# Digital Communications and Laboratory
# Third Homework

Faccin Dario, Santi Giovanni

## MATLAB code

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

% Load input and noise
load('Useful.mat');

% Channel SNR
snr_db = 10;
snr_lin = 10^(snr_db/10);

sigma_a = 2;     % Input variance

[r_c, sigma_w, ~] = channel_sim(in_bits, snr_db, sigma_a);
r_c = r_c + w(:,3);

gm = conj(qc(end:-1:1));          % Matched filter: complex conjugate of qc

figure()
stem(abs(gm));
xlabel('$m\frac{T}{4}$');
ylabel('$g_m$')
xlim([1 length(gm)]);
grid on

h = conv(qc,gm);                          % Impulse response
h = h(h>max(h)/100);
h = h(3:end-2);

h_T = downsample(h,4);            % Downsampling impulse response
r_c_prime = filter(gm,1,r_c);     % Filtering received signal
t0_bar = find(h == max(h));               % Determining timing phase

r_c_prime = r_c_prime(t0_bar:end);        % Remove "transient"
x = downsample(r_c_prime,4);              % Downsample received signal

r_gm = xcorr(gm,gm);                      % Filter autocorrelation
N0 = (sigma_a * E_qc) / (4 * snr_lin);
rw_tilde = N0 .* downsample(r_g, 2);
% Parameters for Linear Equalizer
M1 = 7;
M2 = 0;
D = 6;
[c_opt, Jmin] = Adaptive_DFE(h_T, rw_tilde, sigma_a, M1, M2, D);
```

```
psi = conv(c_opt, h_T); % Overall impulse response

figure
subplot(121), stem(0:length(c_opt)-1,abs(c_opt)), hold on, grid on
ylabel('$|c|$'), xlabel('n'); xlim([0 7]);
subplot(122), stem(-2:length(psi)-3,abs(psi)), grid on
ylabel('$|\psi|$'), xlabel('n'); xlim([-2 8]);

detected = equalization_LE(x, c_opt, M1, D, max(psi));

nerr = length(find(in_bits(1:length(detected))~=detected));
Pe = nerr/length(detected);
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

% Load input and noise
load('Useful.mat');

% Channel SNR
snr_db = 10;
snr_lin = 10^(snr_db/10);

sigma_a = 2;     % Input variance

[r_c, sigma_w, ~] = channel_sim(in_bits, snr_db, sigma_a);
r_c = r_c + w(:,3);

gm = conj(qc(end:-1:1));          % Matched filter: complex conjugate of qc

figure()
stem(abs(gm));
xlabel('$m\frac{T}{4}$');
ylabel('$g_m$')
xlim([1 length(gm)]);
grid on

h = conv(qc,gm);                                % Impulse response
h_T = downsample(h,4);                 % Downsampling impulse response
r_c_prime = filter(gm,1,r_c);    % Filtering received signal
t0_bar = find(h == max(h));            % Determining timing phase

r_c_prime = r_c_prime(t0_bar:end);     % Remove "transient"
x = downsample(r_c_prime,4);           % Downsample received signal

r_gm = xcorr(gm,gm);                   % Filter autocorrelation
N0 = (sigma_a * E_qc) / (4 * snr_lin);
rw_tilde = N0 .* downsample(r_g, 2);

% Parameters for DFE
N2 = floor(length(h_T)/2);
M1 = 5;
D = 4;
M2 = N2 + M1 - 1 - D;
[c_opt, Jmin] = Adaptive_DFE(h_T, rw_tilde, sigma_a, M1, M2, D);
```

```
psi = conv(c_opt, h_T);          % Overall impulse response
psi = psi/max(psi);                          % Normalization

b = - psi(end - M2 + 1:end);     % Feedback coefficients

figure
subplot(131), stem(0:length(c_opt)-1,abs(c_opt)), hold on, grid on
title('$|c|$'), xlabel('n');
subplot(132), stem(0:length(psi)-1,abs(psi)), grid on
title('$|\psi|$'), xlabel('n');
subplot(133), stem(0:length(b)-1,abs(b)), grid on
title('|b|'), xlabel('n');

detected = equalization_DFE(x, c_opt, b, M1, M2, D);

nerr = length(find(in_bits(1:length(detected))~=detected));
Pe = nerr/length(detected);
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

% Load input, noise and filter
load('Useful.mat');
load('GAA_filter.mat');

% Channel SNR
snr_db = 10;
snr_lin = 10^(snr_db/10);

sigma_a = 2;     % Input variance

[r_c, sigma_w, ~] = channel_sim(in_bits, snr_db, sigma_a);
r_c = r_c + w(:,3);

r_c_prime = filter(g_AA , 1, r_c);       % Filtering using antialiasing

qg_up = conv(qc, g_AA);
qg_up = qg_up.';

t0_bar = find(qg_up == max(qg_up));                          % Timing phase
x = downsample(r_c_prime(t0_bar:end), 2);
qg = downsample(qg_up(1:end), 2);
g_m = conj(flipud(qg));                                          % Matched
     filter

x_prime = filter(g_m, 1, x);
x_prime = x_prime(13:end);

h = conv(qg, g_m);

r_g = xcorr(conv(g_AA, g_m));            % AA and MF crosscorrelation
N0 = (sigma_a * E_qc) / (4 * snr_lin);
rw_tilde = N0 .* downsample(r_g, 2);

% Parameters for Equalizer
```

```
N1 = floor(length(h)/2);
N2 = N1;
M1 = 5;
D = 4;
M2 = N2 + M1 - 1 - D;

[c, Jmin] = WienerC_frac(h, rw_tilde, sigma_a, M1, M2, D, N1, N2);
psi = conv(h,c);                                    % Overall impulse response
psi_down = downsample(psi(2:end),2); % The b filter act at T
b = -psi_down(find(psi_down == max(psi_down)) + 1:end);
x_prime = x_prime/max(psi);                         % Normalization
detected = equalization_pointC(x_prime, c_opt, b, D);
detected = detected(1:end-D);
in_bits_2 = in_bits(1:length(detected));
errors = length(find(in_bits_2~=detected(1:length(in_bits_2))));
Pe = errors/length(in_bits_2);
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

% Load input, noise and filter
load('Useful.mat');
load('GAA_filter.mat');

% Channel SNR
snr_db = 10;
snr_lin = 10^(snr_db/10);

sigma_a = 2;     % Input variance

[r_c, sigma_w, ~] = channel_sim(in_bits, snr_db, sigma_a);
r_c = r_c + w(:,3);

r_c_prime = filter(g_AA , 1, r_c);       % Filtering using antialiasing

qg_up = conv(qc, g_AA);
qg_up = qg_up.';

t0_bar = find(qg_up == max(qg_up));              % Timing phase
x = downsample(r_c_prime(t0_bar:end), 2);

qg = downsample(qg_up, 2);
x_prime = x;
h = qg;

r_g = xcorr(g_AA);                                        % AA
      autocorrelation
N0 = (sigma_a * E_qc) / (4 * snr_lin);
rw_tilde = N0 * downsample(r_g, 2);

N1 = floor(length(h)/2);
N2 = 12;
M1 = 10;
D = 4;
M2 = N2 + M1 - 1 - D;
```

```
[c_opt, Jmin] = WienerC_frac(h, rw_tilde, sigma_a, M1, M2, D, N1, N2);
psi = conv(h,c_opt);                                     % Overall impulse response
psi = psi/max(psi);
psi_down = downsample(psi(2:end),2);     % The b filter act at T
b = -psi_down(find(psi_down == max(psi_down)) + 1:end);
x = x/max(psi);                                                  % Normalization
detected = equalization_pointC(x, c_opt, b, D);
detected = detected(2:end-D);
in_bits_2 = in_bits(1:length(detected));
errors = length(find(in_bits_2~=detected(1:length(in_bits_2))));
Pe = errors/length(in_bits_2);
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

% Load input and noise
load('Useful.mat');

% Channel SNR
snr_db = 10;
snr_lin = 10^(snr_db/10);

sigma_a = 2;     % Input variance

[r_c, sigma_w, ~] = channel_sim(in_bits, snr_db, sigma_a);
r_c = r_c + w(:,3);

gm = conj(qc(end:-1:1));          % Matched filter: complex conjugate of qc

h = conv(qc,gm);                                  % Impulse response
t0_bar = find(h == max(h));                % Determining timing phase
h_T = downsample(h,4);                     % Downsampling impulse response
r_c_prime = filter(gm,1,r_c);    % Filtering received signal

r_c_prime = r_c_prime(t0_bar:end);         % Remove "transient"
x = downsample(r_c_prime,4);               % Downsample received signal

r_gm = xcorr(gm,gm);                       % Filter autocorrelation
N0 = (sigma_a * E_qc) / (4 * snr_lin);
rw_tilde = N0 .* downsample(r_g, 2);

% Parameters for DFE
N2 = floor(length(h_T)/2);
M1 = 5;
D = 4;
M2 = N2 + M1 - 1 - D;
[c_opt, Jmin] = Adaptive_DFE(h_T, rw_tilde, sigma_a, M1, M2, D);

psi = conv(c_opt, h_T);           % Overall impulse response
psi = psi/max(psi);                        % Normalization

figure
subplot(121), stem(0:length(c_opt)-1,abs(c_opt)), hold on, grid on
title('$|c|$'), xlabel('n');
subplot(122), stem(0:length(psi)-1,abs(psi)), grid on
title('$|\psi|$'), xlabel('n');
```

```
y = conv(x, c_opt);                        % Input for Viterbi
y = y/max(psi);
detected = Viterbi(y, psi, 0, M2-2, 8, M2);
in_bits_2  =  in_bits(1+8-0 : end-(M2)+(M2-2));
detected = detected.';
detected = detected(D+1:end);

nerr = length(find(in_bits_2(1:length(detected))~=detected));
Pe = nerr/length(detected);
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

% Load input and noise
load('Useful.mat');

% Channel SNR
snr_db = 10;
snr_lin = 10^(snr_db/10);

sigma_a = 2;     % Input variance

[r_c, sigma_w, ~] = channel_sim(in_bits, snr_db, sigma_a);
r_c = r_c + w(:,3);

gm = conj(qc(end:-1:1));          % Matched filter: complex conjugate of qc

h = conv(qc,gm);                                    % Impulse response
t0_bar = find(h == max(h));                % Determining timing phase
h_T = downsample(h,4);                     % Downsampling impulse response
r_c_prime = filter(gm,1,r_c);    % Filtering received signal

r_c_prime = r_c_prime(t0_bar:end);         % Remove "transient"
x = downsample(r_c_prime,4);               % Downsample received signal

r_gm = xcorr(gm,gm);                       % Filter autocorrelation
N0 = (sigma_a * E_qc) / (4 * snr_lin);
rw_tilde = N0 .* downsample(r_gm, 2);

% Parameters for DFE
M1 = 5;
N2 = floor(length(h_T)/2);
D = 4;
M2 = N2 + M1 - 1 - D;
[c_opt, Jmin] = Adaptive_DFE(h_T, rw_tilde, sigma_a, M1, M2, D);

psi = conv(c_opt, h_T);
psi = psi/max(psi);

figure
subplot(121), stem(0:length(c_opt)-1,abs(c_opt)), hold on, grid on
title('$|c|$'), xlabel('n');
subplot(122), stem(0:length(psi)-1,abs(psi)), grid on
title('$|\psi|$'), xlabel('n');
```

```
y = conv(x, c_opt);                % Input for ForwardBackward
y = y/max(psi);

L1 = 0;
L2 = 4;
indexD = find(psi == max(psi));

detected = FBA(y, psi(indexD-L1:indexD+L2), L1, L2);

nerr = length(find(in_bits(1:length(detected))~=detected));
Pe = nerr/length(detected);
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

load('Useful.mat', 'in_bits', 'qc', 'E_qc');

SNR_vect = 8:14;
sigma_a = 2;
M = 4;
gm = conj(qc(end:-1:1));
h = conv(qc,gm);
t0_bar = find(h == max(h));
h = h(h>max(h)/100);
h = h(3:end-2);
h_T = downsample(h,4);
r_gm = xcorr(gm,gm);
realizations = 1:10;
Pe_LE_avg = zeros(length(SNR_vect),1);
Pe_LE = zeros(length(realizations),1);
M1 = 6;
M2 = 0;
D = 4;

for i=1:length(SNR_vect)
        Pe_LE = zeros(length(SNR_vect),1);
        for k=1:length(realizations)
                snr_db = SNR_vect(i);
                snr_lin = 10^(snr_db/10);
                [r_c, sigma_w, qc] = channel_sim(in_bits, snr_db, sigma_a);
                w = wgn(length(r_c),1, 10*log10(sigma_w), 'complex');
                r_c = r_c + w;
                r_c_prime = filter(gm,1,r_c);
                r_c_prime = r_c_prime(t0_bar:end);
                x = downsample(r_c_prime,4);
                N0 = (sigma_a * E_qc) / (4 * snr_lin);
                rw_tilde = N0 .* downsample(r_g, 2);
                [c_opt, Jmin] = Adaptive_DFE(h_T, rw_tilde, sigma_a, M1, M2, D);
                detected = equalization_LE(x, c_opt, M1, D, max(conv(c_opt, h_T)));
                nerr = length(find(in_bits(1:length(detected))~=detected));
                Pe_LE(k) = nerr/length(detected);
        end
        Pe_LE_avg(i) = sum(Pe_LE)/length(Pe_LE);
end

figure();
```

```
semilogy(SNR_vect, Pe_LE_avg, 'b--');
grid on;
ylim([10^-4 10^-1]); xlim([8 14]);

% save('PE_LE_avgs.mat', 'Pe_LE_avg');
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

load('Useful.mat', 'in_bits', 'qc', 'E_qc');

SNR_vect = 8:14;
sigma_a = 2;
M = 4;
gm = conj(qc(end:-1:1));
h = conv(qc,gm);
t0_bar = find(h == max(h));
h = h(h>max(h)/100);
h = h(3:end-2);
h_T = downsample(h,4);
r_gm = xcorr(gm,gm);
realizations = 1:10;
Pe_DFE_avg = zeros(length(SNR_vect),1);
Pe_DFE = zeros(length(realizations),1);
N2 = floor(length(h_T)/2);
N1 = N2;
M1 = 5;
D = 4;
M2 = N2 + M1 - 1 - D;

for i=1:length(SNR_vect)
        Pe_DFE = zeros(length(SNR_vect),1);
        for k=1:length(realizations)
                snr_db = SNR_vect(i);
                snr_lin = 10^(snr_db/10);
                [r_c, sigma_w, qc] = channel_sim(in_bits, snr_db, sigma_a);
                w = wgn(length(r_c),1, 10*log10(sigma_w), 'complex');
                r_c = r_c + w;
                r_c_prime = filter(gm,1,r_c);
                r_c_prime = r_c_prime(t0_bar:end);
                x = downsample(r_c_prime,4);
                N0 = (sigma_a * E_qc) / (4 * snr_lin);
                rw_tilde = N0 .* downsample(r_g, 2);
                [c_opt, Jmin] = Adaptive_DFE(h_T, rw_tilde, sigma_a, M1, M2, D);
                psi = conv(c_opt, h_T);
                psi = psi/max(psi);
                b = - psi(end - M2 + 1:end);
                detected = equalization_DFE(x, c_opt, b, M1, M2, D);
                [Pe_DFE(k),~] = SER(in_bits(1:length(detected)), detected);
        end
        Pe_DFE_avg(i) = sum(Pe_DFE)/length(Pe_DFE);
end

figure();
semilogy(SNR_vect, Pe_DFE_avg, 'b');
grid on;
```

```
ylim([10^-4 10^-1]); xlim([8 14]);

% save('PE_DFE_avgs.mat', 'Pe_DFE_avg');
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

load('Useful.mat', 'in_bits', 'qc', 'E_qc');
load('GAA_filter.mat');
SNR_vect = 8:14;
sigma_a = 2;
qg_up = conv(qc, g_AA);
qg_up = qg_up.';
t0_bar = find(qg_up == max(qg_up));
qg = downsample(qg_up(1:end), 2);
g_m = conj(flipud(qg));
h = conv(qg, g_m);
h = h(h ~= 0);
r_g = xcorr(conv(g_AA, g_m));
realizations = 1:10;
Pe_AA_GM_avg = zeros(length(SNR_vect),1);
Pe_AA_GM = zeros(length(realizations),1);
N1 = floor(length(h)/2);
N2 = N1;
M1 = 10;
D = 4;
M2 = N2 + M1 - 1 - D;

for i=1:length(SNR_vect)
        Pe_AA_GM = zeros(length(SNR_vect),1);
        for k=1:length(realizations)
                snr_db = SNR_vect(i);
                snr_lin = 10^(snr_db/10);
                [r_c, sigma_w, qc] = channel_sim(in_bits, snr_db, sigma_a);
                w = wgn(length(r_c),1, 10*log10(sigma_w), 'complex');
                r_c = r_c + w;
                r_c_prime = filter(g_AA , 1, r_c);
                x = downsample(r_c_prime(t0_bar:end), 2);
                x_prime = filter(g_m, 1, x);
                x_prime = x_prime(13:end);
                N0 = (sigma_a * E_qc) / (4 * snr_lin);
                rw_tilde = N0 .* downsample(r_g, 2);
                [c, Jmin] = WienerC_frac(h, rw_tilde, sigma_a, M1, M2, D, N1, N2);
                psi = conv(h,c);
                psi_down = downsample(psi(2:end),2); % The b filter act at T
                b = -psi_down(find(psi_down == max(psi_down)) + 1:end);
                x_prime = x_prime/max(psi);
                detected = equalization_pointC(x_prime, c, b, D);
                detected = detected(1:end-D);
                in_bits_2 = in_bits(1:length(detected));
                errors = length(find(in_bits_2~=detected(1:length(in_bits_2))));
                Pe_AA_GM(k) = errors/length(in_bits_2);
        end
        Pe_AA_GM_avg(i) = sum(Pe_AA_GM)/length(Pe_AA_GM);
end
```

```
figure();
semilogy(SNR_vect, Pe_AA_GM_avg, 'k--');
grid on;
ylim([10^-4 10^-1]); xlim([8 14]);

% save('PE_AA_GM_avgs.mat', 'Pe_AA_GM_avg');
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

load('Useful.mat', 'in_bits', 'qc', 'E_qc');
load('GAA_filter.mat');
SNR_vect = 8:14;
sigma_a = 2;
qg_up = conv(qc, g_AA);
qg_up = qg_up.';
t0_bar = find(qg_up == max(qg_up));
qg = downsample(qg_up(1:end), 2);
h = qg;
r_g = xcorr(g_AA);
realizations = 1:10;
Pe_AA_NOGM_avg = zeros(length(SNR_vect),1);
Pe_AA_NOGM = zeros(length(realizations),1);
N1 = floor(length(h)/2);
N2 = N1;
M1 = 5;
D = 4;
M2 = N2 + M1 - 1 - D;

for i=1:length(SNR_vect)
        Pe_AA_NOGM = zeros(length(SNR_vect),1);
        for k=1:length(realizations)
                snr_db = SNR_vect(i);
                snr_lin = 10^(snr_db/10);
                [r_c, sigma_w, qc] = channel_sim(in_bits, snr_db, sigma_a);
                w = wgn(length(r_c),1, 10*log10(sigma_w), 'complex');
                r_c = r_c + w;
                r_c_prime = filter(g_AA , 1, r_c);
                x = downsample(r_c_prime(t0_bar:end), 2);
                x_prime = x;
                N0 = (sigma_a * E_qc) / (4 * snr_lin);
                rw_tilde = N0 .* downsample(r_g, 2);
                [c, Jmin] = WienerC_frac(h, rw_tilde, sigma_a, M1, M2, D, N1, N2);
                psi = conv(h,c);
                psi_down = downsample(psi(2:end),2); % The b filter act at T
                b = -psi_down(find(psi_down == max(psi_down)) + 1:end);
                detected = equalization_pointC(x_prime, c, b, D);
                detected = detected(1:end-D);
                in_bits_2 = in_bits(3:length(detected));
                errors = length(find(in_bits_2~=detected(1:length(in_bits_2))));
                Pe_AA_NOGM(k) = errors/length(in_bits_2);
        end
        Pe_AA_NOGM_avg(i) = sum(Pe_AA_NOGM)/length(Pe_AA_NOGM);
end

figure();
```

```
semilogy(SNR_vect, Pe_AA_NOGM_avg, 'k');
grid on;
ylim([10^-4 10^-1]); xlim([8 14]);

% save('PE_AA_NOGM_avgs.mat', 'Pe_AA_NOGM_avg');
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

load('Useful.mat', 'in_bits', 'qc', 'E_qc');

SNR_vect = 8:14;
sigma_a = 2;
M = 4;
gm = conj(qc(end:-1:1));
h = conv(qc,gm);
t0_bar = find(h == max(h));
h_T = downsample(h,4);
r_gm = xcorr(gm,gm);
in_bits_2 = in_bits(1+4-0 : end-4+4-2);
realizations = 1:10;
Pe_VA_avg = zeros(length(SNR_vect),1);
Pe_VA = zeros(length(realizations),1);
M1 = 5;
N2 = floor(length(h_T)/2);
D = 2;
M2 = N2 + M1 - 1 - D;

for i=1:length(SNR_vect)
        Pe_VA = zeros(length(SNR_vect),1);
        for k=1:length(realizations)
                snr_db = SNR_vect(i);
                snr_lin = 10^(snr_db/10);
                [r_c, sigma_w, qc] = channel_sim(in_bits, snr_db, sigma_a);
                w = wgn(length(r_c),1, 10*log10(sigma_w), 'complex');
                r_c = r_c + w;
                r_c_prime = filter(gm,1,r_c);
                r_c_prime = r_c_prime(t0_bar:end);
                x = downsample(r_c_prime,4);
                rw_tilde = sigma_w/4 .* downsample(r_gm, 4);
                [c_opt, Jmin] = Adaptive_DFE(h_T, rw_tilde, sigma_a, M1, M2, D);
                psi = conv(c_opt, h_T);
                psi = psi/max(psi);
                y = conv(x, c_opt);
                y = y/max(psi);
                detected = VBA(y, psi, 0, 2, 4, 2);
                in_bits_2 = in_bits(1+4-0 : end-2+2);
                detected = detected.';
                detected = detected(D+1:end);
                [Pe_VA(k),~] = SER(in_bits_2(1:length(detected)), detected);
        end
        Pe_VA_avg(i) = sum(Pe_VA)/length(Pe_VA);
end

figure();
semilogy(SNR_vect, Pe_VA_avg, 'r--');
```

```
grid on;
ylim([10^-4 10^-1]); xlim([8 14]);

% save('PE_VA_avgs.mat', 'Pe_VA_avg');
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

load('Useful.mat', 'in_bits', 'qc', 'E_qc');

SNR_vect = 8:14;
sigma_a = 2;
M = 4;
realizations = 1:10;
Pe_AWGN_SIM_avg = zeros(length(SNR_vect),1);
Pe_AWGN_SIM = zeros(length(realizations),1);
awgn_bound = zeros(length(SNR_vect),1);

for i=1:length(SNR_vect)
        Pe_AWGN_SIM = zeros(length(SNR_vect),1);
        for k=1:length(realizations)
                snr_db = SNR_vect(i);
                snr_lin = 10^(snr_db/10);
                r_c = in_bits;
                w = wgn(length(r_c),1, 10*log10(sigma_a / snr_lin), 'complex');
                r_c = r_c + w;
                detected = zeros(length(r_c),1);
                for l=1:length(in_bits)
                detected(l) = QPSK_detector(r_c(l));
                end
                [Pe_AWGN_SIM(k),~] = SER(in_bits(1:length(detected)), detected);
        end
        Pe_AWGN_SIM_avg(i) = sum(Pe_AWGN_SIM)/length(Pe_AWGN_SIM);
        awgn_bound(i) = 4*(1-1/sqrt(M))*qfunc(sqrt(snr_lin/(sigma_a/2)));
end

figure();
semilogy(SNR_vect, Pe_AWGN_SIM_avg, 'g--');
hold on; grid on;
semilogy(SNR_vect, awgn_bound, 'g');
ylim([10^-4 10^-1]); xlim([8 14]);

% save('Pe_AWGN_SIM_avgs.mat', 'Pe_AWGN_SIM_avg', 'awgn_bound');
```

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

N1 = floor(length(h)/2);
N2 = N1;
padding = 60;
hpad = padarray(h, padding);
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)));    % Cost function
end
```

```
function [decisions] = equalization_LE(x, c, M1, D, norm_fact)
% EQUALIZATION for LE
y = zeros(length(x)+D,1);
detected = zeros(length(x)+D,1);
for k = 0:length(x)-1+D
        if (k < M1-1)
        xconv = [flipud(x(1:k+1)); zeros(M1-k-1,1)];
    elseif k > length(x)-1 && k < length(x)-1+M1
        xconv = [zeros(k-length(x)+1, 1); flipud(x(end-M1+1+k-...
            length(x)+1:end))];
    elseif k >= length(x)-1+M1 % just in case D is greater than M1
        xconv = zeros(M1,1);
    else
        xconv = flipud(x(k-M1+1+1:k+1));
        end
    y(k+1) = c.'*xconv/norm_fact;
    detected(k+1) = QPSK_detector(y(k+1));
end
decisions = detected(D+1:end);
end
```

```
function [decisions] = equalization_DFE(x, c, b, M1, M2, D)
% EQUALIZATION for DFE
y = zeros(length(x)+D,1);
detected = zeros(length(x)+D,1);
for k = 0:length(x)-1+D
        if (k < M1-1)
        xconv = [flipud(x(1:k+1)); zeros(M1-k-1,1)];
    elseif k > length(x)-1 && k < length(x)-1+M1
        xconv = [zeros(k-length(x)+1, 1); flipud(x(end-M1+1+k-...
```

```
            length(x)+1:end))];
    elseif k >= length(x)-1+M1 % just in case D is greater than M1
        xconv = zeros(M1,1);
    else
        xconv = flipud(x(k - M1 + 1 + 1:k + 1));
        end
        if (k <= M2)
        a_old = [flipud(detected(1:k)); zeros(M2-k,1)];
    else
        a_old = flipud(detected(k-M2+1:k));
        end
    y(k+1) = c.'*xconv;
    detected(k+1) = QPSK_detector(y(k+1) + b.'*a_old);
end
decisions = detected(D+1:end);
end
```

```
function [c_opt, Jmin] = WienerC_frac(h, r_w, sigma_a, M1, M2, D, N1, ~)
% OPTIMUM COEFFICIENTS FOR POINT C
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
R = R + 0.1*eye(M1); % Avoid ill conditioning
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 [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) = QPSK_detector(y(k+1)+b.'*a_past);
end
decisions = detected(D+1:end);
end
```

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

if (L1 > N1) || (L2 > N2)
    disp('Check your input')
    return
end

M = 4;
symb = [1+1i, 1-1i, -1+1i, -1-1i]; % QPSK constellation
Kd = 28; % Size of the Trellis diagram (and of the matrix)
Ns = M ^ (L1+L2); % 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

survSeq = zeros(Ns, Kd);
detectedSymb = zeros(1, length(r_c));
cost = zeros(Ns, 1); % Define Gamma(-1) for each state (cost)

statelength = L1 + L2; % state length
statevec = zeros(1, statelength); % symb idx: old --> new
u_mat = zeros(Ns, M);
for state = 1:Ns
    for j = 1:M
        lastsymbols = [symb(statevec + 1), symb(j)]; % symbols: old --> new
        u_mat(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)
    % Initialize the costs of the new states to -1
    costnew = - ones(Ns, 1);
    % Vector of the predecessors: the i-th element is the predecessor at
```

```
    % time k-1 of the i-th state at time k.
    pred = zeros(Ns, 1);
    % counter iteratively: (mod(state-1, M^(L1+L2-1)) * M + j).
    newstate = 0;
    for state = 1 : Ns  % All states
        for j = 1 : M   % M times
            % Index of the new state: it's mod(state-1, M^(L1+L2-1)) * M + j
            newstate = newstate + 1;
            if newstate > Ns, newstate = 1; end
            u = u_mat(state, j);
            % updatethe cost of the new state and overwrite the predecessor
                        % if this transition has lower cost than before
            newstate_cost = cost(state) + abs(r_c(k) - u)^2;
            if costnew(newstate) == -1 ...      % not assigned yet, or...
                    || costnew(newstate) > newstate_cost  % ...found path with lower cost
                costnew(newstate) = newstate_cost;
                pred(newstate) = state;
            end
        end
    end
    % Update the survivor sequence by shifting the time horizon of the matrix by one, and
    % rewrite the matrix with the new survival sequences sorted by current state.
    % Meanwhile, decide the oldest sample (based on minimum cost) and get rid of it to
    % keep only Kd columns in the matrix.
    temp = zeros(size(survSeq));
    for newstate = 1:Ns
        temp(newstate, 1:Kd) = ...
            [survSeq(pred(newstate), 2:Kd), ...
            symb(mod(newstate-1, M)+1)];
    end
    [~, decided_index] = min(costnew);  % Find the oldest symbol that yields the min cost
    detectedSymb(1+k) = survSeq(decided_index, 1); % and store it.
    survSeq = temp;
    cost = costnew;
end

detectedSymb(length(r_c)+2 : length(r_c)+Kd) = survSeq(decided_index, 1:Kd-1);
% Use the min cost from the last iteration
detectedSymb = detectedSymb(Kd+1 : end);
detected = detectedSymb;
detected = detected(2:end); % Discard first symbol (time k=-1)
end
```

```
function [detected] = FBA(y, psiD, L1, L2)

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

tStart = tic;
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

% 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);
% the index has to go backwards
for k = K:-1:1
    for i = 1:Ns
        % Index of the state
        possible_state = mod(i-1, M^(L1 + L2 - 1))*M + 1;
        % 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);
%Symbol from which we choose max 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 = 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
```

```
function [pn] = PNSeq(L)

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

% Initial conditions (set to one, arbitrary)
% Must not be ALL zeros
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 20
            pn(l) = xor(pn(l-17), pn(l-20));


    end
end

% Bits are {-1, 1}
pn = 2*pn -1;

end
```

```
function [outsym] = QPSK_detector(insym)

if (real(insym)>0)
    if (imag(insym)>0)
        outsym = 1+1i;
    else
        outsym = 1-1i;
```

```
    end
else
    if (imag(insym)>0)
        outsym = -1+1i;
    else
        outsym = -1-1i;
    end
end

end
```

```
function [output] = bitmap(input)
% Check if the input array has even length
L = length(input);
if (mod(L, 2) ~= 0)
disp('Must input an even length array');
return;
end

output = zeros(L,1);

% Map each couple of values to the corresponding symbol
for idx = 1:2:L-1
    if (isequal(input(idx:idx+1), [-1; -1] ))
        output(idx) = -1-1i;
    elseif (isequal(input(idx:idx+1), [1; -1] ))
        output(idx) = 1-1i;
    elseif (isequal(input(idx:idx+1), [-1; 1] ))
        output(idx) = -1+1i;
    elseif (isequal(input(idx:idx+1), [1; 1] ))
        output(idx) = +1+1i;
    end
end

output = output(1:2:end);
end
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex')

load('Useful.mat');
T = 1;              % Symbol period
Tc = T/4;           % upsampling period
Q = T/Tc;           % Interpolation factor
snr_db = 10;
snr_lin = 10^(snr_db/10);
sigma_a = 2;            % Input variance

L = 1023;                   % Input signal: from a PN sequence generate QPSK
x = PNSeq(L);
in_bits = bitmap(x(1:length(x)-1));
```

```
function [output, sigma_w, qc] = channel_sim(x, snr, sigma_a)

T = 1;
Tc = T/4;
```

```
Q = T/Tc;

alpha = 0.67;
beta = 0.7424;

snr_db = snr;
snr_lin = 10^(snr_db/10);

qc_num = [0 0 0 0 0 beta];
qc_denom = [1 -alpha];
qc = impz(qc_num, qc_denom);
qc = [0; 0; 0; 0; 0; qc(qc >=max(qc)*10^(-2))];
E_qc = sum(qc.^2);

sigma_w = sigma_a * E_qc / snr_lin;

a_prime = upsample(x,Q);

s_c = filter(qc_num, qc_denom, a_prime);
r_c = s_c;
output = r_c;
```

```
function Hd = GAA_filter
% All frequency values are normalized to 1.
Fpass = 0.45;           % Passband Frequency
Fstop = 0.55;           % Stopband Frequency
Dpass = 0.05;                   % Passband Ripple
Dstop = 0.01;           % Stopband Attenuation
dens  = 20;             % Density Factor
[N, Fo, Ao, W] = firpmord([Fpass, Fstop], [1 0], [Dpass, Dstop]);
g_AA  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(g_AA);
save('GAA_filter.mat');
```

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

% CREATE INPUT BITS, QC AND NOISE

alpha = 0.67;
beta = 0.7424;
qc_num = [0 0 0 0 0 beta];
qc_denom = [1 -alpha];
qc = impz(qc_num, qc_denom);
qc = [0; 0; 0; 0; 0; qc(qc >=max(qc)/100)];
E_qc = sum(qc.^2);

length_seq = 2^20-1;

SNR_vect = 8:14;
SNR_lin = 10.^(SNR_vect ./ 10);

sigma_w = zeros(length(SNR_vect),1);
sigma_a = 2;

w = zeros(2*length_seq-2, 7);
```

```
for i=1:length(SNR_vect)
    sigma_w(i) = E_qc * sigma_a / SNR_lin(i);
    w(:,i) = wgn(2*length_seq-2,1, 10*log10(sigma_w(i)), 'complex');
end

in_seq = PNSeq(length_seq);
in_seq = in_seq(1:end-1);
in_bits = bitmap(in_seq);
% save('Useful.mat', 'w', 'in_bits', 'qc', 'E_qc');
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

load('Useful.mat');

sigma_a = 2;
sigma_w = sigma_a / 10;
M = 4;

gm = conj(qc(end:-1:1));
h = conv(qc,gm);
h = h(h>max(h)/100);
h = h(3:end-2);
h_T = downsample(h,4);
t0_bar = length(gm);
r_gm = xcorr(gm,gm);
rw_tilde = sigma_w/4 .* downsample(r_gm, 4);
N2 = floor(length(h_T)/2);
N1 = N2;

M1_span = 2:20;
D_span = 2:20;
M2 = 0;

Jvec = zeros(19);
for k=1:length(M1_span)
    for l=1:length(D_span)
        M1 = M1_span(k);
        D = D_span(l);
        [c, Jmin] = Adaptive_DFE(h_T, rw_tilde, sigma_a, M1, M2, D);
        Jvec(k,l) = Jmin;
    end
end

figure, mesh(2:20, 2:20, reshape((Jvec(:, :)), size(Jvec(:, :), 2), size(Jvec(:, :)
    , 2)))
title('$J_{min}$ for LE, SNR = 10 (dB)'); view(160,20);
xlim([2 20]); ylim([2 20]);
xlabel('D'), ylabel('M1'), zlabel('Jmin (dB)')
[min, idx] = min(Jvec(:));

[idx_d, idx_m1] = ind2sub(size(Jvec), idx);
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');
```

```
load('Useful.mat');

sigma_a = 2;
sigma_w = sigma_a / 10;
M = 4;

gm = conj(qc(end:-1:1));
h = conv(qc,gm);
h = h(h>max(h)/100);
h = h(3:end-2);
h_T = downsample(h,4);
t0_bar = length(gm);
r_gm = xcorr(gm,gm);
rw_tilde = sigma_w/4 .* downsample(r_gm, 4);
N2 = floor(length(h_T)/2);
N1 = N2;

M1_span = 2:20;
D_span = 2:20;

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] = Adaptive_DFE(h_T, rw_tilde, sigma_a, M1, M2, D);
        Jvec(k,l) = Jmin;
    end
end

figure, mesh(2:20, 2:20, reshape((Jvec(:, :)), size(Jvec(:, :), 2), size(Jvec(:, :)
    , 2)))
title('$J_{min}$ for DFE, SNR = 10 (dB)'); view(160,20);
xlim([2 20]); ylim([2 20]);
xlabel('D'), ylabel('M1'), zlabel('Jmin (dB)')

[min, idx] = min(Jvec(:));

[idx_d, idx_m1] = ind2sub(size(Jvec), idx);
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

load('Useful.mat');
load('GAA_filter.mat');
sigma_a = 2;
sigma_w = sigma_a / 10;
M = 4;

qg_up = conv(qc, g_AA);
qg_up = qg_up.';
t0_bar = find(qg_up == max(qg_up));

qg = downsample(qg_up(1:end), 2);
g_m = conj(flipud(qg));
```

```
h = conv(qg, g_m);
h = h(h ~= 0);
N0 = (sigma_a * 1) / (4 * 10);

r_g = xcorr(conv(g_AA, g_m));
r_w = N0 * downsample(r_g, 2);

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

M1_span = 2:20;
D_span = 2:20;

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_frac(h, r_w, sigma_a, M1, M2, D, N1, N2);
        Jvec(k,l) = Jmin;
    end
end

figure, mesh(2:20, 2:20, reshape((Jvec(:, :)), size(Jvec(:, :), 2), size(Jvec(:, :)
    , 2)))
title('$J_{min}$ for AA and Matched Filter, SNR = 10 (db)'); view(170,25);
xlim([2 20]); ylim([2 20]);
xlabel('D'), ylabel('M1'), zlabel('Jmin (dB)')

[min, idx] = min(Jvec(:));

[idx_d, idx_m1] = ind2sub(size(Jvec), idx);
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

load('Useful.mat');
load('GAA_filter.mat');
sigma_a = 2;
sigma_w = sigma_a / 10;
M = 4;

qg_up = conv(qc, g_AA);
qg_up = qg_up.';
t0_bar = find(qg_up == max(qg_up));
qg = downsample(qg_up, 2);
h = qg;
r_g = xcorr(g_AA);
N0 = (sigma_a * 1) / (4 * 10);
r_w = N0 * downsample(r_g, 2);
N2 = floor(length(h)/2);
N1 = N2;

M1_span = 2:20;
```

```
D_span = 2:20;

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_frac(h, r_w, sigma_a, M1, M2, D, N1, N2);
        Jvec(k,l) = Jmin;
    end
end

figure, mesh(2:20, 2:20, reshape((Jvec(:, :)), size(Jvec(:, :), 2), size(Jvec(:, :)
    , 2)))
title('$J_{min}$ for AA without Matched Filter, SNR = 10 (dB)'); view(170,25);
xlim([2 20]); ylim([2 20]);
xlabel('D'), ylabel('M1'), zlabel('Jmin (dB)')

[min, idx] = min(Jvec(:));

[idx_d, idx_m1] = ind2sub(size(Jvec), idx);
```

```
clc; close all; clear global; clearvars;
set(0,'defaultTextInterpreter','latex');

SNR_vect = 8:14;
load('Pe_LE_avgs.mat');
load('PE_DFE_avgs.mat');
load('Pe_AA_GM_avgs.mat');
load('Pe_AA_NOGM_avgs.mat');
load('Pe_VA_avgs.mat');
load('Pe_FBA.mat');
load('Pe_AWGN_SIM_avgs.mat');

figure()
semilogy(SNR_vect, Pe_LE_avg, 'b--');
hold on; grid on;
semilogy(SNR_vect, Pe_DFE_avg,'b');
semilogy(SNR_vect, Pe_AA_GM_avg, 'k--');
semilogy(SNR_vect, Pe_AA_NOGM_avg, 'k');
semilogy(SNR_vect, Pe_VA_avg, 'r--');
semilogy(SNR_vect, Pe_FBA, 'r');
semilogy(SNR_vect, Pe_AWGN_SIM_avg, 'g--');
semilogy(SNR_vect, awgn_bound, 'g');
ylim([10^-4 10^-1]); xlim([8 14]);
xlabel('SNR'); ylabel('$P_e$');
legend('MF+LE@T','MF+DFE@T','AAF+MF+DFE@$\frac{T}{2}$','AAF+DFE@$\frac{T}{2}$',...
    'VA','FBA','MF b-S','MF b-T');
set(legend,'Interpreter','latex');
```