4 OCTAVE/MATLAB Problem 2.4 - Gradient Algorithms

Task a)
For this task the Least Mean Square algorithms (LMS) shall be implemented using the derived formulas from the problem class. LMS and NMLS algorithms assume ergodic processes, which allow to replace the expectation operator by a time average of one WSS realization of the process to get the mean and variance values.

The formulas are for **LMS:**

$$\mathbf{c}[n] = \mathbf{c}[n-1] + \mu e^*[n]\mathbf{x}[n] \tag{1}$$

where

$$e[n] = d[n] - y[n] = d[n] - \mathbf{c}^H[n-1]\mathbf{x}[n] \tag{2}$$

and for **NMLS:**

$$\mathbf{c}[n] = \mathbf{c}[n-1] + \frac{\hat{\mu}}{\alpha + \mathbf{x}^H[n]\mathbf{x}[n]} e^*[n]\mathbf{x}[n] \tag{3}$$

where α is a small positive constant to avoid divison by zero.

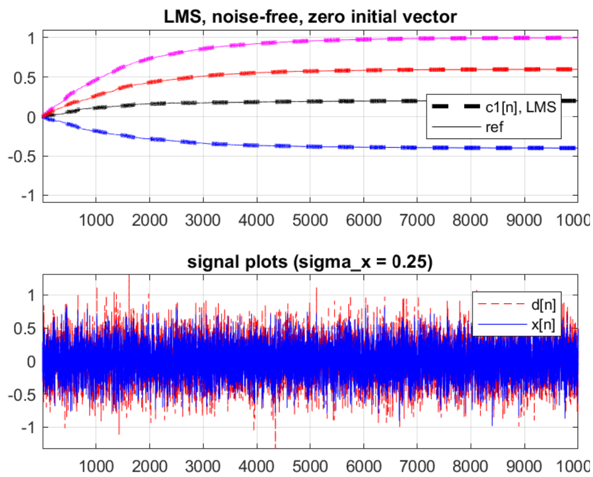

Figure 1: Coefficients for LMS

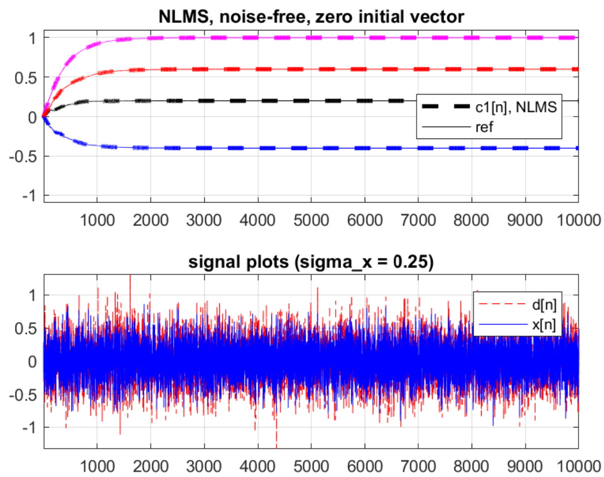

Figure 2: Coefficients for NLMS

Task b)
This time the gradient search should be implemented. This algorithm makes use of the Autocorrelationmatrix **Rxx** and the cross-correlation vector **p**, which are often not know to us (only through estimates), hence why the LMS and NMLS algorithms exist.

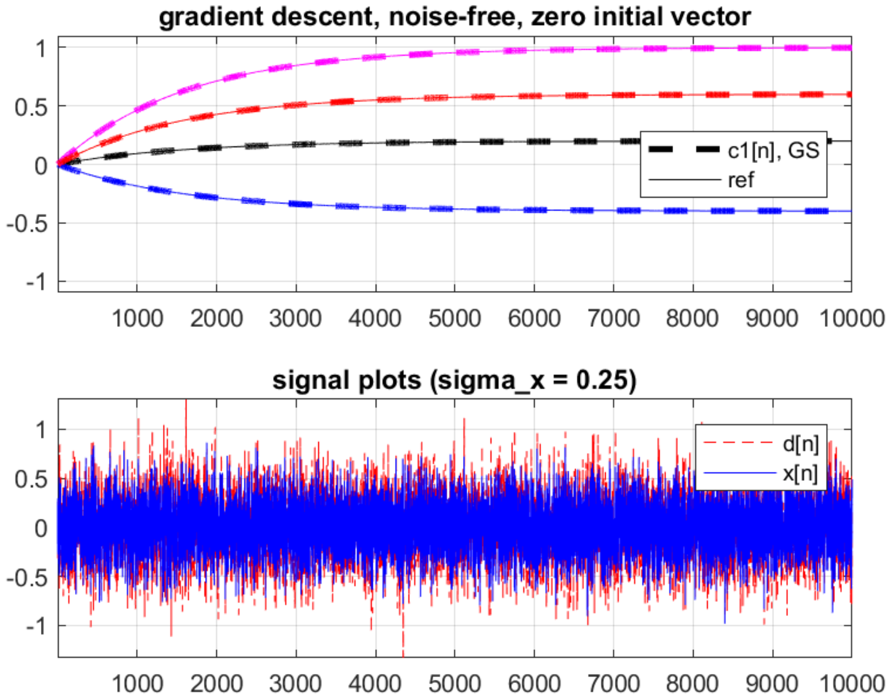

Figure 3: Coefficients for gradient descent

Task c)
In the last task the different gradient algorithms should be compared in a noise environment.

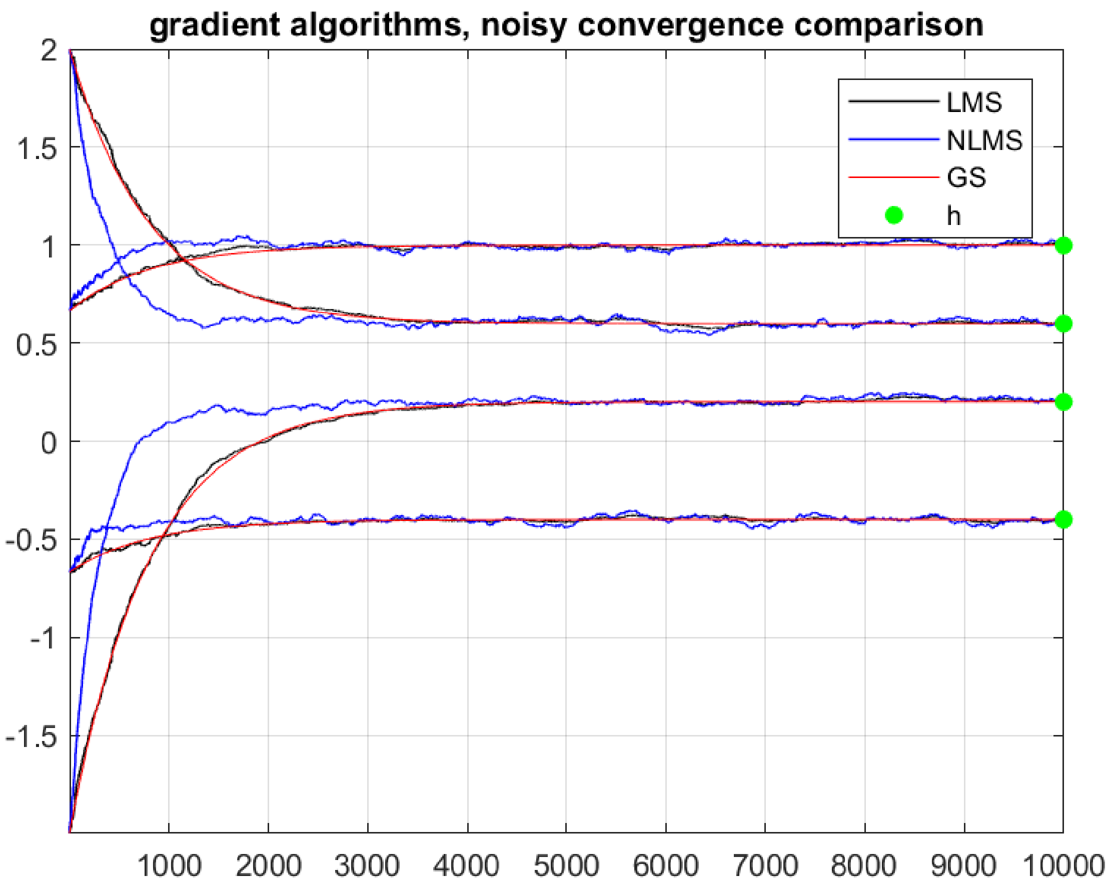

Figure 4: Comparsion of gradient algorithms with noise

LMS: The additional noise causes the LMS to fluctuate a lot. It looks like it will never truly converge to h. In the problem class we derived, that it converges to h (if it converges in the first place) for noise free environment. BUT if there is some noise, e[n] wont be 0 in the optimum point, hence it never truly converges

NMLS: Due to the independent step size to signal energy the NMLS overshoots the LMS and GS, especially in the beginning and like the LMS it can never truly converge if there is some noise.

GS: Gradient search creates a smooth looking curve with no fluctuation what so ever and converges towards h really nicely. It looks like the noise does not affect the algorithm. The downside of GS is the fact that it needs the values of cross correlation **p** and Autocorrelationmatrix **Rxx**, which are often not known and can only be estimated.

4.1 Matlab Code

Main File

```
% Assignment 2, (2.4)
% testing script for stochastic and deterministic gradient algorithms
% Adaptive System UE WS 19/20
% Thomas Wilding, SPSC
clear
close all
clc

%suppres: "Warning: Directory already exists."
id = 'MATLAB:MKDIR:DirectoryExists';
warning('off',id)

mkdir 'Figures' %create Figures folder


%%%% Reference data creation: (1)
% Nx = 1e4; %number of samples
%
% h  = [0.2 -0.4 1 0.6]';  %impulse response of unknown system
% Nh = length(h);
% Nc  = Nh;                %adaptive filter order is order of unknown system
%
% rng(1) %set random number generator
% sigma_x = 0.25; %standard deviation of x
% x = sigma_x*randn(1,Nx);
% d = filter(h,1,x);
%
% save('./data_rng1.mat','x','sigma_x','d','h','Nh','Nc','Nx')


%%%% Reference data creation: (2)
% rng(2) %set random number generator
% sigma_x = 0.25; %standard deviation of x
% x = sigma_x*randn(1,Nx);
% d = filter(h,1,x);
%
% sigma_w = 0.1;
% d  = d+sigma_w*randn(1,Nx);
%
% save('./data_rng2.mat','x','sigma_x','d','h','Nh','Nc','Nx')


%% LMS, noise-free, zero initial vector (with reference)
clear
load('./data_rng1.mat')

c0 = zeros(1,Nc);     %initialize with zero vector

mu = 0.01;
alpha = 0;
[~,~,c_lms]  = lms_algorithm(x,d,Nc,mu,alpha,0); %standard LMS

% save('./ref_lms.mat','c_lms')
ref_lms = load('./ref_lms.mat');
```

```

figure(1)
subplot(2,1,1), hold on, grid on, box on
plot(c_lms(1,:),'--k','LineWidth',2.5), plot(ref_lms.c_lms(1,:),'k','LineWidth',0.5)
plot(c_lms(2,:),'--b','LineWidth',2.5), plot(ref_lms.c_lms(2,:),'b','LineWidth',0.5)
plot(c_lms(3,:),'--m','LineWidth',2.5), plot(ref_lms.c_lms(3,:),'m','LineWidth',0.5)
plot(c_lms(4,:),'--r','LineWidth',2.5), plot(ref_lms.c_lms(4,:),'r','LineWidth',0.5)
legend('c1[n], LMS','ref','Location','east')
xlim([1 Nx]), ylim([-1.1 1.1])
title('LMS, noise-free, zero initial vector')
subplot(2,1,2), hold on, grid on, box on
plot(d,'--r'), plot(x,'b')
axis tight
legend('d[n]','x[n]')
title(sprintf('signal plots (sigma_x = %2.2f)',sigma_x),'Interpreter','none')

saveas(gcf,'Figures/LMS', 'epsc')


%% NLMS, noise-free, zero initial vector (with reference)
c0 = zeros(1,Nc);      %initialize with zero vector

mu = 0.01;
alpha = 0;
[~,~,c_nlms]  = lms_algorithm(x,d,Nc,mu,alpha,1); %normalized LMS

% save('./ref_nlms.mat','c_nlms')
ref_nlms = load('./ref_nlms.mat');

figure(2)
subplot(2,1,1), hold on, grid on, box on
plot(c_nlms(1,:),'--k','LineWidth',2.5), plot(ref_nlms.c_nlms(1,:),'k','LineWidth',0.5)
plot(c_nlms(2,:),'--b','LineWidth',2.5), plot(ref_nlms.c_nlms(2,:),'b','LineWidth',0.5)
plot(c_nlms(3,:),'--m','LineWidth',2.5), plot(ref_nlms.c_nlms(3,:),'m','LineWidth',0.5)
plot(c_nlms(4,:),'--r','LineWidth',2.5), plot(ref_nlms.c_nlms(4,:),'r','LineWidth',0.5)
legend('c1[n], NLMS','ref','Location','east')
xlim([1 Nx]), ylim([-1.1 1.1])
title('NLMS, noise-free, zero initial vector')
subplot(2,1,2), hold on, grid on, box on
plot(d,'--r'), plot(x,'b')
axis tight
legend('d[n]','x[n]')
title(sprintf('signal plots (sigma_x = %2.2f)',sigma_x),'Interpreter','none')

saveas(gcf,'Figures/NLMS', 'epsc')

%% NLMS, noise-free, zero initial vector (with reference)
mu = 0.01;
alpha = 0;

```

```
Rxx = sigma_x^2*eye(Nc); %x is white noise?
p = Rxx*h;

[~,~,c_gs]  = gd_algorithm(x,d,Nc,mu,Rxx,p); %gradient descent

% save('./ref_gd.mat','c_gd')
ref_gd = load('./ref_gd.mat');

figure(3)
subplot(2,1,1), hold on, grid on, box on
plot(c_gs(1,:),'--k','LineWidth',2.5), plot(ref_gd.c_gs(1,:),'k','LineWidth'
    ,0.5)
plot(c_gs(2,:),'--b','LineWidth',2.5), plot(ref_gd.c_gs(2,:),'b','LineWidth'
    ,0.5)
plot(c_gs(3,:),'--m','LineWidth',2.5), plot(ref_gd.c_gs(3,:),'m','LineWidth'
    ,0.5)
plot(c_gs(4,:),'--r','LineWidth',2.5), plot(ref_gd.c_gs(4,:),'r','LineWidth'
    ,0.5)
legend('c1[n], GS','ref','Location','east')
xlim([1 Nx]), ylim([-1.1 1.1])
title('gradient descent, noise-free, zero initial vector')
subplot(2,1,2), hold on, grid on, box on
plot(d,'--r'), plot(x,'b')
axis tight
legend('d[n]','x[n]')
title(sprintf('signal plots (sigma_x = %2.2f)',sigma_x),'Interpreter','none')

saveas(gcf,'Figures/GD', 'epsc')

%% algorithm comparison: noisy, random intial vector (no references)
clear
load('./data_rng2.mat')

Rxx = sigma_x^2*eye(Nc);
p = Rxx*h;

c0 = linspace(-2,2,Nc); %pseudo-random initialization

mu = 0.02;
alpha = 0.1;
[~,~,c_lms]  = lms_algorithm(x,d,Nc,mu,alpha,0,c0);
[~,~,c_nlms] = lms_algorithm(x,d,Nc,mu,alpha,1,c0);
[~,~,c_gs]   = gd_algorithm(x,d,Nc,mu,Rxx,p,c0); %gradient search

figure(4), hold on, grid on, box on
hlms = plot(c_lms.','k','LineWidth',0.75);
hnlms = plot(c_nlms.','b','LineWidth',0.75);
hgs = plot(c_gs.','r');
htrue = scatter(Nx*ones(Nh,1),h,'g','o','filled');
axis tight
legend([hlms(1),hnlms(1),hgs(1),htrue],{'LMS','NLMS','GS','h'})
title('gradient algorithms, noisy convergence comparison')

saveas(gcf,'Figures/comparison', 'epsc')

% PLOT DESCRIPTION
%
% LMS:
```

```
% The additional noise causes the LMS to flucatute a lot it looks like it
% will never truly converge to h. In the problem class we derived, that it
% converges to h (if it converges in the first place) for noise free
% enviroment. BUT if there is some noise, e[n] wont be 0 in the optimum
% point, hence it never truly converges
%
% NMLS:
% Due to the independet step size to signal energy the NMLS overshoots the
% LMS and GS, especially in the beginning and like the LMS it looks like it
% can never truly converge if there is some noise.
%
% GS:
% Gradient search creates a smooth looking curve with no fluctuation what so
% ever and converges towards h really nicely. It looks like the noise does
% not effect the algorithm. The downside of GS is the fact that it needs the
% values of cross corellation p and Autocorrelationmatrix Rxx, which are
% often not known.
%
% LMS and NMLS algorithms assume ergodic process, which allow to replace
% expectation operator by an time average of one WSS realization for mean
% and variance
```

Function lms_algorithm()

```
function [y,e,c] = lms_algorithm(x,d,N,mu,alpha,OPTS,c0)
% INPUTS: % x ....... input signal vector (column vector)
% d ....... desired output signal (of same dimensions as x)
% N ....... number of filter coefficients
% mu ...... step-size parameter
% alpha ... algorithm dependent parameter
% OPTS .... 0 for standard LMS, 1 for normalized LMS
% c0 ...... initial coefficient vector (optional column vector; default all zeros)
% OUTPUTS:
% y ....... output signal vector (same length as x)
% e ....... error signal vector (same length as x)
% c ....... coefficient matrix (N rows, number of columns = length of x)


%formulas from problem class sheets, page 9

if nargin < 7 %check if c0 is given, if not initialize with 0
   c0 = zeros(1,N);
end

if OPTS == 1
    norm_x = 1; %if the NMLS was chosen, the norm of the signal energy does not affect the update coefficient value
else
    norm_x = x(:)'*x(:);
end

if ~(0 < mu/norm_x && mu/norm_x < 2)
    error('Step size mu causes the system to be unstable')
end

%make sure, everything is a column vector
x = x(:);
d = d(:);
c0 = c0(:);

%pad for time instances n < 0
x_pad = [zeros(N-1,1); x];
d_pad = [zeros(N-1,1); d];%same for d to keep things in order

%create placeholders; after calucation elide the appened zeroes in the beggining
y = zeros(size(x_pad));
e = zeros(size(x_pad));
c = zeros(N,length(x_pad));

%intialization for loop
c(:,N-1) = c0; %first iteration uses c0, hence we need to write it into c
mu_calc = mu; %mu for standard LMS; if OPT == 1, it gets overwritten within for loop
for n = N:length(x_pad)

    x_tap = flip(x_pad(n-N+1:n));
    y(n) = c(:,n-1)'*x_tap;
    e(n) = d_pad(n) - y(n); %' means hermitian transposed

```

```
    %change mu depending on chosen OPTS (standard or normalized LMS)
    if OPTS == 1 %normalized LMS
        mu_calc = mu/(alpha + x_tap'*x_tap); %only the energy of the observed
            current signal
    end

    c(:,n) = c(:,n-1) + mu_calc*conj(e(n))*x_tap;

end

%now delete the first entries of y,e and c which are zero, to keep the time
%indices in order
y(1:N-1) = [];
e(1:N-1) = [];
c(:,1:N-1) = [];

end
```

Function gd_algorithm()

```
function [y,e,c] = gd_algorithm(x,d,N,mu,Rxx,p,c0)
% INPUTS: % x ....... input signal vector (column vector)
% d ....... desired output signal (of same dimensions as x)
% N ....... number of filter coefficients
% mu ...... step-size parameter
% Rxx ..... autocorrelation matrix
% p ....... cross-correlation vector (column vector)
% c0 ...... initial coefficient vector (optional column vector; default all zeros)
% OUTPUTS:
% y ....... output signal vector (same length as x)
% e ....... error signal vector (same length as x)
% c ....... coefficient matrix (N rows, number of columns = length of x)


if nargin < 7 %check if c0 is given, if not initialize with 0
    c0 = zeros(1,N);
end

x = x(:); %make sure, it is a column vector
d = d(:);
c0 = c0(:);
p = p(:);

x_pad = [zeros(N-1,1); x];%pad for time instances n < 0
d_pad = [zeros(N-1,1); d];%same for d to keep things in order

y = zeros(size(x_pad)); %create placeholders; after calucation elide the appened zeroes in the beggining
e = zeros(size(x_pad));
c = zeros(N,length(x_pad));


%dont know what this sentence means: Be careful, that in this form the signal statistics are estimated beforehand and not adapted/changed during execution.

c(:,N-1) = c0; %first iteration uses c0, hence we need to write it into c
for n = N:length(x_pad)

    x_tap = flip(x_pad(n-N+1:n)); %flip, so the is value at time n is at the top of the vector
    y(n) = c(:,n-1)'*x_tap;
    e(n) = d_pad(n) - y(n); %' means hermitian transposed

    c(:,n) = c(:,n-1) + mu*(p - Rxx * c(:,n-1)); %changed to update rule for Gradient Search

end


%now delete the first entries of y,e and c which are zero, to keep the time
%indices in order
y(1:N-1) = [];
e(1:N-1) = [];
c(:,1:N-1) = [];
```

```
end
```