3 OCTAVE/MATLAB Problem 1.3

Least-Squares Tracking of a Time-Varying System

3.1 Task a)

For the first task a function called **ls_filter()** shall be implemented which computes the *least-squares* optimum filter coefficients $\mathbf{c}_{LS}[n] = \text{argmin}_{\mathbf{c}} J_{LS}(\mathbf{c}, n)$ according to the cost function

$$J_{LS}(\mathbf{c}, n) = \sum_{k=0}^{n} |e[k]|^2 \tag{1}$$

We know from the problem class:

$$c_{LS} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{d} \tag{2}$$

The Matlab files can be found in the **.zip** file. The code is also appended to this PDF sheet (Section 3.5)

3.2 Task b)

For the second task the 'unknown' system filter coefficients shall be plotted. These are:

$$\mathbf{h}[n] = \begin{bmatrix} -1 \\ 2 - 0.97^n \\ 0.3 \cdot cos(\theta n) \end{bmatrix} \tag{3}$$

where $\theta = \frac{3\pi}{1000}$ for $n \in [0,\ 999]$

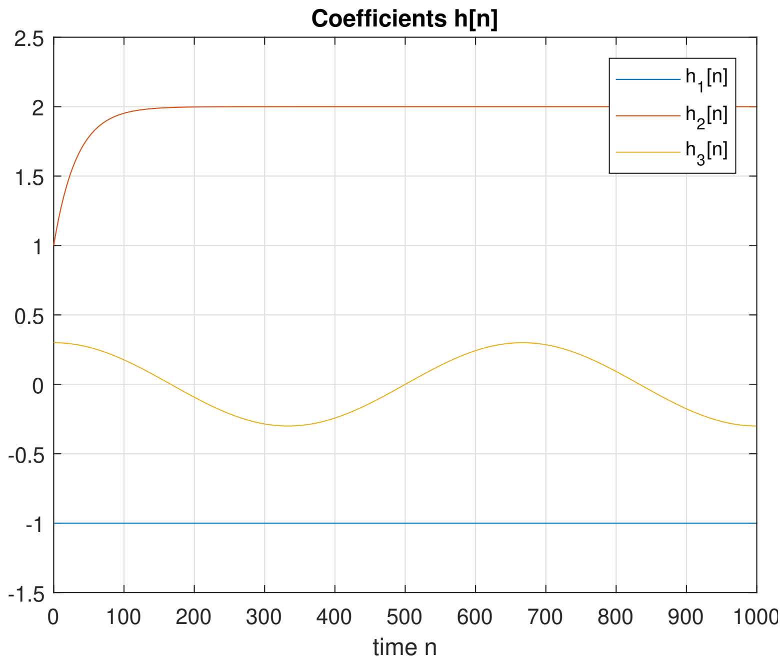


Figure 1: Coefficients for $\mathbf{h}$[n]

3.3 Task c)

In this task the coefficients were calculated using the *least-square* filter programmed in task a). The input signal $x[n]$ is a stationary white noise process with variance $\sigma_x^2 = 1$ (White noise processes always have a zero mean value). To compute the filter coefficients **c**[n] the error $e[n]$ has to be computed first. Therefore the values of d[n] are needed and can be computed from the 'unknown' system **h**[n] (that is why we know the coefficients of **h**[n]. In real life we would simply measure the data of d[n]). The noise w[n] = 0 this time and will be considered in task d).
The adaptive filter should have N = 3 coefficients (the filter order i.e. the amount of delay elements is N - 1 = 2).

$$\mathbf{c}[n] = \begin{bmatrix} c_1[n] \\ c_2[n] \\ c_3[n] \end{bmatrix} \tag{4}$$

The filter coefficients were computed by segmenting the input signal x[n] and calling the function **ls_filter()**. This should be done for segment lengths M = {20, 50}.

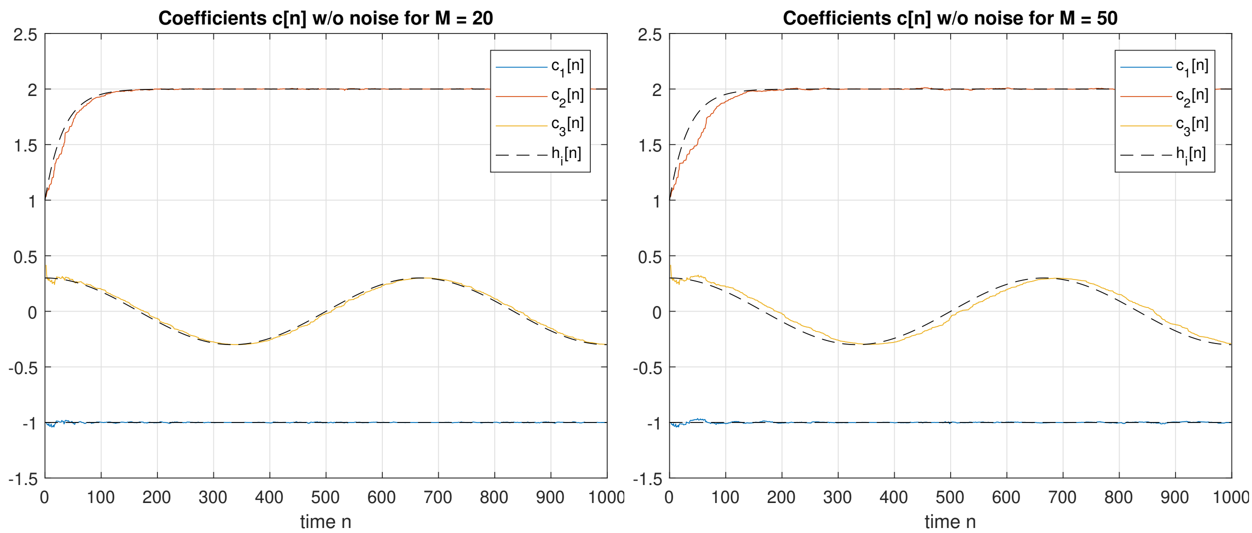


Figure 2: Coefficients for **c**[n] *without* noise and segment length M = 20 and M = 50

As we can see, the values for **h**[n] (black striped lines) correspond quite well with the coefficients which were calculated through the *least-square* sense. The first values of **c**[n] vary a lot from the correct solution due to assumption that $x[n] = 0$ for $n < 0$ and therefore the first $\mathbf{c}[n]$ get computed with only a few entries which are not equal to 0.

Between the two plots with different segment length M is only little difference. The initial deviation is longer due to the longer segment length and the plot gets a little bit more shifted (increases with segment length).

3.4 Task d)

Last but not least some noise w[n] with variance $\sigma_w = 0.02$ disturbs the 'measured' data d[n]. Again the coefficients were calculated and plotted.

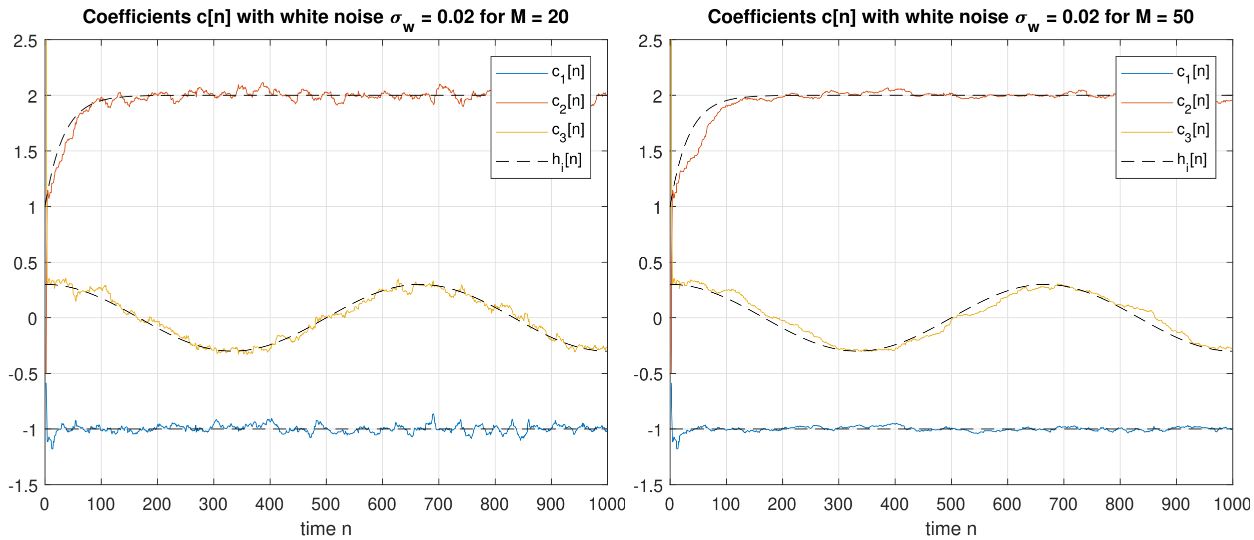


Figure 3: Coefficients for **c**[n] *with* noise and segment length M = 20 and M = 50

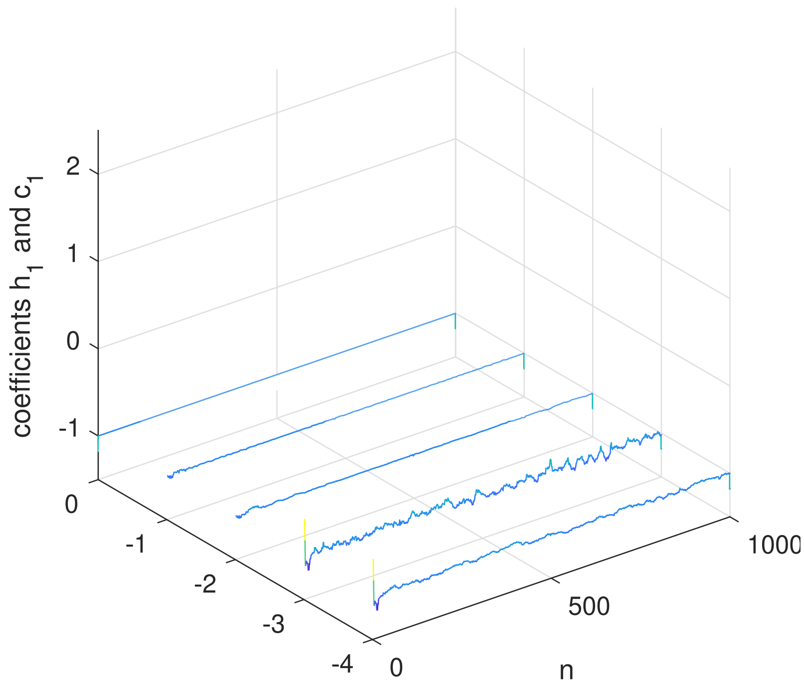
Between $M = 20$ and $M = 50$ is a lot of difference now. The additional noise $w[n]$ causes the coefficients to fluctuate. The plot with the lower segment length $M = 20$ fluctuates way more than the plot with segment length $M = 50$. Due to the property of white noise the effect it has should cancel out for infinite long observation. Since we only look at finite length of data we, some effects of the white noise still can be seen. To further decreases the noise the segment length can be increased.

WATERFALL PLOTS

Order of the lines, starting from top to bottom:

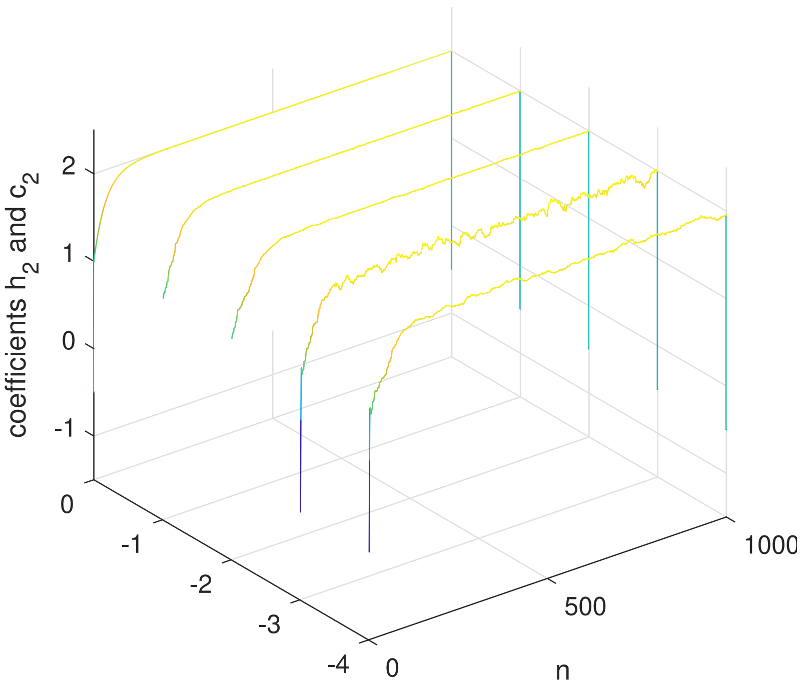
1. $h_i[n]$
2. $c_i[n]$ *without* measurement noise and for segment length $M = 20$
3. $c_i[n]$ *without* measurement noise and for segment length $M = 50$
4. $c_i[n]$ *with* measurement noise and for segment length $M = 20$
5. $c_i[n]$ *with* measurement noise and for segment length $M = 50$

where i denotes the i-th coefficient



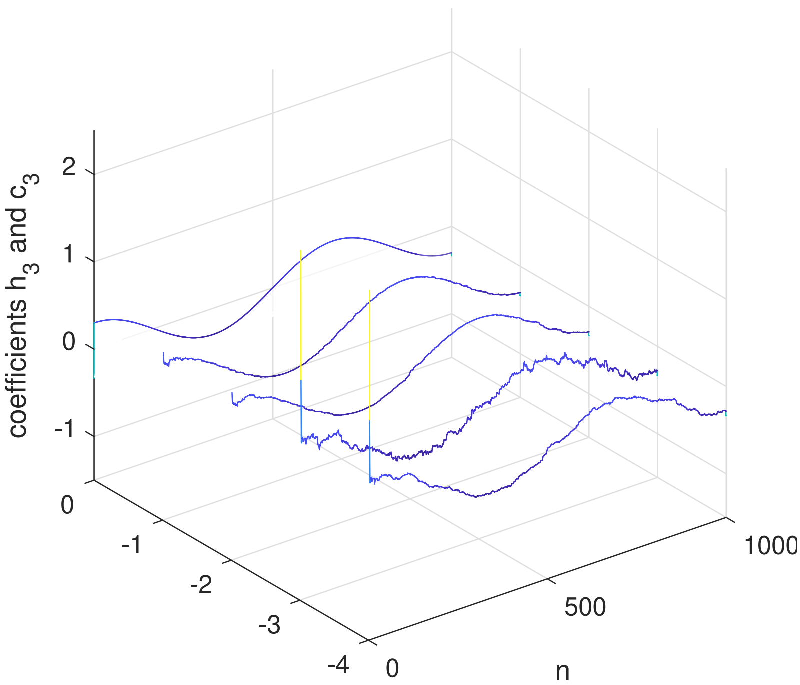
h | M = 20 & w = 0 | M = 50 & w = 0 | M = 20 & w ~= 0 | M = 50 & w ~= 0;

Figure 4: waterfall coefficients i = 1



h | M = 20 & w = 0 | M = 50 & w = 0 | M = 20 & w ~= 0 | M = 50 & w ~= 0;

Figure 5: waterfall coefficients i = 2



h | M = 20 & w = 0 | M = 50 & w = 0 | M = 20 & w ~= 0 | M = 50 & w ~= 0;

Figure 6: waterfall coefficients i = 3

3.5 Matlab Code

Main File

```matlab
close all
clear all
clc

%suppres: "Warning: Matrix is singular to working precision."
id = 'MATLAB:singularMatrix';
warning('off',id)

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

mkdir 'Figures' %create Figures folder



%------------------------------------------------------------------------
% a)
N = 2;
d = [-3; -5; 0]; %values from example 1.1
x = [-1; -1; 1];




k = N-1; %Pad with N - 1 zeroes for the values of x[-1], x[-2], ... x[-N+1]
x_pad = [zeros(k,1); x];
d_pad = [zeros(k,1); d];


c = ls_filter(x_pad, d_pad, N)

%"Probe"
d_hat = conv(x,c);
d_hat = d_hat(1:length(x)) %delete the values which assume that
%                            x[n] = 0 for n > length(x)



%------------------------------------------------------------------------
% b)
theta = 3*pi/1000;
n = 0:999;

h = [-1*ones(1,length(n)); 2-0.97.^n; 0.3*cos(theta*n)];
%h = h1[0], h1[1], ..., h1[n];
%    h2[0], h2[1], ..., h2[n];
%    h3[0], h3[1], ..., h3[n];


figure
    plot(n,h)
    legend('h_1[n]','h_2[n]','h_3[n]')
    grid on
```

```matlab
    ylim([-1.5  2.5])
    title('Coefficients h[n]')
    xlabel('time n')

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

%------------------------------------------------------------------------
% c) and d)
N = 3; %3 filter coefficients in h and c

x = randn(1,length(n)).'; %x[n] = 0 for n < 0 (or 1 in matlab)

d = vector_conv(x, h);
% d_test = vector_conv2(x, h)
% e_test = d - d_test
counter = 1;
for jj = 1:2

    if jj == 1
        w = 0;
    else
        %create white gaussian noise and change variance
        w = transpose(randn(1,length(n)))./(1/sqrt(0.02));
    end

    d = d + w;% add noise after filter h


    for M = [20, 50]
        x_pad = [zeros(M-1,1); x]; %pad with M-1 zeros; x[n] = 0 for n < 0;
        d_pad = [zeros(M-1,1); d]; %and pad d too for the newly created values of x[n]

        c = zeros(N,length(n));
        for ii = n %ii is counts through the time n
            c(:,ii+1) = ls_filter(x_pad(ii+1:M+ii), d_pad(ii+1:M+ii), N);
        end


        if w == 0
            text = ['Coefficients c[n] w/o noise for M = ' num2str(M)];
            text_saveas = ['Coefficients_c_without_noise_M=' num2str(M)];
        else
            text = ['Coefficients c[n] with white noise \sigma_w = ' num2str(round(var(w),2)) ' for M = ' num2str(M)];
            text_saveas = ['Coefficients_c_with_noise_M=' num2str(M)];
        end

        figure
            plot(n,c)
            hold on
            plot(n,h,'--k')
            legend('c_1[n]','c_2[n]','c_3[n]','h_i[n]')
            grid on
            title(text)
            xlabel('time n')
            ylim([-1.5, 2.5]) %due to some singularities, the first values
%                                  of c can get quite big -> ruins the plot ->
```

```
%                               limit it
            saveas(gcf,['Figures/' text_saveas] ,'epsc') %epsc to save the eps in colour

        c_plot(:,:,counter) = c;
        counter = counter + 1;
    end  %for M

end %for jj

kk = 1;
X = [1; 1; 1; 1; 1]* n;
Y = [0; -1; -2; -3; -4]*ones(1,length(n));

for kk = 1:3
Z = [h(kk,:); c_plot(kk,:,1); c_plot(kk,:,2); c_plot(kk,:,3); c_plot(kk,:,4)];
%c_plot(kk,:,1) has the coefficients for M = 20, w = 0;
%c_plot(kk,:,2) has the coefficients for M = 50, w = 0;
%c_plot(kk,:,3) has the coefficients for M = 20, w ~= 0;
%c_plot(kk,:,4) has the coefficients for M = 50, w ~= 0;

 text_saveas = ['waterfall_coefficients_' num2str(kk)];

figure
    waterfall(X,Y,Z)
    zlim([-1.5, 2.5])
    xlabel('n')
    zlabel(['coefficients h_' num2str(kk) ' and c_' num2str(kk)])
    ylabel('h | M = 20 & w = 0 |  M = 50 & w = 0 |  M = 20 & w ~= 0 |  M = 50 & w  ~= 0;')
    saveas(gcf,['Figures/' text_saveas] ,'epsc') %epsc to save the eps in colour
end


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


%seen from plots:
%w[n] = 0:
%theres only little difference between M = 20 and M = 50 and
%the values of c correspond quite well to the values of h,
%altough the plot is a bit shifted(increses with segment
%length M). The first values of c are also not quite the same
%as h, due to the assumption that x[n] = 0, for n < 0 and the
%first c[n] gets computed with only one entry which is not 0;
%
%w[n] ~= 0:
%between M = 20 and M = 50 is a lot of difference now. Probably
%due to the higher amount of samples, the noise cancels out
%(and would fully cancel for M -> infty(because white noise),
%but then the adaptiveness of system would get lost -> c[n] would
%get constant if every x[n] is taken into account)
```

Function ls_filter()

```
function c = ls_filter(x, d, N)
%computes the filter coefficients c for one time instance n, where
%corresponds to the last entry of x

% x is the input signal saved as col vector
% d is the reference signal saved as col vector
% N is the order of the filter i.e. the amount of coefficients in c

% Matrix X to compute the coefficients at time instance n
% X = [x[n-M+N],   x[n-M+N-1],   ..., x[n-M];
%      x[n-M+N+1], x[n-M+N],     ..., x[n-M+1];
%      ...                       ...,
%      x[n],       x[n-1]        ..., x[n-N+1] ];

% Make sure x and d are col vectors
% x = x(:);
% d = d(:);

if isrow(x) || isrow(d)
    error('vector x or vector d is not a column vector')
end

M = length(x); %segment length

X = zeros(M-N+1,N); %create placeholder for entries of X

for ii = 0:N-1 %for order N coefficients of c we need N cols
    X(:,ii+1) = x( (end-M+N)-ii:end-ii ); %end corresponds to current time n
end                                        %to include M-N+1 we have to substract
    -M+N

c = (X.' * X)^-1 * X.' * d(end-M+N:end); %pseudo inverse; multiply with
    segmented d

end
```

Function vector_conv()

```
function y = vector_conv(x, h)
%calcultes the concolution sum defined in Adaptive System UE
%
%x has to be a column vector in form of
% x = x[0];
%     x[1];
%      ...;
%     x[n-1]
%
%where n is the time variable


%h has to be matrix in the form of
% h = h1[0], h1[1], ..., h1[n-1];
%     h2[0], h2[1], ..., h2[n-1];
%     h3[0], h3[1], ..., h3[n-1]


x = x(:); %make sure that x is a col vector


N = size(h,1);


t = 1;
n = 0:length(x)-1;
x_zero_pad = [zeros(N-1,1); x]; %puts zeros for time x[-1], x[-2], ... x[-N+1]

y = zeros(length(n),1);
for n_shift = n + N
    x_tap_input = x_zero_pad(n_shift:-1:n_shift-N+1);
    y(t) = h(:,t)' * x_tap_input; % ' is hermitian transposed
    t = t+1;
end


end
```

Function vector_conv2()

```
 function y = vector_conv2(x, h)
% computes the convolution of x and the coefficients of h at time
% instance n for every n

 y = zeros(length(x),1);

 for n = 1:length(x)
        temp = conv(x,h(:,n));
     y(n) = temp(n);
 end

 end
```