Digital Communications - HW3 - MATLAB Code

Jacopo Pegoraro, Edoardo Vanin

24/04/2018

```
%% AWGN BOUND SIMULATION

Pe_AWGNsim = zeros(length(SNR_dB), 1);
for i=1:length(SNR_dB)

    a_dist(:,i) = a + w(1:length(a), i);

    a_det = zeros(length(a), length(SNR_dB));
    for k=1:length(a)
        a_det(k,i) = threshold_detector(a_dist(k,i));
    end

    [Pe_AWGNsim(i), ~] = SER(a, a_det(:,i));
end

save('Pe_AWGNsim.mat','Pe_AWGNsim')

clc; clear all; close all;

%% Configuration parameters
if ~exist("Noise.mat", 'file')
    noise_seq;
end
load('Noise','w','sigma_w');
verbose = false;
plot_figure = true;

r = 20;
SNR_dB = [8 9 10 11 12 13 14];
SNR_lin = 10.^(SNR_dB./10);
sigma_a = 2;

T = 1;
q_c_num   = [0 0 0 0 0 0.7424];
q_c_denom = [1 -0.67];
q_c = impz(q_c_num, q_c_denom);
```

```matlab
% cut the impulse response when too small
q_c = [0; 0; 0; 0; 0; q_c( q_c >= max(q_c)*10^(-2) )];
E_qc = sum(q_c.^2);

N0 = sigma_w./4;

%% Generation of the input signal

pn = [PN(r)];

pn(pn == 0) = -1;

a = zeros(floor(length(pn)/2),1);
for i=1:2:(length(pn) - 1)
   a((i+1)/2)=  pn(i) + 1i * pn(i+1);
end

clear pn;

%% Filtering through the channel

a_prime = upsample(a, 4);

s_c = filter(q_c_num, q_c_denom, a_prime);

%% Add noise

r_c = zeros(length(s_c), length(SNR_dB));

for i = 1:length(SNR_dB)
     r_c(:,i) = s_c + w(1:length(s_c),i);
end

%% Save the workspace

save("common.mat");

function [decisions] = equalization_DFE(x, c, b, D)
%EQUALIZATION for DFE
M2 = length(b);
y = conv(x,c);
y = y(1:length(x)+D);
detected = zeros(length(x) + D, 1);
for k=0:length(y)-1
      if (k <= M2)
         a_past = [flipud(detected(1:k)); zeros(M2 - k, 1)];
     else
         a_past = flipud(detected(k - M2 + 1: k));
     end
detected(k + 1) = threshold_detector(y(k + 1) + b.' * a_past);
```

```
end
%scatterplot(y)
decisions = detected(D + 1:end);
end

function [decisions] = equalization_LE(x, c, D, norm_factor)
%EQUALIZATION+detection for LE

y = conv(x,c);
y = y(1:length(x)+D);
y_tilde = y./norm_factor;
detected = zeros(length(x) + D, 1);
for k=0:length(y)-1
detected(k + 1) = threshold_detector(y_tilde(k + 1));
end
%scatterplot(y_tilde);
decisions = detected(D + 1:end);
end

function [decisions] = equalization_pointC(x, c, b, D)
%EQUALIZATION for DFE
M2 = length(b);
y = conv(x,c);
y = downsample(y,2);
y = y(1:floor(length(x)/2));
detected = zeros(ceil(length(x)/2) + D, 1);
for k=0:length(y)-1
     if (k <= M2)
         a_past = [flipud(detected(1:k)); zeros(M2 - k, 1)];
    else
         a_past = flipud(detected(k-M2+1:k));
    end
detected(k + 1) = threshold_detector(y(k + 1) + b.'*a_past);
end
%scatterplot(y)
decisions = detected(D + 1:end);
end

function [detected] = FBA(y, psiD, L1, L2)
% y: the data after filter c (hopefully no effect of the precursors)
% psiD: overall system impulse response after the cancellation
% of precursors by filter c
% L1: number of considered precursors for each symbol
% L2: number of considered precursors for each symbol

M = 4;              % cardinality of the constellation
Ns = M^(L1+L2);     % # of states
K = length(y);
%QPSK symbols
symb = [1+1i, 1-1i, -1+1i, -1-1i];
```

```
tStart = tic;

% initialize matrix with the input data U
states_symbols = zeros(Ns, M);
statelength = L1 + L2;
statevec = zeros(1, statelength);
U = zeros(Ns, M);
for state = 1:Ns
    for j = 1:M
        lastsymbols = [symb(statevec + 1), symb(j)];
        U(state, j) = lastsymbols * flipud(psiD);
    end
    states_symbols(state,:) = lastsymbols(1:M);
    % update statevec
    statevec(statelength) = statevec(statelength) + 1;
    i = statelength;
    while (statevec(i) >= M && i > 1)
        statevec(i) = 0;
        i = i-1;
        statevec(i) = statevec(i) + 1;
    end
end

%computation of matrix C (3D)
c = zeros(M, Ns, K+1);

for k = 1:K
    c(:, :, k) = (-abs(y(k) - U).^2).';
end
c(:,:,K+1) = 0;


% backward metric
b = zeros(Ns, K+1);    %matrix

% the index has to go backwards
for k = K:-1:1
    for i = 1:Ns
        % the index of the state is computed with the following
        % expression
        possible_state = mod(i-1, M^(L1 + L2 - 1))*M + 1;
        % the value of b is computed from b(k+1)
        b(i, k) = max(b(possible_state:possible_state+M-1, k+1) ...
            + c(:, i, k+1));
    end
end
```

```
% forward metric, state metric, log-likelihood function
% f_old is set to -1
f_old = zeros(Ns, 1);
f_new = zeros(Ns, 1);
%will contain the symbols from which we select the one with the highest
%likelihood
likely = zeros(M, 1);
detected = zeros(K, 1);
row_step = (0:M-1)*M^(L1+L2-1);
for k = 1:K
    for j = 1:Ns
        in_vec = ceil(j/M) + row_step;
        f_new(j) = max(f_old(in_vec) + c(mod(j-1, 4)+1, in_vec, k).');
    end
    v = f_new + b(:, k);
    for beta = 1:M
        ind = find(states_symbols(:,M) == symb(beta));
        likely(beta) = max(v(ind));
    end
    [~, maxind] = max(likely);
    detected(k) = symb(maxind);
    f_old = f_new;
end

toc(tStart)

end


load('P_e_LE.mat','Pe_LE')
load('P_e_DFE.mat','Pe_DFE')
load('Pe_AWGNsim.mat','Pe_AWGNsim')
%load('Pe_c.mat','Pe_c')
%load('Pe_d.mat','Pe_d')
load('viterbi.mat','Pe_viterbi')
load('fba.mat','Pe_FBA')
Pe_c = [0.07011 0.04315 0.024578 0.01244 0.005542 0.002073 0.0007324];
Pe_d=[0.0798 0.0569 0.0385 0.0245 0.0149 0.0079 0.0040];

SNR=[8:14];
SNR_lin = 10.^(SNR./10);
sigma_a = 2;
awgn_bound = 2*qfunc(sqrt(SNR_lin));


figure,
semilogy(SNR, Pe_LE,'b--')
grid on;
hold on,
semilogy(SNR, Pe_DFE,'b')
```

```
hold on,
semilogy(SNR, Pe_c,'k--')
hold on,
semilogy(SNR, Pe_d,'k')
hold on,
semilogy(SNR, Pe_viterbi, 'r--')
hold on,
semilogy(SNR, Pe_FBA, 'r')
hold on,
semilogy(SNR, Pe_AWGNsim, 'g--')
hold on,
semilogy(SNR, awgn_bound,'g')
ylim([10^-4 10^-1])
xlim([8 14])
xlabel('SNR [dB]')
ylabel('P_e')
legend('MF+LE@T','MF+DFE@T','AAF+MF+DFE@T/2','AAF+DFE@T/2','VA','FBA', ...
    'MF b-S','MF b-T');

clc; close all; clear global; clearvars;
%generates white noise with variance -10dB, 3 million samples

q_c_num   = [0 0 0 0 0 0.7424];
q_c_denom = [1 -0.67];
q_c = impz(q_c_num, q_c_denom);

length_w = 3 * 10^6;
% cut the impulse response
q_c = [0; 0; 0; 0; 0; q_c( q_c >= max(q_c)*10^(-2) )];
E_qc = sum(q_c.^2);

SNR_dB = [8:14];
sigma_a = 2;
SNR_lin = 10.^(SNR_dB./10);
w = zeros(length_w, 7);
sigma_w = zeros(length(SNR_dB),1);
for i=1:length(SNR_dB)
    sigma_w(i) = (sigma_a * E_qc) / SNR_lin(i);
    w(:,i) = sqrt(sigma_w(i))/sqrt(2).*(randn(length_w, 1) + ...
        1i*randn(length_w, 1));
    %w(:,i) = wgn(length_w, 1, 10*log10(sigma_w(i)), 'complex');
end

save('Noise','w','sigma_w')

function [pn] = PN(r)

L = pow2(r) - 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)));
        case 9
            pn(l) = xor(pn(l-5), pn(l-9));
        case 10
            pn(l) = xor(pn(l-7), pn(l-10));
        case 11
            pn(l) = xor(pn(l-9), pn(l-11));
        case 12
            pn(l) = xor(xor(pn(l-2), pn(l-10)), xor(pn(l-11), pn(l-12)));
        case 13
            pn(l) = xor(xor(pn(l-1), pn(l-11)), xor(pn(l-12), pn(l-13)));
        case 14
            pn(l) = xor(xor(pn(l-2), pn(l-12)), xor(pn(l-13), pn(l-14)));
        case 15
            pn(l) = xor(pn(l-14), pn(l-15));
        case 16
            pn(l) = xor(xor(pn(l-11), pn(l-13)), xor(pn(l-14), pn(l-16)));
        case 17
            pn(l) = xor(pn(l-14), pn(l-17));
        case 18
            pn(l) = xor(pn(l-11), pn(l-18));
        case 19
            pn(l) = xor(xor(pn(l-14), pn(l-17)), xor(pn(l-19), pn(l-18)));
        case 20
            pn(l) = xor(pn(l-17), pn(l-20));
    end
end
end

clc; clear all; close all;
format long g
```

```
%% Load common variable
if ~exist("common.mat", 'file')
    common;
end

load("common.mat");
Pe_LE = zeros(length(SNR_dB),1);
errors = zeros(length(SNR_dB),1);
r_r = zeros(length(s_c), length(SNR_dB));

%% Receiver filter

% Match filter
g_m = conj(flipud(q_c));

% Compute the h impulse response
h = conv(q_c, g_m);
h = downsample(h,4);
%h = h(h ~= 0);

for i=1:length(SNR_dB)
    r_r(:,i) = filter(g_m, 1, r_c(:,i));
end
% For debugging purpose
s_r = filter(g_m, 1, s_c);

%% Sampling
%t_0 equal to the peak of h
t_0_bar = length(g_m);
x_no_noise = downsample(s_r(t_0_bar:end), 4);

x = zeros(length(x_no_noise), length(SNR_dB));
for i=1:length(SNR_dB)
    x(:,i) = downsample(r_r(t_0_bar:end, i), 4);
end

%scatterplot(x)
%% Filtering thorugh C and equalization

r_gm = xcorr(g_m);
% r_w = N0 .* downsample(r_gm, 4);
for i=1:length(SNR_dB)
    r_w_up(:,i)= N0(i) * r_gm;
    r_w(:,i) = downsample(r_w_up(:,i), 4);
end

D = 2;
M1 = 5;
%M1 = 5; D = 4;
```

```
c =zeros(M1, length(SNR_dB));
for i=1:length(SNR_dB)

    c(:,i) = WienerC_LE(h, r_w(:,i), sigma_a, M1, D);

    psi(:,i) = conv(c(:,i), h);
    %psi(:,i) = psi(:,i)/max(psi(:,i));

    decisions = equalization_LE(x(:,i), c(:,i), D, max(psi(:,i)));

    [Pe_LE(i), errors(i)] = SER(a(1:length(decisions)), decisions);
end

%save('P_e_LE.mat','Pe_LE')

%% plots
if plot_figure == true

    [Q_c, f] = freqz(q_c_num, q_c_denom, 'whole');
%     figure, plot(real(a(1:50))), title('a'), ylim([-1.5 1.5])
%     figure, plot(real(a_prime(1:50))), title('a_pr'),ylim([-1.5 1.5])
%     figure, plot(real(s_c(1:50))), title('s_c'),ylim([-1.5 1.5])
%     figure, plot(real(s_r(1:50))), title('s_r'), ylim([-1.5 1.5])
%     figure, plot(real(r_c(1:50,3))), title('r_c'),ylim([-3 3])
%     figure, plot(real(x(1:50,3))), title('x'),ylim([-3 3])

    figure, stem(h)
    title('h_i'), xlabel('nT')

    figure, stem([0:length(q_c)-1], q_c), xlabel('nT/4'), title('q_c')
    figure, stem(g_m), xlabel('nT/4'), title('g_M')

    figure
    plot(f/(0.5*pi), 10*log10(abs(Q_c))), xlim([0 2])
    title('Frequency response Q_c')
    xlabel('Normalized frequency, T=1')
    ylabel('Q_c [dB]')

    figure, stem([-4:8], abs(psi(:,3))), xlabel('nT'), ...
        title('|\Psi|, D=2, M1=5')
    figure, stem([0:length(c(:,3))-1], abs(c(:,3))), xlabel('nT'), ...
        title('|c|')
end

clc;
clear all;
close all;
format long g
%% Load common variable
if ~exist("common.mat", 'file')
```

```
    common;
end

load("common.mat");
Pe_DFE = zeros(length(SNR_dB),1);
errors = zeros(length(SNR_dB),1);
r_r = zeros(length(s_c), length(SNR_dB));

%% Receiver filter

% Costruzione del filtro g_M
% Per l'esercizio a Ã¨ un "semplice" matched filter
g_m = conj(flipud(q_c));

% Calculate the h impulse response
h = conv(q_c, g_m);
h = downsample(h,4);
h = h(h ~= 0);

N1 = floor(length(h)/2);
N2 = N1;

for i=1:length(SNR_dB)
    r_r(:,i) = filter(g_m, 1, r_c(:,i));
end

% For debuggig pourpose
s_r = filter(g_m, 1, s_c);

%% Sampling

t_0_bar = length(g_m);
x_no_noise = downsample(s_r(t_0_bar:end), 4);
x = zeros(length(x_no_noise), length(SNR_dB));
for i=1:length(SNR_dB)
    x(:,i) = downsample(r_r(t_0_bar:end, i), 4);
end

%% Filtering through C and equalization

r_gm = xcorr(g_m);
% r_w = N0 .* downsample(r_gm, 4);
for i=1:length(SNR_dB)
    r_w_up(:,i)= N0(i) * r_gm;
    r_w(:,i) = downsample(r_w_up(:,i), 4);
end

% M1_span = [2:20];
% D_span = [2:20];
```

```
%
% % M1_span = 4;
% % D_span = 2;
%
% Jvec = zeros(19);
% for k=1:length(M1_span)
%      for l=1:length(D_span)
%           M1 = M1_span(k);
%           D = D_span(l);
%           M2 = N2 + M1 - 1 - D;
%           [c, Jmin] = WienerC_DFE(h, r_w, sigma_a, M1, M2, D);
%           Jvec(k,l) = Jmin;
%      end
% end
%
% for i=1:length(D_span)
%     figure,
%     plot(2:20, Jvec(:,i))
% end

% psi = conv(c, h);
% psi = psi/max(psi);
%
% b = - psi(end - M2 + 1:end);
%
% for i=1:length(SNR_dB)
%      decisions = equalization_DFE(x(:,i), c, b, M1, M2, D);
%
%      [Pe(i), errors(i)] = SER(a(1:length(decisions)), decisions);
% end

% M1 = 5;
% D = 0;
M1 = 5; D = 4;
M2 = N2 + M1 - 1 - D;

c =zeros(M1, length(SNR_dB));
b = zeros(M2,1);
for i=1:length(SNR_dB)

    c(:,i) = WienerC_DFE(h, r_w(:,i), sigma_a, M1, M2, D);

    psi(:,i) = conv(c(:,i), h);
    %psi(:,i) = psi(:,i)/max(psi(:,i));
    b(:,i) = -psi(find(psi==max(psi))+1:end,i);
    decisions = equalization_DFE(x(:,i), c(:,i), b(:,i), D);

    [Pe_DFE(i), errors(i)] = SER(a(1:length(decisions)), decisions);
end
%save('P_e_DFE.mat','Pe_DFE')
```

```
%% plots
if plot_figure == true

    [Q_c, f] = freqz(q_c_num, q_c_denom, 'whole');

    %figure, stem(h)
    %title('h_i'), xlabel('nT')

    %figure, stem(q_c), xlabel('nT/4'), title('q_c')
    figure, stem([0:length(g_m)-1],g_m), xlabel('nT/4'), title('g_M')

    figure
    plot(f/(2*pi), 10*log10(abs(Q_c))), xlim([0 0.5])
    title('Frequency response Q_c')

    figure, stem([-2:6],abs(psi(:,3))), xlabel('nT'), ...
        title('|\Psi|, D=4, M1=5')
    figure, stem([0:length(c(:,3))-1], abs(c(:,3))), xlabel('nT'), ...
        title('|c|'), xlim([0 6])
    figure, stem([0:length(b(:,3))-1], abs(b(:,3))), xlabel('nT'), ...
        title('|b|'), xlim([-1 6])
end

 clear all; close all;

%% Load common variable
if ~exist("common.mat", 'file')
    common;
end

load("common.mat");

select = 3;

%% AA filter

Fpass = 0.2;              % Passband Frequency
Fstop = 0.3;              % Stopband Frequency
Dpass = 0.057501127785;   % Passband Ripple
Dstop = 0.01;             % Stopband Attenuation
dens  = 20;               % Density Factor

% Calculate the order from the parameters using FIRPMORD.
[N, Fo, Ao, W] = firpmord([Fpass, Fstop], [1 0], [Dpass, Dstop]);

% Calculate the coefficients using the FIRPM function.
g_AA  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(g_AA);
```

```
r_r = filter(g_AA , 1, r_c(:,select));
s_r = filter(g_AA, 1, s_c);

figure, stem(s_r(1:100)), title('s_r'), xlabel('nT/4')
figure, stem(r_r(1:100)), title('r_r'), xlabel('nT/4')

qg_up = conv(q_c, g_AA);
qg_up = qg_up.';
%freqz(qg_up, 1,'whole');
figure, stem(qg_up), title('convolution of g_AA and q_c'), xlabel('nT/4')

%% Timing phase and decimation

t0_bar = find(qg_up == max(qg_up));
x_prime = downsample(r_r(t0_bar:end), 2);
x_NN_prime = downsample(s_r(t0_bar:end), 2);

figure, stem(x_NN_prime(1:100)), title('xprime without noise'), xlabel('nT/2')
figure, stem(x_prime(1:100)), title('xprime'), xlabel('nT/2')

qg = downsample(qg_up(1:end), 2);
K = 1/sum(qg.^2);
g_m = K*conj(flipud(qg));

figure, stem(g_m), title('g_m'), xlabel('nT/2')

x = filter(g_m, 1, x_prime);
x = x(13:end);
figure, stem(x(1:100)), title('x'), xlabel('nT/2')

x_NN = filter(g_m, 1, x_NN_prime);

h = conv(qg, g_m);
h = h(h ~= 0);

%scatterplot(x_NN)
figure, stem(h), title('h'), xlabel('nT/2')
figure, stem(x_NN(1:100)), title('x without noise'), xlabel('nT/2')

%% Equalization and symbol detection

r_g = xcorr(conv(g_AA, flipud(qg_up)));
N0 = (sigma_a * E_qc) / (4 * SNR_lin(select));
r_w = N0 * downsample(r_g, 2);

figure, stem(r_w), title('r_w'), xlabel('nT/2')
figure, stem(r_g), title('r_g'), xlabel('nT/2')

N1 = floor(length(h)/2);
N2 = N1;
```

```
M1 = 5;
D = 4;
M2 = N2 + M1 - 1 - D;

[c, Jmin] = WienerC_frac(h, r_w, sigma_a, M1, M2, D, N1, N2);
psi = conv(h,c);

figure, stem(c), title('c'), xlabel('nT/2')
figure, stem(psi), title('psi'), xlabel('nT/2')

psi_down = downsample(psi(2:end),2); % The b filter acts at T
b = -psi_down(find(psi_down == max(psi_down)) + 1:end);

figure, stem(b), title('b'), xlabel('nT')
decisions = equalization_pointC(x, c, b, D);

%detection
[Pe_c, errors] = SER(a(4:length(decisions)), decisions)

clear all; close all;

%% Load common variable
if ~exist("common.mat", 'file')
    common;
end

load("common.mat");

select = 3;

%% AA filter

Fpass = 0.2;              % Passband Frequency
Fstop = 0.3;              % Stopband Frequency
Dpass = 0.057501127785;   % Passband Ripple
Dstop = 0.01;             % Stopband Attenuation
dens  = 20;               % Density Factor

% Calculate the order from the parameters using FIRPMORD.
[N, Fo, Ao, W] = firpmord([Fpass, Fstop], [1 0], [Dpass, Dstop]);

% Calculate the coefficients using the FIRPM function.
g_AA  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(g_AA);

select=3;

r_r = filter(g_AA , 1, r_c(:,select));
figure, stem(r_r(1:100)), title('r_r'), xlabel('nT/4')
```

```
s_r=filter(g_AA, 1, s_c);
figure, stem(s_r(1:100)), title('s_r'), xlabel('nT/4')
qg_up = conv(q_c, g_AA);
qg_up = qg_up.';
figure, stem(qg_up), title('convolution of g_AA and q_c'), xlabel('nT/4')
t0_bar = find(qg_up==max(qg_up));
x = downsample(r_r(t0_bar:end), 2);
figure, stem(x(1:100)), title('xprime'), xlabel('nT/2')
x_NN=downsample(s_r(t0_bar:end), 2);
figure, stem(x_NN(1:100)), title('xprime without noise'), xlabel('nT/2')

%scatterplot(x_NN)
h = downsample(qg_up,2);
figure, stem(h), title('h'), xlabel('nT/2')

r_g = xcorr(g_AA);
figure, stem(r_g), title('r_g'), xlabel('nT/2')
N0 = (sigma_a * E_qc)/(4*SNR_lin(select));
r_w = N0 * downsample(r_g, 2);
figure, stem(r_w), title('r_w'), xlabel('nT/2')

N1 = 11;
N2 = 12;

M1 = 5;
D = 2;
M2 = N2 + M1 - 1 - D;

[c, Jmin]= WienerC_frac(h, r_w, sigma_a, M1, M2, D, N1, N2);
figure, stem(c), title('c'), xlabel('nT/2')
psi = conv(h,c);
figure, stem(psi), title('psi'), xlabel('nT/2')
psi_down = downsample(psi(2:end),2);
b = -psi_down(find(psi_down==max(psi_down))+1:end);
figure, stem(b), title('b'), xlabel('nT')
decisions = equalization_pointC(x, c, b, D);

%detection
[Pe_c, errors] = SER(a(1:length(decisions)), decisions);


%% plots

figure, stem(g_AA), title('g_AA'), xlabel('nT/4')
% figure, stem(r_c(1:100,3)), title('r_c'), xlabel('nT/4')
% figure, stem(r_r(1:100,3)), title('r_r'), xlabel('nT/4')
% figure, stem(s_r(1:100)), title('s_r'), xlabel('nT/4')
% figure, stem(qg_up), title('convolution of g_AA and q_c'), xlabel('nT/4')
% figure, stem(x(1:100,3)), title('x'), xlabel('nT/2')
% figure, stem(x_NN(1:100)), title('x without noise'), xlabel('nT/2')
```

```
figure, stem(h), title('h'), xlabel('nT/2')
% figure, stem(r_gAA), title('r_g'), xlabel('nT/2')
% figure, stem(r_w(:,3)), title('r_g'), xlabel('nT/2')
figure, stem(c), title('c'), xlabel('nT/2')
figure, stem(psi), title('psi'), xlabel('nT/2')

clc;
clear all;
close all;
format long g
%% Load common variable
if ~exist("common.mat", 'file')
    common;
end

load("common.mat");

Pe_viterbi = zeros(length(SNR_dB),1);
errors = zeros(length(SNR_dB),1);
r_r = zeros(length(s_c), length(SNR_dB));

%% Receiver filter

% match filter
g_m = conj(flipud(q_c));

% Compute the impulse response h
h = conv(q_c, g_m);
h = downsample(h,4);
h = h(h ~= 0);

N1 = floor(length(h)/2);
N2 = N1;

for i=1:length(SNR_dB)
    r_r(:,i) = filter(g_m, 1, r_c(:,i));
end

% For debuggig pourpose
s_r = filter(g_m, 1, s_c);

%% Sampling

t_0_bar = length(g_m);
x_no_noise = downsample(s_r(t_0_bar:end), 4);
x = zeros(length(x_no_noise), length(SNR_dB));
for i=1:length(SNR_dB)
    x(:,i) = downsample(r_r(t_0_bar:end, i), 4);
end
```

```
%% Filtering through C and equalization

r_gm = xcorr(g_m);
% r_w = N0 .* downsample(r_gm, 4);
for i=1:length(SNR_dB)
    r_w_up(:,i)= N0(i) * r_gm;
    r_w(:,i) = downsample(r_w_up(:,i), 4);
end

M1 = 5;
D = 2;
M2 = N2 + M1 - 1 - D;


c =zeros(M1, length(SNR_dB));
b = zeros(M2,1);

for i=1:length(SNR_dB)

    [c(:,i) Jmin(i)]= WienerC_DFE(h, r_w(:,i), sigma_a, M1, M2, D);

    psi(:,i) = conv(c(:,i), h);
    y = conv(x(:,i),c(:,i));
    y = y./max(psi(:,i));
    a_conf  =  a(1+4-0 : end-M2+M2-2);
    decisions = VBA(y, psi(:,i), 0, M2-2, 4, M2);
    decisions = decisions(D+1:end);
    [Pe_viterbi(i),errors(i)] = SER(a_conf(1:length(decisions)),decisions);
end

%save('viterbi.mat', 'Pe_viterbi');

clc;
clear all;
%close all;
format long g
%% Load common variable
if ~exist("common.mat", 'file')
    common;
end

load("common.mat");

Pe_FBA = zeros(length(SNR_dB),1);
errors = zeros(length(SNR_dB),1);
r_r = zeros(length(s_c), length(SNR_dB));

%% Receiver filter

% match filter
```

```
g_m = conj(flipud(q_c));

% Computes the impulse response h
h = conv(q_c, g_m);
h = downsample(h,4);
h = h(h ~= 0);

N1 = floor(length(h)/2);
N2 = N1;

for i=1:length(SNR_dB)
    r_r(:,i) = filter(g_m, 1, r_c(:,i));
end

% For debuggig pourpose
s_r = filter(g_m, 1, s_c);

%% Sampling

t_0_bar = length(g_m);
x_no_noise = downsample(s_r(t_0_bar:end), 4);
x = zeros(length(x_no_noise), length(SNR_dB));
for i=1:length(SNR_dB)
    x(:,i) = downsample(r_r(t_0_bar:end, i), 4);
end

%% Filtering through C and equalization

r_gm = xcorr(g_m);
% r_w = N0 .* downsample(r_gm, 4);
for i=1:length(SNR_dB)
    r_w_up(:,i)= N0(i) * r_gm;
    r_w(:,i) = downsample(r_w_up(:,i), 4);
end

M1 = 5; D = 4;
M2 = N2 + M1 - 1 - D;

c =zeros(M1, length(SNR_dB));
b = zeros(M2,1);

for i=1:length(SNR_dB)

    [c(:,i) Jmin(i)]= WienerC_DFE(h, r_w(:,i), sigma_a, M1, M2, D);
    psi(:,i) = conv(c(:,i), h);
    %psi(:,i) = psi(:,i)/max(psi(:,i));
    b(:,i) = -psi(find(psi==max(psi))+1:end,i);
    y = conv(x(:,i),c(:,i));
    y = y ./ max(psi(:,i));
    %var_w(i) = 10^(Jmin(i)/10) - (abs(1-max(psi(:,i)))^2)*sigma_a;
```

```
    indexD = find(psi(:,i) == max(psi(:,i)));
    L1 = 0; L2 = 4;
    psi_pad = [psi(:,i); 0; 0];
    indexD = find(psi_pad == max(psi_pad));
    decisions = FBA(y, psi_pad(indexD:end), L1, L2);
    [Pe_FBA(i), errors(i)] = SER(a(1:end-4), decisions(5:end));
end

%save('fba.mat', 'Pe_FBA');

function [Pe, count_errors] = SER(sent, detected)
% Computes the symbol-error rate, it accepts QPSK symbols
count_errors = 0;
for i=1:length(sent)
    if sent(i) ~= detected(i)
        count_errors = count_errors + 1;
    end
end
% count_errors = sum((sent-detected)~=0);
Pe = count_errors/length(sent);
end

function [a_hat_kD] = threshold_detector(y_k)

if (real(y_k) > 0)
    if (imag(y_k) > 0)
        a_hat_kD = 1+1i;
    else
        a_hat_kD= 1-1i;
    end
else
    if (imag(y_k) > 0)
        a_hat_kD = -1+1i;
    else
        a_hat_kD = -1-1i;
    end
end

end

function [detected] = VBA(r_c, hi, L1, L2, N1, N2)

M = 4;
symb = [1+1i, 1-1i, -1+1i, -1-1i]; % All possible transmitted symbols (QPSK)
Kd = 28;
Ns = M ^ (L1+L2); % Number of states
r_c  =  r_c(1+N1-L1 : end-N2+L2);   % Discard initial and final samples of r
hi = hi(1+N1-L1 : end-N2+L2);   % Discard initial and final samples of hi

tStart = tic;
```

```
survSeq = zeros(Ns, Kd);
detectedSymb = zeros(1, length(r_c));
cost = zeros(Ns, 1);

statelength = L1 + L2;
statevec = zeros(1, statelength);
%matrix with the input values
U = zeros(Ns, M);
for state = 1:Ns
    for j = 1:M
        lastsymbols = [symb(statevec + 1), symb(j)];
        U(state, j) = lastsymbols * flipud(hi);
    end
    statevec(statelength) = statevec(statelength) + 1;
    i = statelength;
    while (statevec(i) >= M && i > 1)
        statevec(i) = 0;
        i = i-1;
        statevec(i) = statevec(i) + 1;
    end
end

for k = 1 : length(r_c)

    nextcost = - ones(Ns, 1);
    pred = zeros(Ns, 1);
    nextstate = 0;

    for state = 1 : Ns

        for j = 1 : M
            nextstate = nextstate + 1;
            if nextstate > Ns, nextstate = 1; end
            u = U(state, j);
            newstate_cost = cost(state) + abs(r_c(k) - u)^2;
            if nextcost(nextstate) == -1 ...
                    || nextcost(nextstate) > newstate_cost
                nextcost(nextstate) = newstate_cost;
                pred(nextstate) = state;
            end
        end
    end

    temp = zeros(size(survSeq));
    for nextstate = 1:Ns
        temp(nextstate, 1:Kd) = ...
            [survSeq(pred(nextstate), 2:Kd), ...
            symb(mod(nextstate-1, M)+1)];
    end
```

```
    [~, decided_index] = min(nextcost);
    detectedSymb(1+k) = survSeq(decided_index, 1);
    survSeq = temp;

    % Update the cost to be used as cost at time k-1 in the next iteration
    cost = nextcost;
end

toc(tStart)

detectedSymb(length(r_c)+2 : length(r_c)+Kd) = survSeq(decided_index, 1:Kd-1);

detectedSymb = detectedSymb(Kd+1 : end);
detected = detectedSymb;
detected = detected(2:end);
end

 function [c_opt, Jmin] = WienerC_DFE(h, r_w, sigma_a, M1, M2, D)

    N1 = floor(length(h)/2);
    N2 = N1;
    padding = 60;
    hpad = padarray(h, padding);

    % Padding the noise correlation
    r_w_pad = padarray(r_w, padding);

    p  = zeros(M1 ,1);

    for i = 0 : M1-1
        p(i + 1) = sigma_a * conj(hpad(N1 + padding + 1 + D - i));
    end

    R = zeros(M1);
    for row = 0:(M1-1)
        for col = 0:(M1-1)

            fsum = (hpad((padding + 1):(N1 + N2 + padding + 1))).' ...
                * conj(hpad((padding + 1 - (row - col)):( N1 + N2 + ...
                padding + 1 - (row - col))));

            if M2==0
                ssum=0;
            else
                ssum = (hpad((N1+padding+1+1+D-col): ...
                    (N1+padding+1+M2+D-col))).' * ...
                    conj((hpad((N1+padding+1+1+D-row):...
                    (N1+padding+1+M2+D-row))));
            end
```

```
            R(row + 1, col + 1) = sigma_a * (fsum - ssum) +...
                r_w_pad(padding + 1 + row - col + ...
                (floor(length(r_w) / 2 )));
        end
    end

    c_opt = R \ p;

    temp2 = zeros(M1, 1);

    for l = 0:M1-1
        temp2(l + 1) = c_opt(l + 1) * hpad(N1 + padding + 1 + D - l);
    end

    Jmin = 10*log10(sigma_a * (1 - sum(temp2)));
end

 function [c_opt, Jmin] = WienerC_frac(h, r_w, sigma_a, M1, M2, D, N1, N2)

    padding = 100;
    hpad = padarray(h, padding);

    % Padding the noise correlation
    r_w_pad = padarray(r_w, padding);

    p  = zeros(M1 ,1);

    for i = 0 : M1-1
        p(i + 1) = sigma_a * conj(hpad(N1 + padding + 1 + 2*D - i));
    end

    R = zeros(M1);
    for row = 0:(M1-1)
        for col = 0:(M1-1)

            f=zeros(length(h),1);
            for n=0:length(h)-1
                f(n+1) = hpad(padding + 1 + 2 * n - col)*conj(...
                    hpad(padding + 1 + 2 * n - row));
            end
            fsum = sum(f);

            s=zeros(M2,1);
            for j=1:M2
                s(j) = hpad(N1 + padding + 1 + 2*(j+D) -col)*conj(...
                    hpad(N1 + padding + 1+2*(j+D) -row ));
            end
            ssum = sum(s);
```

```
            R(row + 1, col + 1) = sigma_a * (fsum - ssum) + r_w_pad(...
                padding + 1 + row - col + (floor(length(r_w) / 2 )));
        end
    end
    %to avoid ill conditioning
    R = R + 0.1*eye(M1);
    c_opt = R \ p;

    temp2 = zeros(M1, 1);
    for l = 0:M1-1
        temp2(l + 1) = c_opt(l + 1) * hpad(N1 + padding + 1 +2*D-l);
    end

    Jmin = 10*log10(abs(sigma_a * (1 - sum(temp2))));
end

function [c_opt, Jmin] = WienerC_LE(h, r_w, sigma_a, M1, D)
%calls Wiener for DFE passing M2=0
[c_opt, Jmin] = WienerC_DFE(h, r_w, sigma_a, M1, 0, D);
end
```