5 OCTAVE/MATLAB Problem 2.5 - Performance Analysis

Task a)
In this task the effect of the step size parameter μ should be investigated for the LMS- and GD-Algorithm for $\mu = \{0.0001, 0.001, 0.01, 1\}$. To visualise this better two formulas were introduced to display the misalignment and error.

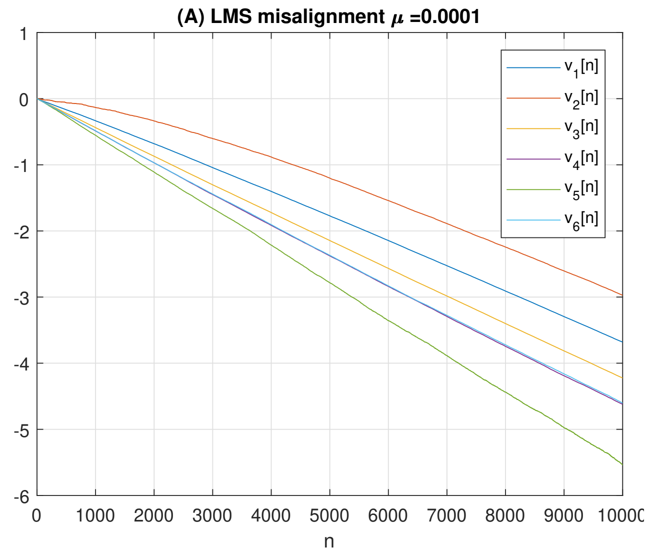
The first formula is:

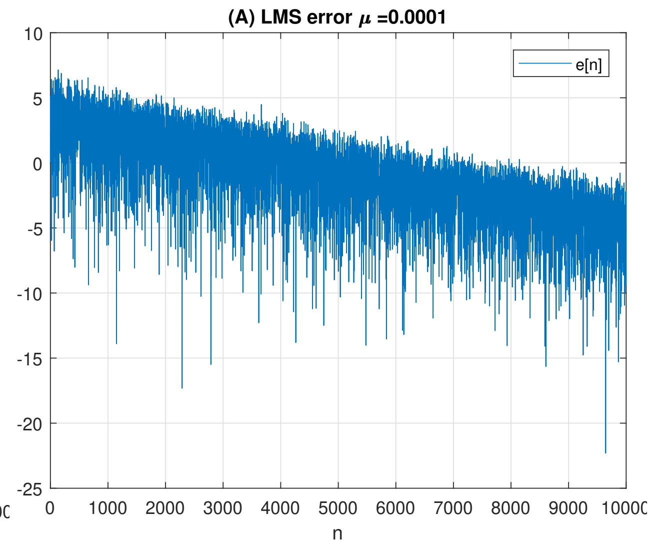
$$\ln\left(\frac{|\mathbf{E}\{v_k[n]\}|}{|\mathbf{E}\{v_k[0]\}|}\right) \tag{1}$$

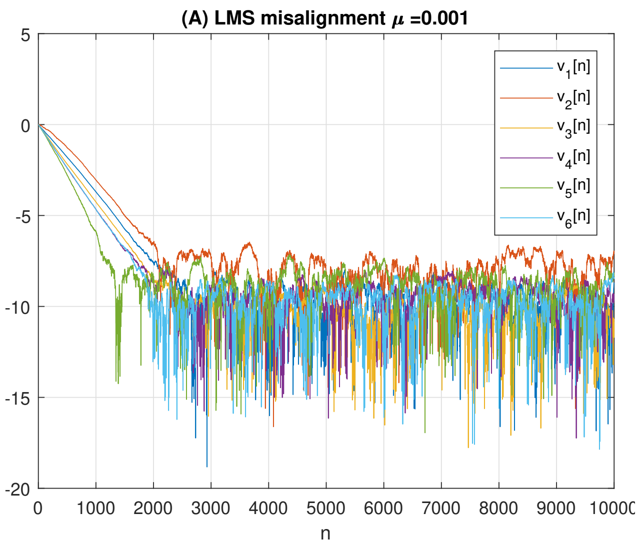
and the second one:

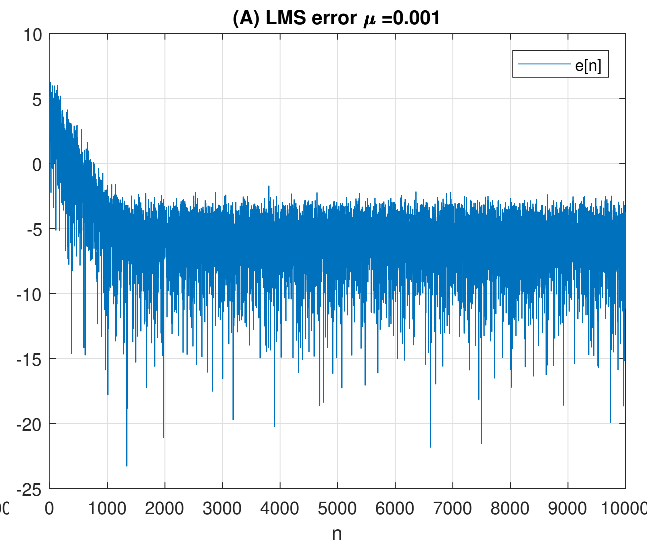
$$\ln\left(\frac{\mathbf{E}\{e^2[n]\}}{\mathbf{E}\{e^2[0]\}}\right) \tag{2}$$

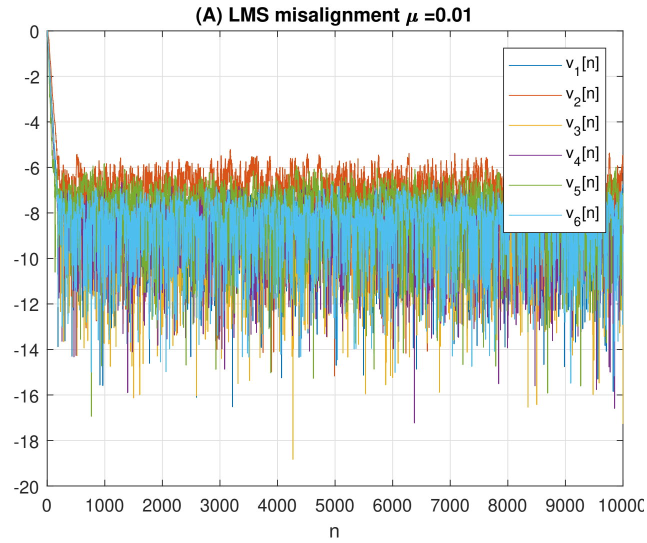
The plots for varying step size of the LMS algorithm:

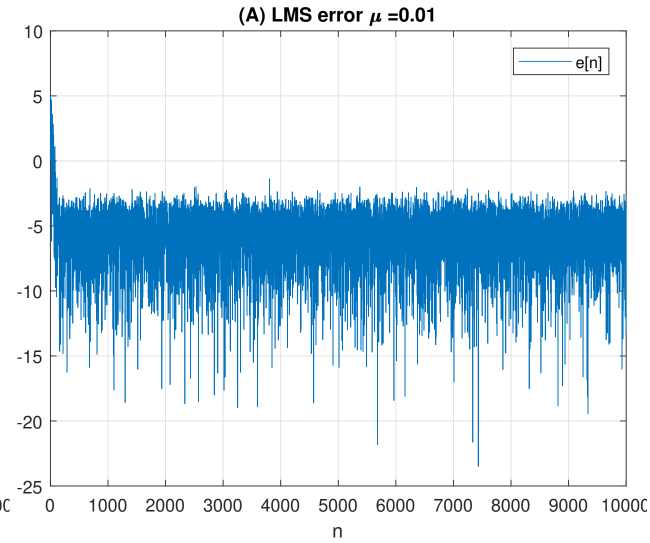


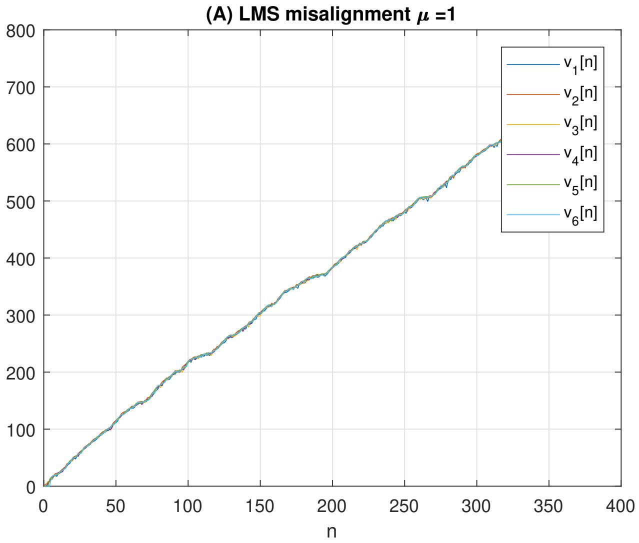


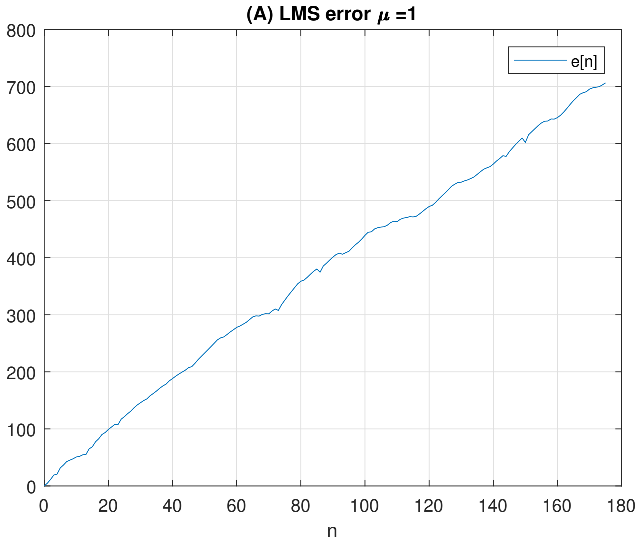












The first pair of plots shows the values of the above mentioned formulas for the LMS algorithm using a step size of $\mu = 0.0001$. As the first formula builds a fraction of $\mathbf{v}[n]$ and then the logarithm, the lower the numbers displayed in the misalignment plot, the better it the algorithm converges. The second formula follows the same process.

Now for the discussion of the plots:

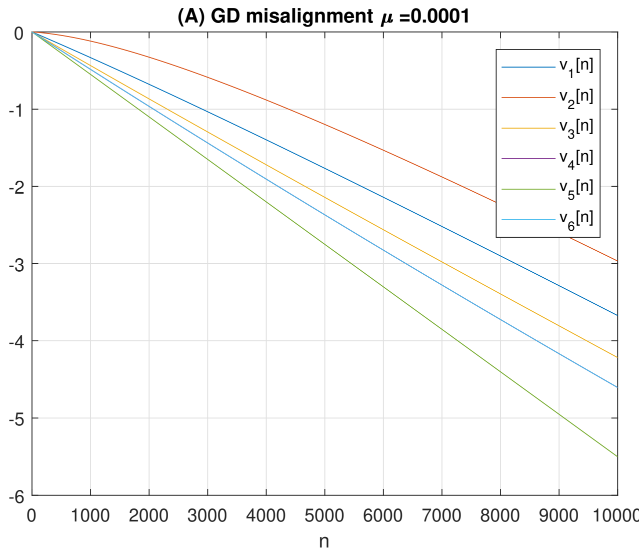
LMS for $\mu = 0.0001$:
The misalignment coefficients keep decreasing within the plot as well the error and the given amount of samples is not enough to reach the point of convergence.

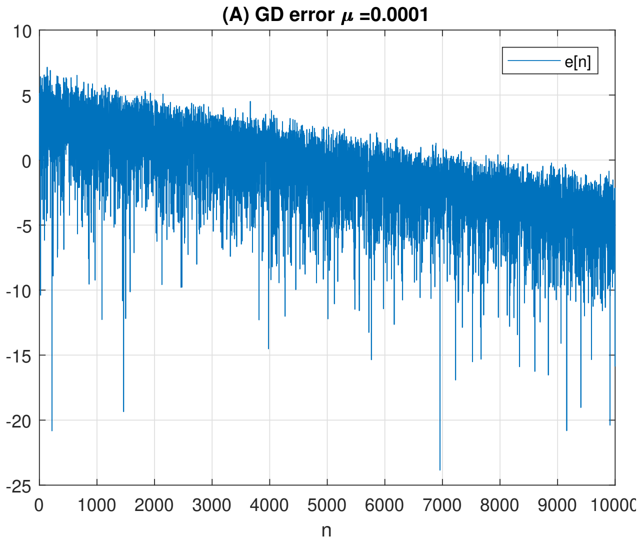
LMS for $\mu = 0.001$:
This time the misalignment coefficients do converge, since after about 2000 samples the coefficients and the error don't keep decreasing anymore. The step size is big enough to allow convergence within the time(i.e. samples).

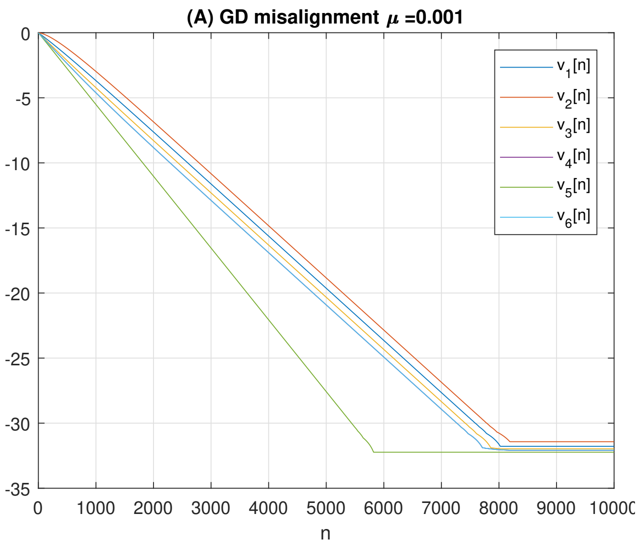
LMS for $\mu = 0.01$:
Both plots reached their 'converged' state much earlier now due to the higher step size μ and there is no sign of divergence due to the large step size.

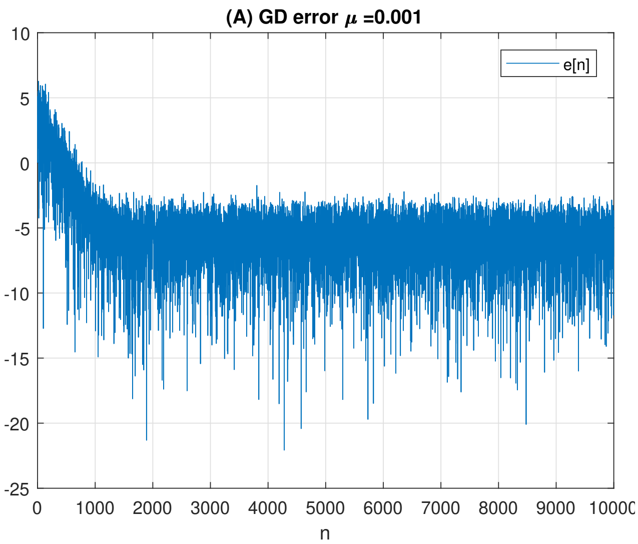
LMS for $\mu = 1$:
Now the step size increased dramatically and the LMS algorithm does not converge anymore. The value of the gradient from the start point is too big, hence we overshoot the valley of the cost function by far and end up at point with an even bigger gradient. This keeps repeating until Matlab displays NaN (=Not a Number) and the plots stop at those samples.

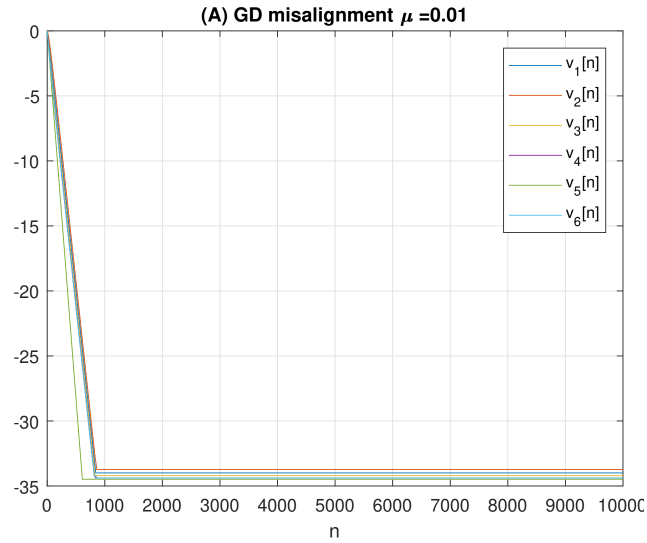
The plots for varying step size of the GD algorithm:

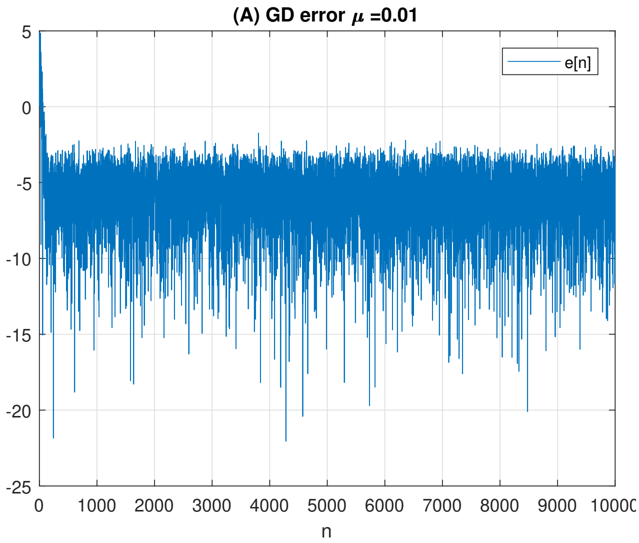


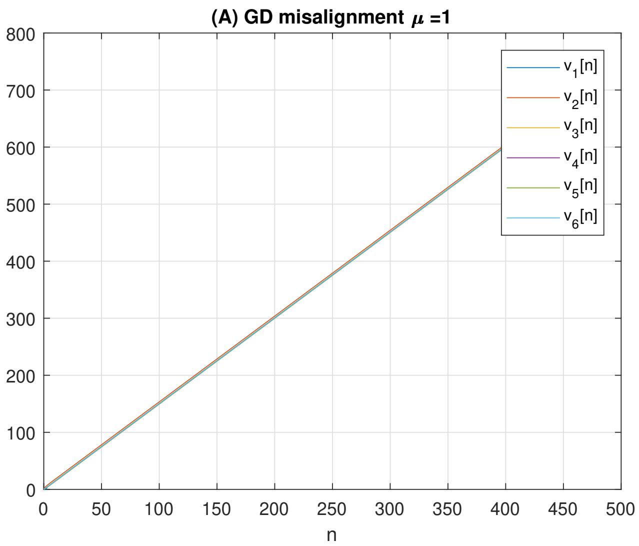


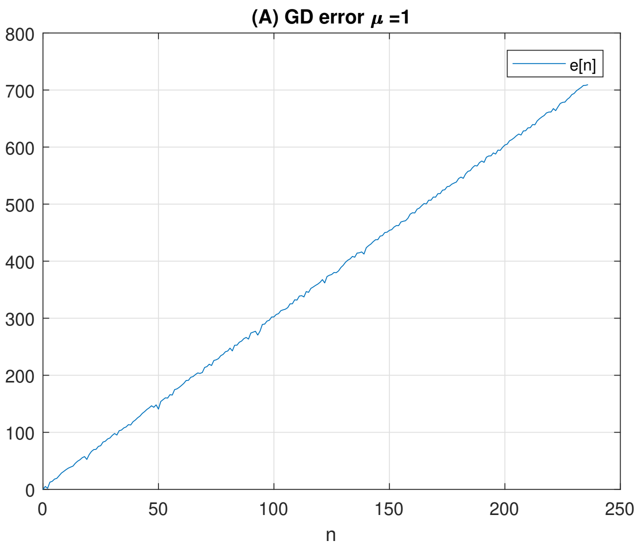












Discussion of the GD plots:

GD for $\mu = 0.0001$:
These plots look quite similar to the LMS one and are not 'converged' yet.

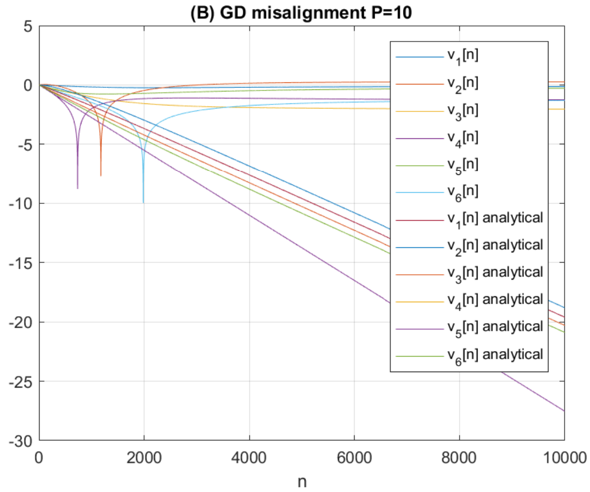
GD for $\mu = 0.001$:
Increasing the step size helps the algorithm to 'converge' much faster and in the GD case the misalignment coefficients get much smaller than in the LMS case. And after 'convergence' there is no random fluctuations for the misalignment, they show a straight line. It should also be noted, that the error looks much earlier converged than the misalignment.

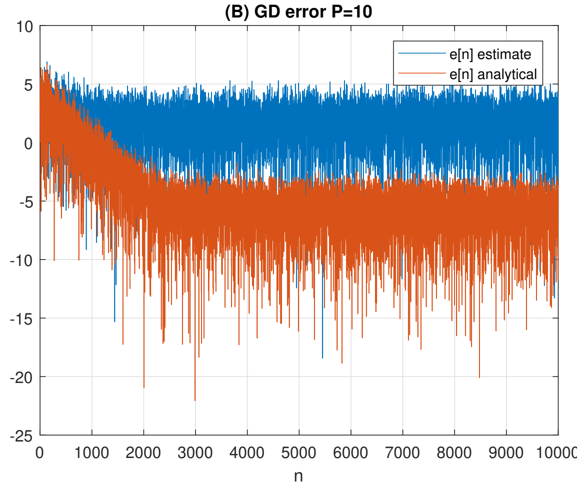
GD for $\mu = 0.01$:
Similar to LMS, the algorithm converges much faster than before.

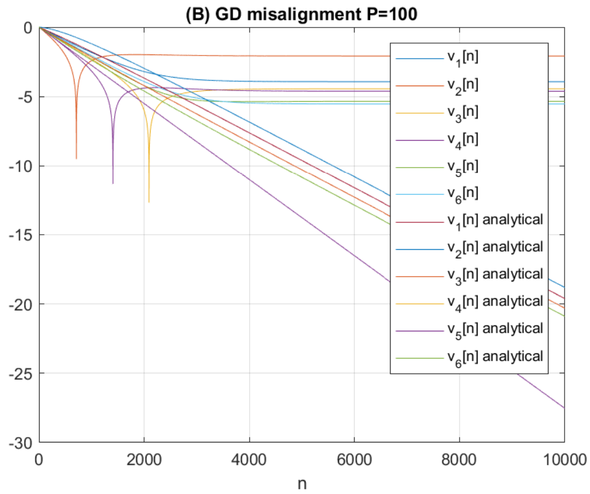
GD for $\mu = 1$:
Similar to LMS, the algorithm diverges.

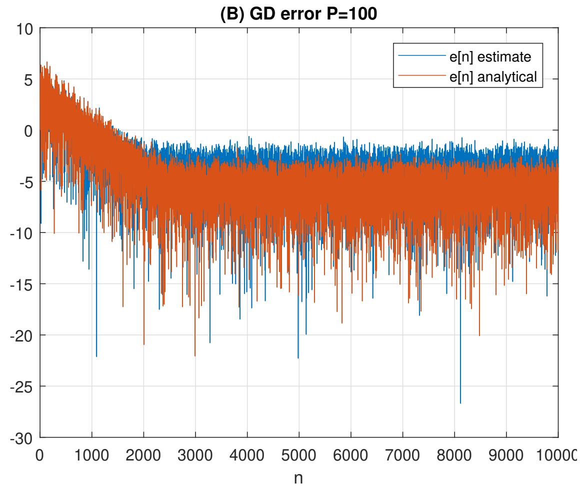
Task b)

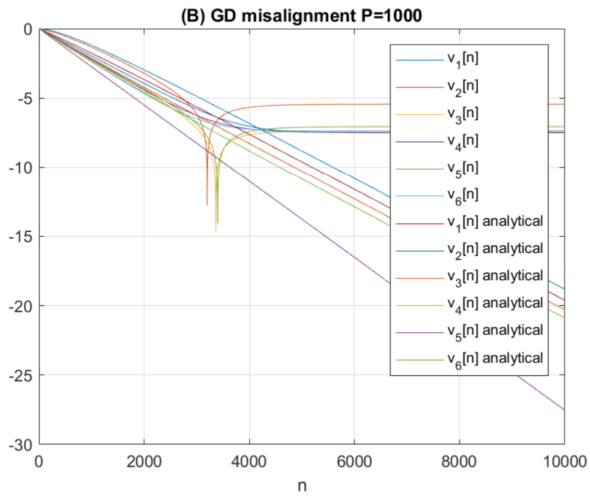
Now the effect of amount of samples should be investigated for a given stepsize $\mu = 0.0005$ using the GD algorithm. Each plot displays the values, once for the analytical solutions for **Rxx** and **p** and once for the estimated statistics.

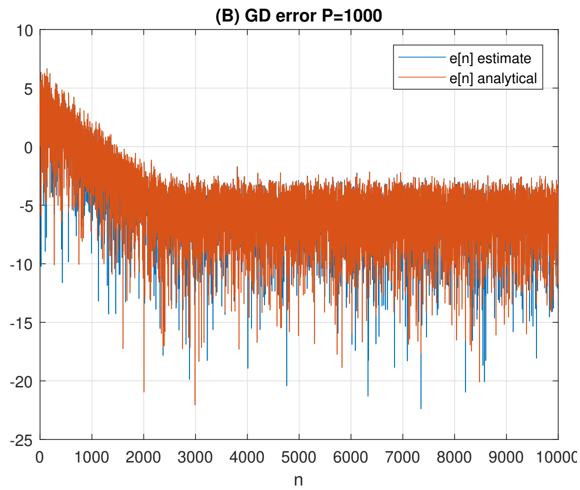






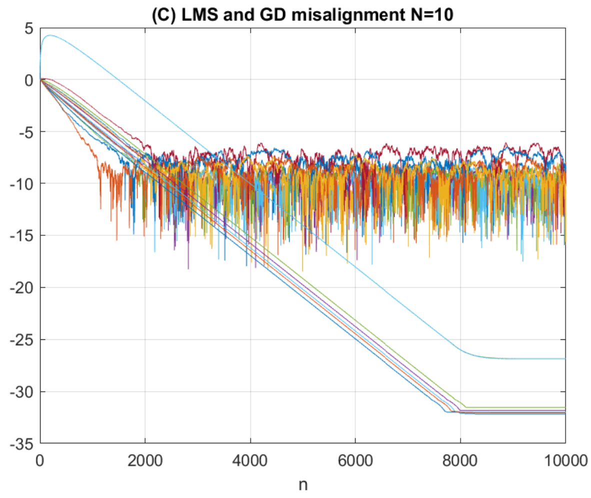


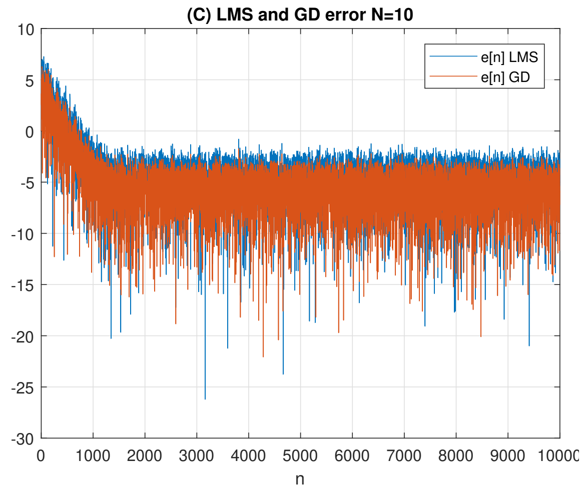


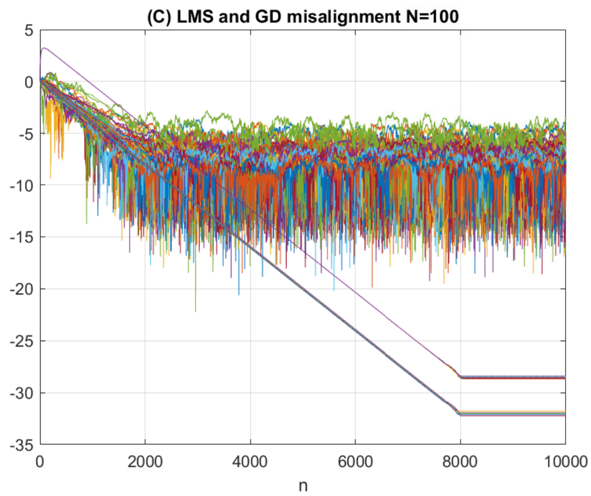


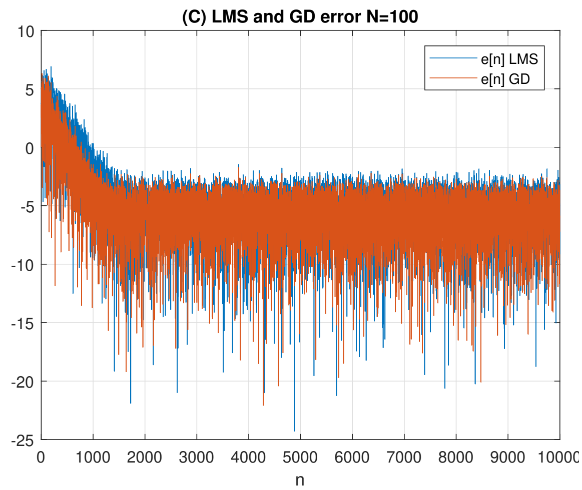
From the plots we can see, the lower the sample size P, the worse the misalignment coefficients and the error is. The smaller sample size causes the sample correlations **Rxx** and **p** to be less precise and therefore also GD is less precise. It should be noted, that the analytical solution does not converge in time, due to the small step size and finite amount of samples.

Task c)
For the third task, the order number N should be varied $N = \{10, 100\}$ using the LMS and GD. Both results (GD and LMS) are plotted in one plot for the misalignment and the error.









The legend entries are not displayed in the misalignment plot, because it would be way too much (would be 20 entries for $N = 10$ and 200 for $N = 100$). Just like in the plots before the fluctuating coefficients belong to LMS and the other one to GD.

The LMS algorithm gets worse when increasing order number N, which is displayed by the misalignment and the error, but over modelling has only a small effect on GD.

Task d)
Update equation for **LMS**:

$$\mathbf{c}[n] = \mathbf{c}[n-1] + \mu \cdot e[n] \cdot \mathbf{x}[n] \tag{3}$$

As we can see the update equation does depend on x[n] directly (unlike GD), so the fluctuations of x[n] can be seen in the LMS coefficients too. Due to this, for each realization the LMS can vary.

Update equation for **GD**:

$$\mathbf{c}[n] = \mathbf{c}[n-1] + \mu \cdot \left(\mathbf{p} - \mathbf{R}_{xx} \cdot \mathbf{c}[n-1]\right) \tag{4}$$

The GD makes use of the statistics of x[n] ($\mathbf{R}_{xx}, \mathbf{p}$) and not x[n] directly, so the fluctuations of x[n] don't affect the coefficients directly. Using enough samples to estimate the statistics of x[n], the GD should not vary, as the statistics should be the same for different realizations, assuming WSS.

Task e): Bonus
Not done.

5.1 Matlab Code

Main File

```
close all
clear all
clc


load data.mat

mkdir Figures

% Task a) LMS and Gradient Descent for different mu

N = length(h);

mu_list = [0.0001, 0.001, 0.01, 1];
% mu_list = [];

OPTS = 0; %for standard LMS
alpha = 0;

v = zeros(N,size(X,2),size(X,1));
e = zeros(size(X,1), size(X,2));

for mu = mu_list

    for r = 1:size(X,1) %r...realization index
        x = X(r,:);
        d = D(r,:);

        [~,e(r,:),c] = lms_algorithm(x,d,N,mu,alpha,OPTS); %compute e for each realization

        v(:,:,r) = c - h; %compute v for each realization

    end


    %compute the expectation by averaging over different realizations
    e_expect = mean(e,1); %mean along the 1 dimension (rows get added)
    v_expect = mean(v,3); %mean along the third dimension (2D matrixes with row and col get added)


    v_plot = log(abs(v_expect)./abs(v_expect(:,1)));
    e_plot = log(e_expect.^2/e_expect(1).^2);


    n = 0:size(X,2)-1;

    figure
        plot(n,v_plot)
        title(['(A) LMS misalignment \mu =' num2str(mu)])
        legend('v_1[n]', 'v_2[n]', 'v_3[n]', 'v_4[n]', 'v_5[n]', 'v_6[n]')
        xlabel('n')
        grid on
```

```
        saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')


    figure
        plot(n,e_plot)
        title(['(A) LMS error \mu =' num2str(mu)])
        legend('e[n]')
        xlabel('n')
        grid on

        saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')

end %end for mu



%---------------------------------------
%Gradient Search


%get Rxx and p
h = [2, -0.5, 4, -2, -1, 2].';

A = 1;
theta = pi/2;
sigma_u = sqrt(4);

rxx_ref = @(k) A^2/2 * cos(theta*k) + sigma_u^2 * kroneckerDelta(sym(k));
Rxx_ref = double(toeplitz(rxx_ref(0:length(h)-1))); %double convertes the sym
    data to double precision numbers

p_ref = Rxx_ref * h; %c_MSE = h = Rxx^-1 * p


v = zeros(N,size(X,2),size(X,1));
e = zeros(size(X,1), size(X,2));

for mu = mu_list

    for r = 1:size(X,1) %r ... realization index
        x = X(r,:);
        d = D(r,:);

        [~,e(r,:),c] = gd_algorithm(x,d,N,mu,Rxx_ref,p_ref);

        v(:,:,r) = c - h;
    end

    %compute the expectation by averaging over different realizations
    e_expect = mean(e,1); %mean along the 1 dimension (rows get added)
    v_expect = mean(v,3); %mean along the third dimension (2D matrixes with row
        and col get added)


    v_plot = log(abs(v_expect)./abs(v_expect(:,1)));
    e_plot = log(e_expect.^2/e_expect(1).^2);

```

```

    n = 0:size(X,2)-1;

    figure
        plot(n,v_plot)
        title(['(A) GD misalignment \mu =' num2str(mu)])
        legend('v_1[n]', 'v_2[n]', 'v_3[n]', 'v_4[n]', 'v_5[n]', 'v_6[n]')
        xlabel('n')
        grid on

        saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')


    figure
        plot(n,e_plot)
        title(['(A) GD error \mu =' num2str(mu)])
        legend('e[n]')
        xlabel('n')
        grid on

        saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')

end %end for mu

% ----------------------------------------------------------------------------
% Task b)

mu = 0.0005;
 P_list = [10, 100, 1000];
% P_list = [];

 r = 1; %used realization
N = length(h);

 for P = P_list

    [rxx, mxx] = cross_correlation(X(r,:),X(r,:),P,N);
    [rdx, mdx] = cross_correlation(D(r,:),X(r,:),P,N);

    Rxx = toeplitz(rxx);
    p = rdx;


    for r = 1:size(X,1) %r...realization index
        x = X(r,:);
        d = D(r,:);

        %use estimated statistics
        [~,e(r,:),c] = gd_algorithm(x,d,N,mu,Rxx,p);
        v(:,:,r) = c - h;

        %use analytical statistics
        [~,e_an(r,:),c_an] = gd_algorithm(x,d,N,mu,Rxx_ref,p_ref);
        v_an(:,:,r) = c_an - h;
    end

   %compute the expectation by averaging over different realizations
    e_expect = mean(e,1); %mean along the 1 dimension (rows get added)
```

```
    v_expect = mean(v,3); %mean along the third dimension (2D matrixes with row
        and col get added)

    e_expect_an = mean(e_an,1); %mean along the 1 dimension (rows get added)
    v_expect_an = mean(v_an,3); %mean along the third dimension (2D matrixes
        with row and col get added)


    v_plot = log(abs(v_expect)./abs(v_expect(:,1)));
    e_plot = log(e_expect.^2/e_expect(1).^2);

    v_plot_an = log(abs(v_expect_an)./abs(v_expect_an(:,1)));
    e_plot_an = log(e_expect_an.^2/e_expect_an(1).^2);


    n = 0:size(X,2)-1;

    figure
        plot(n,v_plot)
        hold on
        plot(n,v_plot_an)
        title(['(B) GD misalignment P=' num2str(P)])
        legend('v_1[n]', 'v_2[n]', 'v_3[n]', 'v_4[n]', 'v_5[n]', 'v_6[n]', ...
                'v_1[n] analytical', 'v_2[n] analytical', 'v_3[n] analytical', '
                    v_4[n] analytical', 'v_5[n] analytical', 'v_6[n] analytical')
        xlabel('n')
        grid on

        saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')


    figure
        plot(n,e_plot)
        hold on
        plot(n,e_plot_an)
        title(['(B) GD error P=' num2str(P)])
        legend('e[n] estimate', 'e[n] analytical')
        xlabel('n')
        grid on

        saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')



end


%Seen from plots: with increasing window size, the gradient descend
%convergers better and better, the misalignment gets lower


% ------------------------------------------------------------------------
% Task c)


mu = 0.001;
alpha = 0;
OPTS = 0;
```

```

N_list = [10, 100];
% N_list = [];

for N = N_list

    %reset
    v_LMS = [];
    v_GD = [];

    %random initialization for c0
    c0 = rand(N,1);

    A = 1;
    theta = pi/2;
    sigma_u = sqrt(4);

    rxx_ref = @(k) A^2/2 * cos(theta*k) + sigma_u^2 * kroneckerDelta(sym(k));
    Rxx_ref = double(toeplitz(rxx_ref(0:N-1))); %double convertes the sym data
        to double precision numbers

    h_calc = [h; zeros(N-length(h),1)];

    p_ref = Rxx_ref * h_calc; %c_MSE = h = Rxx^-1 * p

    for r = 1:size(X,1) %r... realization index
        x = X(r,:);
        d = D(r,:);

        [~,e_LMS(r,:),c_LMS] = lms_algorithm(x,d,N,mu,alpha,OPTS,c0); %compute e
            for each realization
        [~,e_GD(r,:),c_GD] = gd_algorithm(x,d,N,mu,Rxx_ref,p_ref);

        v_LMS(:,:,r) = c_LMS - h_calc; %compute v for each realization
        v_GD(:,:,r) = c_GD - h_calc; %compute v for each realization

    end


    %compute the expectation by averaging over different realizations
    e_expect_LMS = mean(e_LMS,1); %mean along the 1 dimension (rows get added)
    v_expect_LMS = mean(v_LMS,3); %mean along the third dimension (2D matrixes
        with row and col get added)

    e_expect_GD = mean(e_GD,1); %mean along the 1 dimension (rows get added)
    v_expect_GD = mean(v_GD,3); %mean along the third dimension (2D matrixes
        with row and col get added)


    v_plot_LMS = log(abs(v_expect_LMS)./abs(v_expect_LMS(:,1)));
    e_plot_LMS = log(e_expect_LMS.^2/e_expect_LMS(1).^2);

    v_plot_GD = log(abs(v_expect_GD)./abs(v_expect_GD(:,1)));
    e_plot_GD = log(e_expect_GD.^2/e_expect_GD(1).^2);


    n = 0:size(X,2)-1;

```

```
    figure
        plot(n,v_plot_LMS)
        hold on
        plot(n,v_plot_GD)
        title(['(C) LMS and GD misalignment N=' num2str(N)])
        %legend not useful, too many coefficients
        xlabel('n')
        grid on

        saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')


    figure
        plot(n,e_plot_LMS)
        hold on
        plot(n,e_plot_GD)
        title(['(C) LMS and GD error N=' num2str(N)])
        legend('e[n] LMS', 'e[n] GD')
        xlabel('n')
        grid on

        saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')



end

%seen from plot: increasing order number has an influence (its overmodelled
%in this case)
%The LMS gets way worse convergence wise, the GD seems less impacted by
%this










%create a placeholder function to overwrite the saveas function
% function saveas(~, ~, ~)
%     disp('Figure not saved')
% end

a = 1;
```

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
```

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
```

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 cross_correlation()

```
function [rdx, mxx] = cross_correlation(x,y,P,N)


if N > P
    error('samples to average P must be greater or equal to filter coefficients
       N')
end

x_pad = [x(:); zeros(N-1,1)];
y = y(:); %make sure its a col vector

P_window = 1:P;

for k = 0:N-1
    rdx(k+1) = x_pad(P_window + k).' * y(P_window) / P;
end

mxx = 0:N-1;
rdx = rdx(:); %make sure its a col vector


end

% [rdx, mxx] = cross_correlation([1 2 3 4 5],[10 20 30 40 50],3, 2)
```

Function own_strrep()

```
function new_string = own_strrep(f)

    new_string = strrep(f.Title.String,' ', '_');
    new_string = strrep(new_string,'(', '');
    new_string = strrep(new_string,')', '');
    new_string = strrep(new_string,'\', '');
    new_string = strrep(new_string,'.', '_');

end
```