4 OCTAVE/MATLAB Problem 1.4

Weighted-Least-Squares Tracking of a Time-Varying System

4.1 Task a)

For the first task the optimal filter coeffiecents shall be derived so that $\mathbf{c}_{wLS}[n] = \text{argmin}_{\mathbf{c}} J_{wLS}(\mathbf{c}, n)$ according to the wegihted least squares cost function

$$J_{wLS}(\mathbf{c}, n) = \sum_{k=n-M+1}^{n} g[n-k] \cdot |e[k]|^2 \tag{1}$$

The derivation was done by hand and can be found on the next page.

For reference the 'unknown' system filter coefficients were plotted. These are:

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

Note: θ was changed from Problem 1.3 to Problem 1.4: $\theta = \frac{3\pi}{100}$

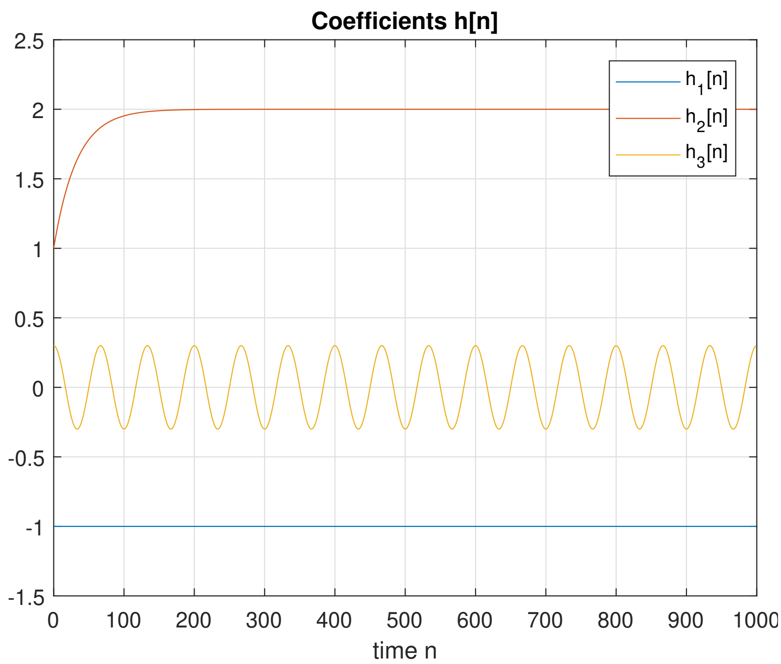

Figure 1: Coefficients for **h**[n]

Problem 1.4

$$J_{WLS}(\underline{c}, n) = \sum_{k=n-M+1}^{n} g[n-k] \cdot |e[k]|^2 \overset{e \in \mathbb{R}}{=} \sum_{k=n-M+1}^{n} e[k] \cdot g[n-k] \cdot e[k]$$

„local variable" bc $g[n-k]$

$$= \underline{e}^T[n] \cdot \underline{G} \cdot \underline{e}[n] = [e[n], e[n-1], \ldots, e[n-M+1]] \cdot \begin{bmatrix} g[M-1] & 0 & \cdots & 0 \\ 0 & g[M-2] & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & g[0] \end{bmatrix} \cdot \begin{bmatrix} e[n] \\ e[n-1] \\ \vdots \\ e[n-M+1] \end{bmatrix}$$

P. 1.1

from problem class: $\underline{e}[n] = \underline{d}[n] - \underline{X} \cdot \underline{c}[n]$

$$\underline{d}[n] = \begin{bmatrix} d[n] \\ \vdots \\ d[n-M+1] \end{bmatrix}$$

$$\underline{X} = \begin{bmatrix} x[n-M+1] & \cdots & x[n-M+1-N] \\ \vdots & & \vdots \\ x[n] & \cdots & x[n-N] \end{bmatrix}$$

$$J_{WLS}(\underline{c}, n) = \underline{e}^T[n] \cdot \underline{G} \cdot \underline{e}[n] = (\underline{d} - \underline{X} \cdot \underline{c})^T \cdot \underline{G} \cdot (\underline{d} - \underline{X} \cdot \underline{c})$$

$$= (\underline{d}^T - \underline{c}^T \cdot \underline{X}^T) \cdot \underline{G} \cdot (\underline{d} - \underline{X} \cdot \underline{c}) =$$

$$= (\underline{d}^T \underline{G} - \underline{c}^T \underline{X}^T \cdot \underline{G}) \cdot (\underline{d} - \underline{X} \cdot \underline{c}) =$$

$$= (\underline{d}^T \underline{G} \underline{d} - \underline{c}^T \cdot \underline{X}^T \underline{G} \cdot \underline{d} - \underbrace{(\underline{d}^T \cdot \underline{G} \cdot \underline{X} \cdot \underline{c})}_{\text{Scalar}}{}^T + \underline{c}^T \cdot \underline{X}^T \cdot \underline{G} \cdot \underline{X} \cdot \underline{c})$$

$$= \underline{d}^T \cdot \underline{G} \cdot \underline{d} - \underline{c}^T \underline{X}^T \cdot \underline{G} \cdot \underline{d} - \underline{c}^T \cdot \underline{X}^T \cdot \underline{G}^T \cdot \underline{d} + \underline{c}^T \cdot \underline{X}^T \cdot \underline{G} \cdot \underline{X} \cdot \underline{c} \quad // \ \underline{G}^T = \underline{G}$$

vector is in front

$$= \underbrace{\underline{d}^T \cdot \underline{G} \cdot \underline{d}}_{\text{constant}} - \underline{c}^T \cdot \underline{X}^T \cdot \underline{G} \cdot \underline{d} \cdot 2 + \underline{c}^T \cdot \underline{X}^T \cdot \underline{G} \cdot \underline{X} \cdot \underline{c}$$

$$\underline{\nabla}_c(J_{WLS}) = 0 - \underline{X}^T \underline{G} \cdot \underline{d} \cdot \cancel{2} + \cancel{2} \, \underline{X}^T \underline{G} \underline{X} \, \underline{c} \overset{!}{=} 0$$

$$\underline{X}^T \cdot \underline{G} \, \underline{X} \cdot \underline{c} = \underline{X}^T \underline{G} \cdot \underline{d}$$

$$\underline{c} = (\underline{X}^T \cdot \underline{G} \cdot \underline{X})^{-1} \cdot \underline{X}^T \cdot \underline{G} \cdot \underline{d}$$

$$\boxed{\underline{c}_{WLS} = (\underline{X}^T \cdot \underline{G} \cdot \underline{X})^{-1} \cdot \underline{X}^T \cdot \underline{G} \cdot \underline{d}}$$

4.2 Task b)

In the second task the already programmed `ls_filter()` was changed to include the derived formula from Task a).The new function `ls_filter_weighted(x, d, N, λ)` expects 4 input parameters. The new parameter λ is used for the calculation of the weights:

$$g[m] = \lambda^{-m} \tag{3}$$

The Matlab files can be found in the **.zip** file. The code is also appended to this PDF (Section 4.4)

4.3 Task c)

Different values for λ were tested and plotted for $\theta = \frac{3\pi}{100}$ and $M = 50$. For reference the coefficients of h were drawn as striped black lines.

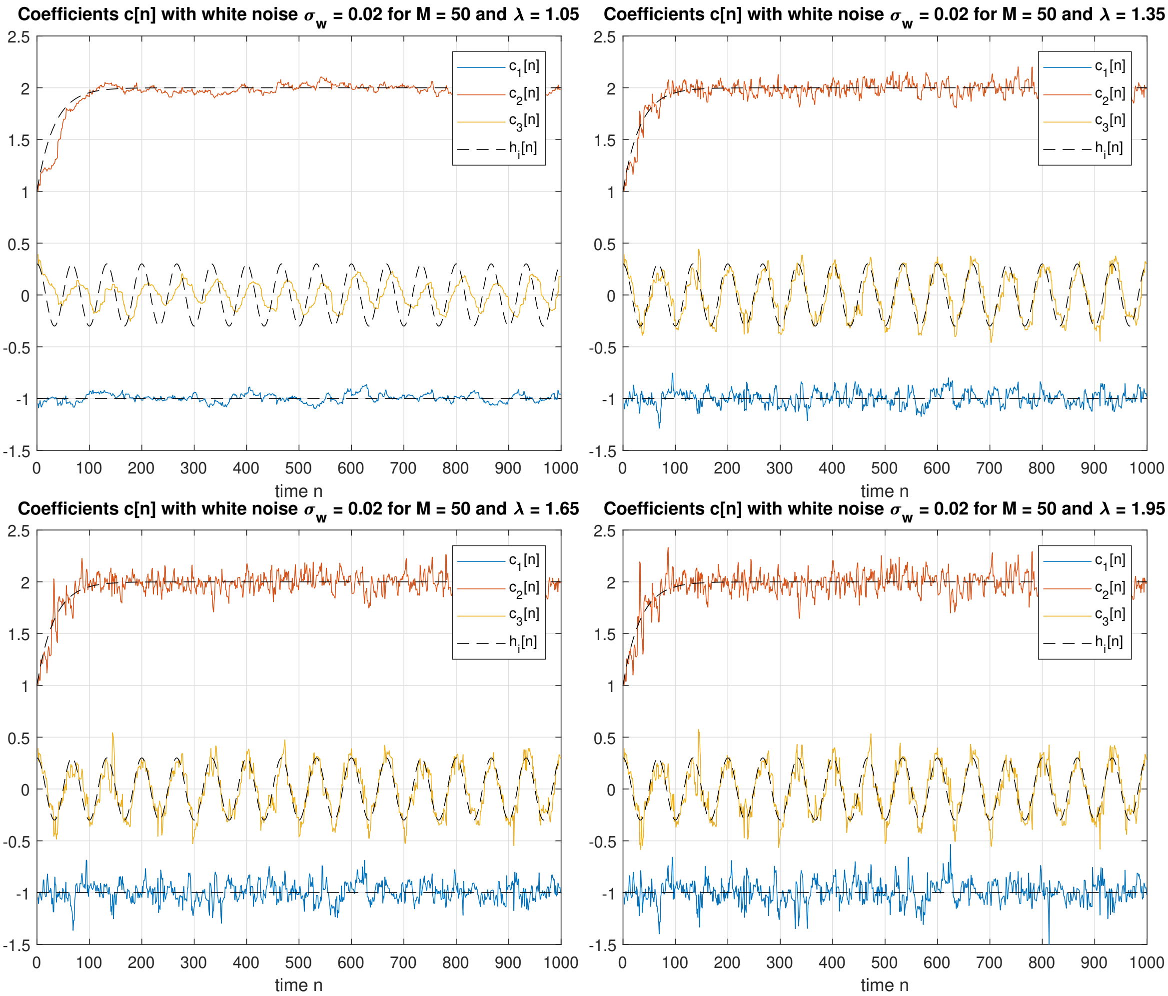

Figure 2: Coefficients for **c**[n] with noise, segment length M = 50 and varying λ

The bigger the λ the greater the added noise. This is because the *weighted-least-squares* emphasises more on the newly gathered information for $e[n]$ which is corrupted by noise $w[n]$. But on the other hand the system is faster now, there is almost no delay between the real time varying coefficients of $\mathbf{h}[n]$ and the calculated $\mathbf{c}[n]$, which was caused by the segment length M.
A reasonable choice for λ seems to be $\lambda = 1.35$ because the delay between $\mathbf{h}[n]$ and $\mathbf{c}[n]$ is marginal and the noise doesnt seem too bad.

4.4 Matlab Code

Main File

```
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'

%------------------------------------------------------------------------
% calculate h
theta = 3*pi/100; %change theta from problem 1.3
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
    ylim([-1.5 2.5])
    title('Coefficients h[n]')
    xlabel('time n')

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

%------------------------------------------------------------------------
% weighted system identification
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);

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

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


for lambda = 1.05:0.3:2

    for M = [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_weighted(x_pad(ii+1:M+ii), d_pad(ii+1:M+ii), N, lambda);
        end


        text = ['Coefficients c[n] with white noise \sigma_w = ' num2str(round(var(w),2)) ' for M = ' num2str(M) ' and \lambda = ' num2str(lambda)];
        text_saveas = ['Coefficients_c_with_noise_M=' num2str(M) '_and_lambda=' strrep(num2str(lambda),'.','_')];


        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


    end  %for M

end %for lambda

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

Function ls_filter_weighted()

```matlab
function c = ls_filter_weighted(x, d, N, lambda)
%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


%compute the weighting matrix
% lambda = 1.5;
m = 0:-1:-(M-N);
g = lambda.^m;
G = diag(flip(g)); %flip because of form of matrix G (from g[n - k])

%compute coefficients with weighted input

c = (X.' * G * X)^-1 * X.' * G * d(end-M+N:end);

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
```