(i);
end

%% ESTIMATE OF h WITH THE CORRELATION METHOD
%we approximate the filter h with an FIR filter so the taps of h_0, h_1
%become the coefficients b_i of the frequency response

[h0_corr, h1_corr, r0_corr, r1_corr] =
corrEst(x, r_even,r_odd, Ncurrent);

%total estimated impulse response
h_corr=zeros(Ncurrent,1);
for i=1:length(h0_corr)
    h_corr(2*i-1)=h0_corr(i);
end
for i=1:length(h1_corr)
    h_corr(2*i)=h1_corr(i);
end
xlabel('nT_y'), ylim([-0.5 1.2]), xlim([-2 20])
% legend('h_{est-CORR}','h_{analytic}')


d_hatCORR=zeros(length(d_true),1);
%P/S converter
for i=1:2*L
    d_hatCORR(2*i-1)=r0_corr(i);
    d_hatCORR(2*i)=r1_corr(i);
end
%estimate of sigmaw
delta_dCORR=d_true-d_hatCORR;
Epsilon_minCORR=sum(delta_dCORR(L:2*L-1).^2);
sw_hatCORR_dB=10*log10(Epsilon_minCORR/L);
%sw_hatCORR_dB=10*log10((var(h0_corr)+var(h1_corr))*L/2);
sw_collectCORR(Ncurrent)=sw_hatCORR_dB;

%% LS
%the receiver knows only x(k) and the output d_true

[h0_ls, h1_ls, r0_ls, r1_ls] =
LSest(x, r_even,r_odd, Ncurrent);

%total estimated impulse response
h_ls=zeros(Ncurrent,1);
```

```
        for i=1:length(h0_ls)
            h_ls(2*i-1)=h0_ls(i);
        end
        for i=1:length(h1_ls)
            h_ls(2*i)=h1_ls(i);
        end

        %estimate of sigmaw
        d_hatLS=zeros(length(d_true),1);
        %P/S converter
        for i=1:2*L
            d_hatLS(2*i-1)=r0_ls(i);
            d_hatLS(2*i)=r1_ls(i);
        end
        delta_dLS=d_true-d_hatLS;
        Epsilon_minLS=sum(delta_dLS(L:2*L-1).^2);
        sw_hatLS_dB=10*log10(Epsilon_minLS/L);
%          Emin=(1/2)*(Emin0+Emin1);
%          sw_hatLS_dB=10*log10(Emin/L);
        sw_collectLS(Ncurrent)=sw_hatLS_dB;

    end

    SWcorr(:,n)=sw_collectCORR;
    SWls(:,n)=sw_collectLS;
end
SWcorr=SWcorr(2:end,:);
SWls=SWls(2:end,:);
save('swCORR.mat','SWcorr')
save('swLS.mat','SWls')

clc; close all; clear global; clearvars;

%% COMPUTE r(k)
L=31;
Ncurrent=9;

%additive noise
sigdB=-8;
sigmaw=10^(sigdB/10);

load('Noise_try.mat','w')


%PN sequence
x=[PN(L); PN(L)];
%map all zeros to -1
for i=1:length(x)
    if x(i)==0
        x(i)=-1;
```

```
    end
end

%filter polyphase components
a1=-0.9635;
a2=0.4642;
h=impz(1, [1 a1 a2]);

h_even=h(1:2:end);
h_odd=h(2:2:end);
%to seave the previous results using the same noise
wcut=w(1:2*length(x));
w_even=wcut(1:2:end);
w_odd=wcut(2:2:end);
r_even=filter(h_even,1,x)+w_even;
r_odd=filter(h_odd,1,x)+w_odd;
%total output with a P/S converter
d_true=zeros(length(x),1);
for i=1:length(r_even)
    d_true(2*i-1)=r_even(i);
    d_true(2*i)=r_odd(i);
end

%% ESTIMATE OF h WITH THE CORRELATION METHOD
%we approximate the filter h with an FIR filter so the taps of h_0, h_1
%become the coefficients b_i of the frequency response

[h0_corr, h1_corr, r0_corr, r1_corr] = corrEst(x, r_even, r_odd, Ncurrent);

%total estimated impulse response
h_corr=zeros(Ncurrent,1);
for i=1:length(h0_corr)
    h_corr(2*i-1)=h0_corr(i);
end
for i=1:length(h1_corr)
    h_corr(2*i)=h1_corr(i);
end
figure, stem(0:Ncurrent-1,h_corr), hold on,
stem(0:length(h)-1,h,'r*'), ...
    title(['Correlation method N=' int2str(Ncurrent),', ...
           L=' int2str(L)]), xlabel('nT_y'), ...
           ylim([-0.5 1.2]), xlim([-2 20])
legend('h_{est-CORR}','h_{analytic}')

d_hatCORR=zeros(length(x),1);
%P/S converter
for i=1:2*L
    d_hatCORR(2*i-1)=r0_corr(i);
    d_hatCORR(2*i)=r1_corr(i);
end
```

```
%estimate of sigmaw
delta_dCORR=d_true-d_hatCORR;
Epsilon_minCORR=sum(delta_dCORR(L:2*L-1).^2);
sw_hatCORR_dB=10*log10(Epsilon_minCORR/L);
%sw_hatCORR_dB=10*log10((var(h0_corr)+var(h1_corr))/(2*L));


%% LS
%the receiver knows only x(k) and the output d_true

[h0_ls, h1_ls, r0_ls, r1_ls] = LSest(x, r_even, r_odd, Ncurrent);

%total estimated impulse response
h_ls=zeros(Ncurrent,1);
for i=1:length(h0_ls)
    h_ls(2*i-1)=h0_ls(i);
end
for i=1:length(h1_ls)
    h_ls(2*i)=h1_ls(i);
end
figure, stem(0:Ncurrent-1, h_ls), hold on,
stem(0:length(h)-1,h,'r*'), title(['LS method N=' int2str(Ncurrent),',
        L=' int2str(L)]), xlabel('nT_y'), ylim([-0.5 1.2]), xlim([-2 20])
legend('h_{est-LS}','h_{analytic}')

%estimate of sigmaw
d_hatLS=zeros(length(d_true),1);
%P/S converter
for i=1:2*L
    d_hatLS(2*i-1)=r0_ls(i);
    d_hatLS(2*i)=r1_ls(i);
end
delta_dLS=d_true-d_hatLS;
Epsilon_minLS=sum(delta_dLS(L:2*L-1).^2);
sw_hatLS_dB=10*log10(Epsilon_minLS/L);

% Emin=(1/2)*(Emin0+Emin1);
% sw_hatLS_dB=10*log10(Emin/L);

function [h0, h1, r_0, r_1] = corrEst(x, d0, d1, Ncurrent)
%computes the estimates h_0 and h_1 and the outputs ot the 2 filters with
%the correlation method
L=length(x)/2;

h0=zeros(L, 1);
h1=zeros(L, 1);
for m=1:L-1
    rtemp0=zeros(L,1);
    rtemp1=zeros(L,1);
    for k=1:L
```

```
        %starts using the samples of d after a transient of ength L-1
        rtemp0(k)=d0(L-2+k)*conj(x(L-1+k-m));
        rtemp1(k)=d1(L-2+k)*conj(x(L-1+k-m));
    end
    h0(m)=sum(rtemp0)/L;
    h1(m)=sum(rtemp1)/L;
end

if (Ncurrent<L)
h0=h0(1:ceil(Ncurrent/2));
h1=h1(1:floor(Ncurrent/2));
end
%computes the outputs of the two polyphase estimated components
r_0=filter(h0,1,x);
r_1=filter(h1,1,x);

end

function [h0, h1, r_0, r_1] = LSest(x, d0, d1, Ncurrent)
%computes the estimate h_0 and h_1 and the outputs of the 2 filter with the
%LS method
L=length(x)/2;
%split even and odd samples of the output of the channel
% Ed0=sum(d0(L:2*L-1).^2);
% Ed1=sum(d1(L:2*L-1).^2);

I=zeros(L);
for k=1:L
    I(:,k)=x(L-k+1:(2*L-k));
end
o0=d0(L:2*L-1);
o1=d1(L:2*L-1);
Phi=I'*I;
theta0=I'*o0;
theta1=I'*o1;
h0=Phi\theta0;
h1=Phi\theta1;

if (Ncurrent<L)
h0=h0(1:ceil(Ncurrent/2));
h1=h1(1:floor(Ncurrent/2));
theta0=theta0(1:ceil(Ncurrent/2));
theta1=theta1(1:floor(Ncurrent/2));
end

% Emin0=Ed0-theta0'*h0;
% Emin1=Ed1-theta1'*h1;
%computes the outputs of the two polyphase estimated components
r_0=filter(h0,1,x);
r_1=filter(h1,1,x);
```

```
end

clc; close all; clear global; clearvars;
%generates white noise with variance -8dB, 1 million samples
sigdB=-8;
sighalf=0.5*10^(-sigdB/10);
sigdBhalf=10*log10(sighalf);
sigmaw=10^(sigdB/10);
w = wgn(10000,1,sigdB,'real');
save('Noise_try','w')



%% MAKES THE PLOT REQUIRED TO CHOOSE L AND N
% load('swCORR.mat','SWcorr')
% load('swLS.mat','SWls')
sigdB=-8;
N=[2:20];
%N=[1:5];
figure,

plot(N,sigdB*ones(19,1),'b--','LineWidth',2), hold on,
plot(N,SWcorr(:,1),'r--o'), hold on,
plot(N,SWcorr(:,2),'g--x'), hold on,
plot(N,SWcorr(:,3),'k--d'), hold on,
plot(N,SWcorr(:,4),'y--v'), hold on,
plot(N,SWcorr(:,5),'c--*'), hold on,
plot(N,SWcorr(:,6),'m--s'), hold on,
plot(N,SWls(:,1),'-ro'), hold on,
plot(N,SWls(:,2),'-gx'), hold on,
plot(N,SWls(:,3),'-kd'), hold on,
plot(N,SWls(:,4),'-yv'), hold on,
plot(N,SWls(:,5),'-c*'), hold on,
plot(N,SWls(:,6),'-ms')
legend('\sigma_w^2','L=7','L=15', ...
        'L=31','L=63','L=127', ...
        'L=255','L=7','L=15', ...
        'L=31','L=63','L=127',...
        'L=255','Location','South', ...
        'Orientation','horizontal')
ylabel('\epsilon/L')
xlabel('N')
title('\epsilon/L vs N')

% figure,
%
% plot(N,sigdB*ones(5,1),'b--','LineWidth',2), hold on,
% plot(N,SWls(1,:),'r--'), hold on,
% plot(N,SWls(4,:),'g--'), hold on,
```

```
% plot(N,SWls(10,:),'k--'), hold on,
% plot(N,SWls(15,:),'y--'), hold on,
% plot(N,SWls(19,:),'c--')

function [pn] = PN(L)

r = log2(L+1);
pn = zeros(L,1);

pn(1:r) = ones(1,r).';

for l=r+1:L
    switch r
        case 1
            pn(l) = pn(l-1);
        case 2
            pn(l) = xor(pn(l-1), pn(l-2));
        case 3
            pn(l) = xor(pn(l-2), pn(l-3));
        case 4
            pn(l) = xor(pn(l-3), pn(l-4));
        case 5
            pn(l) = xor(pn(l-3), pn(l-5));
        case 6
            pn(l) = xor(pn(l-5), pn(l-6));
        case 7
            pn(l) = xor(pn(l-6), pn(l-7));
        case 8
            pn(l) = xor(xor(pn(l-2), pn(l-3)), xor(pn(l-4), pn(l-8)));
    end
end
end

clc; close all; clear global; clearvars;

%% SETUP OF THE GIVEN PARAMETERS
Tc=1;
fd=(40*10^-5)/Tc;
Tp=1/10*(1/fd);
Nsamples=80000;
hplot_samples=7500;
KdB=2;
K=10^(KdB/10);
C=sqrt(K/(K+1));

%setup of the doppler filter coefficients (page 317)
a_ds=[1, -4.4153, 8.6283, -9.4592, 6.1051, ...
    -1.3542, -3.3622, 7.2390, -7.9361, 5.1221,...
    -1.8401, 2.8706e-1];
b_ds=[1.3651e-4, 8.1905e-4, 2.0476e-3, 2.7302e-3, ...
```

```
2.0476e-3, 9.0939e-4, 6.7852e-4, 1.3550e-3, 1.8076e-3, ...
1.3550e-3, 5.3726e-4, 6.1818e-5, -7.1294e-5, ...
    -9.5058e-5, -7.1294e-5, -2.5505e-5, 1.3321e-5, ...
    4.5186e-5, 6.0248e-5, 4.5186e-5, 1.8074e-5, 3.0124e-6];
% The energy of the doppler filter needs to be normalized to 1
h_ds=impz(b_ds, a_ds);
E_hds=sum(h_ds.^2);
b_ds=b_ds/sqrt(E_hds);
Npoints=Nsamples/Tp;
[Hds, f]=freqz(b_ds,a_ds,Npoints,'whole');
Hds2=(1/Npoints)*abs(Hds).^2;
% Hds=fft(h_ds, Npoints);
% Hds2=abs(Hds).^2;
D_f=fftshift(Hds2);
figure, plot((f/(2*pi))-0.5, D_f)
ylabel('|H_{ds}|^2')
xlim([-0.5 0.5])
xlabel('Normalized frequency (over Tp)')
%normalize the power of h0.
Md=1-C^2;

%% SIMULATION OF THE CHANNEL IMPULSE RESPONSE
%compute the transient as 5*Neq*Tp because the final sampling time is Tq
pvec=abs(roots(a_ds));
pmax=max(pvec);
Neq=ceil(-1/log(pmax));
transient=5*Tp*Neq;
%transient=length(h_ds)*Tp;

h_nsamples=Nsamples+transient;
w_samples=ceil(h_nsamples/Tp);
% Complex-valued Gaussian white noise with zero mean and unit variance
w=wgn(w_samples,1,0,'complex');

h_full=filter(b_ds, a_ds, w);
figure, stem(impz(b_ds,a_ds),'.'), ylabel('|h_{ds}|'), ...
xlabel('nT_p'), xlim([0 150])

%interpolation to Tq
t = 1:length(h_full);
t_int = Tc/Tp:Tc/Tp:length(h_full);
h_int = interp1(t, h_full, t_int, 'spline');

%multiply by sqrt(M_h0) to give the desired power
h_int=h_int*sqrt(Md);

%drop the transient and add C because of LOS component
h=h_int(transient+1:end)+C;
```

```
%plot of 7500 samples of h
figure,
plot(abs(h(1:hplot_samples)))
xlabel('nT_c')
ylabel('|h_0(nT_c)|')
xlim([1 hplot_samples])
title('Impulse response of the channel')

%% ESTIMATE OF THE PDF OF H0bar

h_bar=h/sqrt(C^2+Md); % the normalization here does not make much sense
% as M_h0=1-C^2, but it's to keep the formulas as in the book

% mean and variance of the realization of h
h_sd=std(abs(h_bar));
h_mean=mean(abs(h_bar));
% magnitude
mag_h=abs(h_bar);
% fit the data with a Rice ditribution to derive the parameters
[v, s]=ricefit(mag_h);
% parameters v ans s are related to the rice factor K (wikipedia rician
% distribution)
K_est=v^2/(2*s^2); % estimated value of K

% compute the theoretical and the estimated distributions
x=linspace(0,3,1000);
est=ricepdf(x,v,s);
v_th=sqrt(K/(K+1));
s_th=sqrt(1/(2*(K+1)));
th=ricepdf(x,v_th,s_th );

% plot of the theoretical and estimated distributions
figure
%plot(x,est,'r'), hold on, plot(x,th,'b')
Nbins=17;
histogram(mag_h, Nbins,'Normalization','pdf','DisplayStyle','stairs'), ...
hold on, plot(x,th,'r-.'),% hold on, plot(x,est,'k'),
plot(x,est,'r-.'), hold on, plot(x,th,'k--'),
legend('pdf with K_{est}','theoretical pdf')
title('Estimate of the pdf of h_0')
ylabel('f_x(a)')
xlabel('a')
%legend('Histogram','theoretical pdf','Estimated pdf')

%% SPECTRUM ESTIMATION

% Welch estimator
D=ceil(Nsamples/4);    %window length
S=ceil(D/2);    %overlap
```

```
w_welch=window(@bartlett,D);
%w_welch=kaiser(D,5);

Welch_P = welchPSD(h', w_welch, S);
Welch_P=Welch_P/Nsamples;
%f=1/Tc:1/Tc:Nsamples;
Welch_centered=fftshift(Welch_P);

figure,
freq1=[-Nsamples/2+1:Nsamples/2];
freq2=[-Npoints/2+1:Npoints/2];
% freq2=[-1/Tp:1/Tp:1/Tp];
peak=10*log10(C^2);
analytic=10*log10(Md*D_f);
analytic(length(analytic)/2)=peak;
plot(freq1, 10*log10(Welch_centered)) , hold on, plot(freq2, analytic,'r')

ylim([-40 0])
%ylim([-55 -15])
xlim([-5*Nsamples*fd 5*Nsamples*fd])
xticks([-160 -128 -96 -64 -32 0 32 64 96 128 160])
xticklabels({'-5f_d','-4f_d','-3f_d','-2f_d','-f_d',...
    '0','f_d','2f_d','3f_d','4f_d','5f_d'});
ylabel('PSD [dB]')
xlabel('f')
legend('Estimate','Theoretical curve')

function [welch_est] = welchPSD(inputsig, window, overlaps)
% REQUIRES COLUMN VECTOR FOR THE INPUT DATA

% Length of the window
D = length(window);
% Length of input signal
K = length(inputsig);
% Normalized energy of the window
Mw = sum(window .^ 2) * (1/D);
% Number of subsequences
N_s = floor((K-D)/(D-overlaps) + 1);
%Initialization of each periodogram
P_per = zeros(K, N_s);
%inputsig=inputsig-mean(inputsig);
for s = 0:(N_s-1)
    % Windowed input
    x_s = window .* inputsig(s*(D-overlaps)+1:s*(D-overlaps)+D);
    % DFT on K samples of windowed input
    X_s = fft(x_s, K);
    % Periodogram for the window
    P_per(:,s+1) = (abs(X_s)).^2 * (1/(D*Mw)); % Tc = 1;
end
% Sum of all periodograms
```

```
welch_est = sum(P_per, 2) * (1/N_s);
end
```