Signal Processing and Speech Communication Lab.
Graz University of Technology

AS

Adaptive Systems—Homework Assignment 2 v1.0

Name(s)	Matr.No(s).
Hannes Reindl	01532129
Philip Samer	01634718

Your solutions to the problems (your calculations, the answers to each task as well as the OCTAVE/MATLAB plots) have to be uploaded to the TeachCenter as a single `*.pdf` file, no later than **2019/1/20**. Use **this page** as the title page and fill in your **name(s) and matriculation number(s)**. Submitting your homework as a LaTeX document can earn you **up to 3 bonus points!**
All scripts needed for your OCTAVE/MATLAB solutions (all `*.m` files) have to be uploaded to the TeachCenter as a single `*.zip` archive, no later than **2019/1/20**.
All filenames consist of the assignment number and your matriculation number(s) such as `Assignment2_MatrNo1_MatrNo2.*`, for example,

Problem solutions:	`Assignment2_01312345_01312346.pdf`
OCTAVE/MATLAB files:	`Assignment2_01312345_01312346.zip`

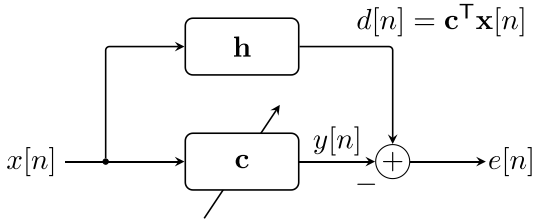
Please make sure that your approaches, procedures, and results are clearly presented. Justify your answers! A single upload of the files per group is sufficient.

Deterministic-Gradient Algorithms (Steepest-Descent Algorithms)
Algorithms belonging to the class of *deterministic* gradient overcome the problem of inverting the autocorrelation matrix $\mathbf{R}_{xx}$ needed to compute the optimum filter coefficients in the MSE-sense, i.e., the Wiener-Hopf solution. The gradient search algorithm in turn iteratively updates the coefficients according to the equation

$$\mathbf{c}[n] = \mathbf{c}[n-1] - \tilde{\mu}\nabla_{\mathbf{c}} J(\mathbf{c})\big|_{\mathbf{c}=\mathbf{c}[n-1]} \tag{1}$$

by tacking a step in the direction of the negative gradient. As the gradient of the MSE cost function $J(\mathbf{c}) = J_{\text{MSE}}(\mathbf{c})$ is used, the behaviour of these algorithm(s) is deterministic.

Analytical Problem 2.1 (13 Points)—Deterministic-Gradient Algorithm



A variation of the standard gradient search from (1) is the *coefficient–leakage* gradient search algorithm. This algorithm uses the coefficient update rule

$$\mathbf{c}[n] = (1 - \mu\alpha)\mathbf{c}[n-1] + \mu(\mathbf{p} - \mathbf{R}_{xx}\mathbf{c}[n-1]) \tag{2}$$

where $\alpha \in \mathbb{R}$ is the leakage coefficient ($0 \leq \alpha < 1$) and $\mu \in \mathbb{R}$ the step-size. Assume that $x[n] \in \mathbb{R}$ and $d[n] \in \mathbb{R}$. Use an arbitrary filter order N, i.e., $\mathbf{c}[n] \in \mathbb{R}^N$

(a) (2 points) Show how to derive the update equation for the coefficient–leakage gradient search algorithm from the modified cost function

$$J(\mathbf{c}[n]) = \mathsf{E}\big[e^2[n]\big] + \alpha||\mathbf{c}[n]||^2$$

(b) (2 points) Assume that the algorithm converged. Where does it converge to?

(c) (2 points) Transform the update equation for the filter coefficients $\mathbf{c}[n]$ to adapt the misalignment vector defined as $\mathbf{v}[n] = \mathbf{c}[n] - \mathbf{c}_\infty$.

(d) (2 points) Apply a unitary coordinate transform $\mathcal{T}(\cdot)$ such that the transformed components of the misalignment vector $\tilde{\mathbf{v}}[n] = \mathcal{T}(\mathbf{v}[n])$ are decoupled.

(e) (2 points) Use the decoupled update equation of the misalignment vector and express it as a function of time n and the initial transformed misalignment vector $\tilde{\mathbf{v}}[0]$.

(f) (3 points) Find an expression of the time constants for each of the decoupled coefficients. What is the influence of the leakage parameter α?

OCTAVE/MATLAB Problem 2.2 (6 Points)—Signal Statistics

The optimal solution to a system identification problem in the mean-square error sense can be found in closed form by using the Wiener-Hopf or iteratively by applying the *gradient descent* algorithm, both of which rely on the knowledge of the signal statistics. Assuming stationarity, one needs to know the autocorrelation of the filter input signal $x[n] \in \mathbb{R}$

$$r_{xx}[k] = \mathsf{E}\big[x[n+k]x[n]\big]$$

as well as the cross-correlation between $x[n]$ and the reference signal $d[n] \in \mathbb{R}$ defined as

$$p[k] = \mathsf{E}\big[d[n]x[n-k]\big].$$

Computing the statistical expectation $\mathsf{E}\big[\cdot\big]$ (or *ensemble average*) requires averaging over different realizations of a random variable or process, which needs to be approximated in an actual application scenario. The *mean ergodic* (m.e.) and *correlation ergodic* (c.e.) theorem state that the ensemble averages can be exchanged with time averages under certain conditions, which are usually fulfilled by wide sense stationary (WSS) processes. These time-averages for mean and correlation are defined in terms of the sample averages as

$$\mu_x = \mathsf{E}\big[x[n]\big] \quad \overset{\text{m.e.}}{\longleftrightarrow} \quad \hat{\mu}[P] = \frac{1}{P}\sum_{n=0}^{P-1} x[n]$$

$$r_{xy}[k] = \mathsf{E}\big[x[n+k]y[n]\big] \quad \overset{\text{c.e.}}{\longleftrightarrow} \quad \hat{r}_{xy}[k,P] = \frac{1}{P}\sum_{n=0}^{P-1} x[n+k]y[n], \qquad 0 \le k \le P-1$$

and are computed over a signal window of size P. In connection to system identification it will be sufficient to evaluate time lags $0 \le k \le N-1$ with $N \le P$ (implying the need for at least as many samples to average as filter coefficients N).

(a) (2 points) Show analytically that the expected values of both sample averages are equal to the quantities that are approximated, i.e., show that $\mathsf{E}\big[\hat{\mu}[P]\big] = \mu_x$ and $\mathsf{E}\big[\hat{r}_{xy}[k,P]\big] = r_{xy}[k]$ assuming $x[n]$ and $y[n]$ are jointly WSS.

(b) (2 points) Write an Octave/Matlab script that uses the sample correlation $\hat{r}_{xy}[k,P]$ to estimate the cross- and autocorrelation functions $r_{dx}[k] = p[k]$ and $r_{xx}[k]$ for the signals $x[n]$ and $d[n]$ provided in the file `data.mat`. Refer to Appendix 1 for a detailed description of the file content.
Hint: Be careful that you are not allowed to mix different realizations of $x[n]$ and $d[n]$ when computing the sample correlations.

(c) (2 points) Compute the sample estimates for different window sizes P for different signal realizations and compare them to the analytical values for $r_{xx}[k]$ and $p[k]$ given in Appendix 1. Plot and discuss the results. How does the accuracy of the estimates scale with the window size P?
Hint: Compute the statistics separately for some realizations, you do not need to use all.

Stochastic-Gradient Algorithms
In contrast to the gradient search algorithm where a *deterministic* gradient is used to iteratively compute the Wiener-Hopf solution, the LMS algorithm uses a *stochastic* approximation of the gradient and is thus part of the class of *stochastic* gradient algorithms. As shown in the problem class, the update equation for the LMS-algorithm assuming real signals

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

can be derived from the deterministic gradient algorithm by using an instantaneous estimate of the gradient found to be the input signal $\mathbf{x}[n]$ scaled with the *a-priori* error $e[n]$. The result is a random update of the filter coefficients form one iteration cycle to the next, justifying the use of the term *stochastic* gradient.

Analytical Problem 2.3 (6 Points)—LMS Algorithm

Two often used variations of the standard LMS algorithm given in (3) either employ a leakage factor applied to the previous coefficients or a signal power dependent step size. The corresponding update equations assuming real signals are then

$$\text{coefficient-leakage LMS}: \quad \mathbf{c}[n] = (1-\mu\alpha)\mathbf{c}[n-1] + \mu e[n]\mathbf{x}[n] \qquad 0 \leq \alpha < 1 \tag{4}$$

$$\text{normalized LMS}: \quad \mathbf{c}[n] = \mathbf{c}[n-1] + \frac{\mu}{||\mathbf{x}[n]||^2 + \alpha} e[n]\mathbf{x}[n]. \qquad \alpha > 0 \tag{5}$$

(a) (4 points) The convergence properties of the LMS algorithm can only be examined *on average*. Using the assumption of convergence on average, i.e., assuming that

$$\mathsf{E}\big[\mathbf{c}[n]\big] = \mathsf{E}\big[\mathbf{c}[n-1]\big] = \mathsf{E}\big[\mathbf{c}_\infty\big]$$

compute the expected converged solution of the LMS (3) and the coefficient–leakage LMS (4) for a noisy system identification scenario assuming real coefficients. For your derivations assume a white Gaussian input signal $x[n] \in \mathbb{R}$ with non-zero variance σ_x^2 and uncorrelated white Gaussian noise $w[n]$ with non-zero variance σ_w^2 (e.g. see Fig. 1 below). Compare the results you obtained with a general system response $\mathbf{h} \in \mathbb{R}^N$. What do you observe?
Hint: Compute $\mathsf{E}\big[\mathbf{c}[n]\big]$ to find the converged coefficients in terms of a general, unknown system response $\mathbf{h}$.

(b) (2 points) Explain the effect of the normalization term in the normalized LMS algorithm (5). What is the effect of α?
Hint: Think about the step-size μ and its relation to convergence.

OCTAVE/MATLAB Problem 2.4 (7 Points)—Gradient Algorithms

Implement the standard *deterministic* and *stochastic* gradient algorithms examined in theory in the previous tasks, and use the provided script `gradient_test.m` to test the performance. The script and some needed data files are contained in `gradient_test.zip`.

(a) (4 points) Write Matlab a function `lms_algorithm()` that implements the standard LMS algorithm (3) and the normalized LMS (5) according to the following specification:

```
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)
```

(b) (2 points) Write Matlab a function `gd_algorithm()` implementing the standard gradient descent algorithm (1) according to the specification below. Be careful, that in this form the signal statistics are estimated beforehand and not adapted/changed during execution.

```
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)
```

(c) (1 points) The first 3 plots are only for validation of your implementation. Briefly explain and discuss what you observe in the fourth plot.

OCTAVE/MATLAB Problem 2.5 (10 Points)—Performance Analysis

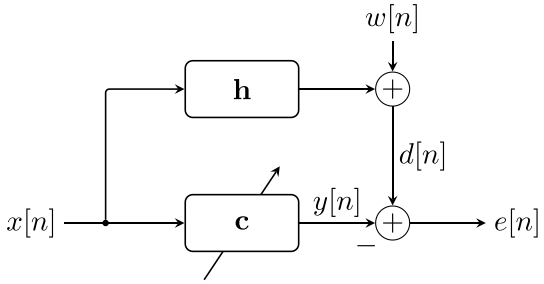


Figure 1: System identification in a noisy environment.

This problem will quantify and compare the performance of the *deterministic* and *stochastic* gradient algorithms implemented in the previous task. The input and reference data for the system is provided in the file `data.mat`, providing you with multiple realizations of the needed signals $x[n]$ and $d[n]$ and with the unknown filter coefficients $\mathbf{h}$. Refer to Appendix 1 for a detailed description of the content of `data.mat` and the general system setup.
For the noisy system identification problem shown in Fig. 1 we want to investigate the convergence behavior of both the deterministic and the stochastic gradient algorithms implemented in the previous task. For investigating the performance we will use the misalignment vector $\mathbf{v}[n]$ defined as

$$\mathbf{v}[n] = \mathbf{c}[n] - \mathbf{h}.$$

For each subtask, visualize the behavior of each component k of the misalignment vector over time n as

$$\ln \frac{|\mathsf{E}\big[v_k[n]\big]|}{|\mathsf{E}\big[v_k[0]\big]|}.$$

Also plot the logarithm of the MSE, i.e., plot

$$\ln \frac{\mathsf{E}\big[e^2[n]\big]}{\mathsf{E}\big[e^2[0]\big]}.$$

To compute the expectations $\mathsf{E}\big[\cdot\big]$ you can average over the results of different realizations using

$$\mathsf{E}\big[\mathbf{v}[n]\big] \approx \frac{1}{R}\sum_{r=1}^{R}\mathbf{v}_r[n] \qquad \text{and} \qquad \mathsf{E}\big[e^2[n]\big] \approx \frac{1}{R}\sum_{r=1}^{R}e_r^2[n]$$

where $\mathbf{v}_r[n]$ and $e_r[n]$ are the results obtained for a realization r of the input signal and reference signal.
Perform the following tasks using your implementations of the stochastic and deterministic gradient algorithms from the previous task. Until stated otherwise assume that you know the true filter order, i.e., use $N = \dim(\mathbf{c}[n]) = \dim(\mathbf{h})$.

(a) (3 points) Investigate the effect of the step-size parameter μ when using the standard LMS algorithm (3) and the standard gradient descent algorithm (1) with the analytical values for $\mathbf{R}_{xx}$ and $\mathbf{p}$. Use $\mu = \{0.0001, 0.001, 0.01, 1\}$. Are the algorithms stable in all these cases?

(b) (3 points) To investigate the effect of the number of samples used to compute the sample correlations, now compare the behavior of gradient search algorithm (1) using sample estimates

with values using signal windows of size $P = \{10, 100, 1000\}$ and compare it to the results using the analytical values for the statistics. Use a stepsize $\mu = 0.0005$. What do you observe?

(c) (2 points) Now try to drastically increase the adaptive filter order, using $N = \{10, 100\}$ coefficients and the standard LMS algorithm (3) and the standard gradient descent algorithm using the analytical solutions for $\mathbf{R}_{xx}$ and $\mathbf{p}$ with a step size $\mu = 0.001$. What do you observe in terms the resulting error? Use a random initialization for $\mathbf{c}[0]$.

(d) (2 points) How does the MSE of the different algorithms behave *after* convergence in all previous tasks? How do the different algorithms behave for different signal realizations?

(e) Bonus: (4 points) Modify your functions `gd_algorithm()` and `lms_algorithm()` such that they implement the coefficient-leakage variants of the corresponding algorithms, i.e., (2) and (4) and show where the algorithms converge to. Compare the results to your analytical solutions.

Analytical Problem 2.6 (5 Points)—Bonus: Wiener-Hopf Solution

In most scenarios it is a strong limitation when only assuming real coefficients. The mean-squared error cost function for general signals (real or complex) is given as

$$J_{\text{MSE}}(\mathbf{c}) = \mathsf{E}\big[|e[n]|^2\big] = \mathsf{E}\big[e[n]e^*[n]\big]$$

with error $e[n] = y[n] - d[n]$, filter output $y[n] = \mathbf{c}^{\mathsf{H}}\mathbf{x}[n]$ and noisy reference signal $d[n] = \mathbf{h}^{\mathsf{H}}\mathbf{x}[n] + w[n]$, corresponding to a noisy system identification scenario. Assume that the filter input signal $x[n]$ and noise $w[n]$ are statistically independent.

(a) (4 points) Derive the Wiener-Hopf solution for complex filter coefficients $\mathbf{c} \in \mathbb{C}^N$ and a complex filter input signal $x[n] \in \mathbb{C}$. Use the definition of the partial derivative w.r.t. a complex coefficient $c_k = a_k + jb_k$ as

$$\frac{\partial}{\partial c_k} = \frac{\partial}{\partial a_k} + j\frac{\partial}{\partial b_k}$$

and the corresponding gradient w.r.t. to the filter coefficient vector $\mathbf{c}$ defined as

$$\nabla_{\mathbf{c}} = \begin{bmatrix} \frac{\partial}{\partial c_1} \\ \vdots \\ \frac{\partial}{\partial c_k} \\ \vdots \\ \frac{\partial}{\partial c_N} \end{bmatrix}$$

and carefully specify the corresponding vector and matrices. Clearly indicate where you use transpose $(\cdot)^{\mathsf{T}}$ and where a conjugate transpose $(\cdot)^{\mathsf{H}}$.

(b) (1 points) Write down the resulting expression for the resulting minimum MSE $J_{\text{min}} = J_{\text{MSE}}(\mathbf{c}_{\text{MSE}})$. Simplify as much as possible.

Appendix 1—Data Description (`data.mat`)

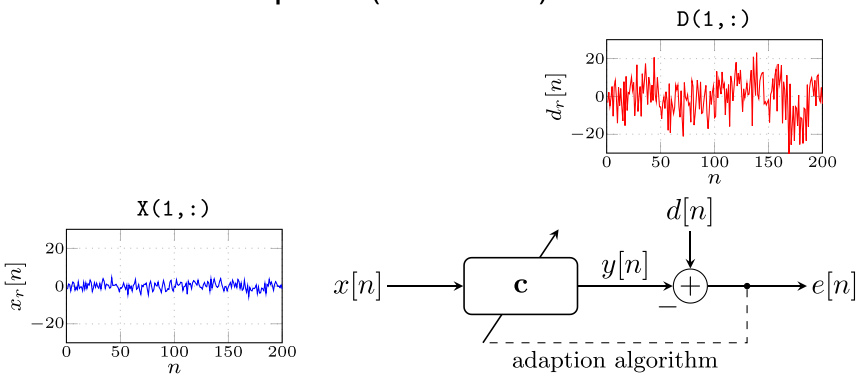


Figure 2: General system identification

The provided data used throughout this assignment is provided in the file `data.mat`. The file contains multiple realizations of a filter input signal $x[n]$, a reference signal $d[n]$, stored in the data matrices `X` and `D` respectively, and the true filter coefficients $\mathtt{h} \in \mathbb{R}^N$ with $N = 6$. It further contains the standard deviations $\mathtt{sigma\_u} = \sigma_u$ of $u[n]$ and $\mathtt{sigma\_w} = \sigma_w$ of $w[n]$.
To allow an investigation of the filter behavior in a somewhat statistical sense, $R = 100$ realizations of the signals $x[n]$ and $d[n]$, both with length $L = 10000$ are provided: each row of `X` and `D` contains a new realization, resulting in signal matrices with equal size $\mathtt{X} \in \mathbb{R}^{R\times L}$ and $\mathtt{D} \in \mathbb{R}^{R\times L}$. The underlying problem is a general noisy system identification, as shown in Fig. 1 or Fig. 2, where the latter also visualizes the signals available at the filter for performing the identification task. Depending on what type of adaption algorithm is used for finding the optimal coefficients, the signal statistics might be needed, the estimation of which is treated in Problem 2.2. For comparison, the true statistics used for generating the data are given here.

Filter Input Signal The input signal $x[n]$ is a superposition of a sinusoidal with a uniformly distributed random phase $\varphi \sim \mathcal{U}(-\pi, \pi]$, frequency $\theta = \pi/2$ and amplitude $A = 1$ and zero mean white Gaussian noise with variance $\sigma_u^2 = 4$, i.e., $u[n] \sim \mathcal{N}(0, \sigma_u^2)$. The (stationary) random signal $x[n]$ can thus be written as

$$x[n] = A\sin(\theta n + \varphi) + u[n].$$

with autocorrelation function given as

$$r_{xx}[k] = \frac{A^2}{2}\cos(\theta k) + \sigma_u^2\delta[k]$$

Reference Signal The reference signal $d[n]$ is corrupted by zero-mean additive white Gaussian noise $w[n] \sim \mathcal{N}(0, \sigma_w^2)$ with variance $\sigma_w^2 = 0.01$ after passing through the unknown system $\mathbf{h}$ and can thus be written as

$$d[n] = \mathbf{h}^\mathsf{T}\mathbf{x}[n] + w[n].$$

Unknown System The true coefficient vector $\mathbf{h}$ of the system which should be identified is given as

$$\mathbf{h} = [2,\ -0.5,\ 4,\ -2,\ -1,\ 2]^\mathsf{T}$$

which is only needed for the performance evaluation of the algorithms, i.e., to compute the misalignment vectors $\mathbf{v}[n] = \mathbf{c}[n] - \mathbf{h}$ at each iteration step.

1 Analytical Problem 2.1 - Deterministic - Gradient Algorithm

(a)

$$J(\underline{c}[n]) = E[e[n]^2] + \alpha \, ||\underline{c}[n]||^2$$
$$J(\underline{c}[n]) = E[|d[n]|^2] - 2\,(\underline{p}^H \, \underline{c})^T + \underline{c}^H \, \underline{R}_{xx} \, \underline{c} + alpha \, ||\underline{c}[n]||^2$$

$$\underline{\nabla} J_{MSE}(\underline{c}[n]) = 0 - 2 \underbrace{\underline{p}^*}_{\in \mathbb{R}} + 2 \, \underline{R}_{xx} \, \underline{c}[n] + \alpha \, 2 \, \underline{c}[n]$$
$$\underline{\nabla} J_{MSE}(\underline{c}[n]) = 2\,(\underline{R}_{xx} \, \underline{c}[n] - \underline{p}) + 2 \, \alpha \, \underline{c}[n]$$

Gradient search: $-\underline{\nabla} J_{MSE}(\underline{c})|_{c=c[n-1]}$

$$-\underline{\nabla} J_{MSE}(\underline{c})|_{c=c[n-1]} = 2\,(\underline{R}_{xx} \, \underline{c}[n-1] - \underline{p}) + 2 \, \alpha \, \underline{c}[n-1]$$
$$\text{Update rule: } \underline{c}[n] = \underline{c}[n-1] + \underbrace{\widetilde{\mu}}_{\mu = \widetilde{\mu} \, 2} \, ((\underline{p} - \underline{R}_{xx} \, \underline{c}[n-1]) - \alpha \, c[n-1]) \, 2$$
$$\underline{c}[n] = \underline{c}[n-1] - \mu \, \alpha \, c[n-1] \, + \mu(\underline{p} - \underline{R}_{xx} \, \underline{c}[n-1])$$
$$\underline{c}[n] = \underline{c}[n-1] \, (1 - \mu \, \alpha) + \mu(\underline{p} - \underline{R}_{xx} \, \underline{c}[n-1])$$

(b) Where does it converge to?

$$n \to \infty \;\Rightarrow\; c_\infty = ?$$

$$\underline{c}[n] = \underline{c}[n-1] \, (1 - \mu \, \alpha) + \mu(\underline{p} - \underline{R}_{xx} \, \underline{c}[n-1])$$
$$\underline{c}_\infty = \underline{c}_\infty \, (1 - \mu \, \alpha) + \mu(\underline{p} - \underline{R}_{xx} \, \underline{c}_\infty)$$
$$\underline{c}_\infty \, (1 - 1 + \mu \, \alpha) = \mu \, (\underline{p} - \underline{R}_{xx} \, \underline{c}_\infty)$$
$$\underline{c}_\infty \, \cancel{\mu} \, \alpha = \cancel{\mu} \, (\underline{p} - \underline{R}_{xx} \, \underline{c}_\infty)$$
$$(\underline{R}_{xx} + \underline{I} \, \alpha) \, \underline{c}_\infty = \underline{p}$$
$$\underline{c}_\infty = (\underline{R}_{xx} + \underline{I} \, \alpha)^{-1} \, \underline{p}$$

(c) Misalignment vector $\underline{v}[n] = \underline{c}[n] - \underline{c}_\infty$

$$\underline{c}[n] = \underline{c}[n-1] \, (1 - \mu \, \alpha) + \mu(\underline{p} - \underline{R}_{xx} \, \underline{c}[n-1])$$
$$\underline{c}[n] - \underline{c}_\infty = \underline{c}[n-1] \, (1 - \mu \, \alpha) - \underline{c}_\infty + \mu(\underbrace{\underline{p}}_{= (\underline{R}_{xx} + \alpha) \, \underline{c}_\infty} - \underline{R}_{xx} \, \underline{c}[n-1]) - (1 - \mu \, \alpha) \, \underline{c}_\infty + (1 - \mu \, \alpha) \, \underline{c}_\infty$$
$$\underline{v}[n] = (1 - \mu \, \alpha) \, \underline{v}[n-1] - \cancel{\underline{c}_\infty} + \cancel{\underline{c}_\infty} - \mu \, \alpha \, \underline{c}_\infty + \mu(\underline{R}_{xx} \, \underline{c}_\infty + \alpha \underline{c}_\infty - \underline{R}_{xx} \, \underline{c}[n-1])$$
$$\underline{v}[n] = (1 - \mu \, \alpha) \, \underline{v}[n-1] - \cancel{\mu \, \alpha \, \underline{c}_\infty} + \cancel{\mu \, \alpha \, \underline{c}_\infty} + \mu \, \underline{R}_{xx} \, (-\underline{v}[n-1])$$
$$\underline{v}[n] = (1 - \mu \, \alpha) \, \underline{v}[n-1] + \mu \, \underline{R}_{xx} \, (-\underline{v}[n-1])$$
$$\underline{v}[n] = ((1 - \mu \, \alpha) \, \underline{I} - \mu \, \underline{R}_{xx}) \, \underline{v}[n-1]$$

(d) decoupled

$$\underline{v}[n] = ((1-\mu\ \alpha)\ \underline{I} - \mu\ \underline{R}_{xx})\ \underline{v}[n-1]$$

Eigendecomposition: $\underline{R}_{xx} = \underline{Q}\ \underline{\Lambda}\ \underline{Q}^H$

$\underline{Q}$... unitary matrix; $\underline{Q}^{-1} = \underline{Q}^H$

$\underline{\Lambda}$... diagonal matrix consisting of eigenvalues; $\underline{\Lambda} = diag(\lambda_1, \lambda_2, ..., \lambda_{n-1})$

$$||\underline{Q}^H\ \underline{a}||_2^2 = \underline{a}^H\ \underline{Q}\ \underline{Q}^H\ \underline{a} = \underline{a}^H\ \underline{I}\ \underline{a} = ||\underline{a}||_2^2$$

Q matrix does not scale vector, it only rotates it

$$\underline{v}[n] = ((1-\mu\ \alpha)\ \underline{I} - \mu\ \underline{Q}\ \underline{\Lambda}\ \underline{Q}^H)\ \underline{v}[n-1]$$

$$\underbrace{\underline{Q}^H\ \underline{v}[n]}_{=\underline{\tilde{v}}[n]} = (1-\mu\ \alpha)\ \underline{Q}^H\ \underline{v}[n-1] - \underline{Q}^H\ \mu\ \underline{Q}\ \underline{\Lambda}\ \underline{Q}^H\ \underline{v}[n-1]$$

$$\underline{\tilde{v}}[n] = (1-\mu\ \alpha)\ \underline{\tilde{v}}[n-1] - \mu\ \underbrace{\underline{Q}^H\ \underline{Q}}_{=\underline{I}}\ \underline{\Lambda}\ \underline{\tilde{v}}[n-1]$$

$$\underline{\tilde{v}}[n] = (1-\mu\ \alpha)\ \underline{\tilde{v}}[n-1] - \mu\ \underline{\Lambda}\ \underline{\tilde{v}}[n-1]$$

(e)

$$\underline{\tilde{v}}[n] = ((1-\mu\ \alpha)\ - \mu\ \underline{\Lambda})\ \underline{\tilde{v}}[n-1]$$

$$\underline{\tilde{v}}[n] = ((1-\mu\ \alpha)\ - \mu\ \underline{\Lambda})\ ((1-\mu\ \alpha)\ - \mu\ \underline{\Lambda})\ \underline{\tilde{v}}[n-2]$$

$$\underline{\tilde{v}}[n] = ((1-\mu\ \alpha)\ - \mu\ \underline{\Lambda})^n\ \underline{\tilde{v}}[0]$$

(f)

$$\underline{\tilde{v}_i}[n] = ((1-\mu\ \alpha)\ - \mu\ \underline{\lambda}_i)^n\ \underline{\tilde{v}_i}[0]$$

$$\underbrace{|\underline{\tilde{v}_i}|[n]}_{\underline{\tilde{v}_i}[n]=0\ for\ n\to\infty} = |\underbrace{(1-\mu\ \alpha)\ - \mu\ \underline{\lambda}_i|^n}_{exponential\ decoy:\ |e^{-\frac{n}{t_i}}|}\ |\underline{\tilde{v}_i}[0]|$$

$$|(1-\mu\ \alpha)\ - \mu\ \underline{\lambda}_i|^n = |e^{-\frac{n}{t_i}}|$$

$$e^{-\frac{n}{t_i}} = |(1-\mu\ \alpha)\ - \mu\ \underline{\lambda}_i|^n$$

$$-\frac{\not{n}}{t_i}\ ln(e) = \not{n}\ ln(|(1-\mu\ \alpha)\ - \mu\ \underline{\lambda}_i|)$$

$$t_i = -\frac{1}{ln(|(1-\mu\ \alpha)\ - \mu\ \underline{\lambda}_i|)}$$

convergence time can be changed with changing the step size (or eigenvalue)

10.1

2 OCTAVE/MATLAB Problem 2.2 - Signal Statistics

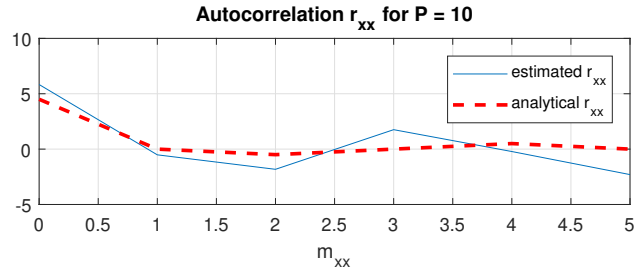
Task a)
In the first task it should be shown, that the estimated quantities are equal to the analytical within the expectation operator i.e. $\mathbf{E}\{\hat{\mu}[P]\} = \mu_x$ and $\mathbf{E}\{\hat{r}_{xy}[k,P]\} = r_{xy}[k]$ assuming $x[n]$ and $y[n]$ are jointly WSS.

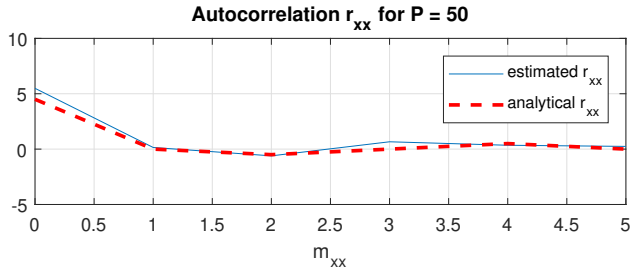
$$\begin{aligned}
E\{\hat{\mu}[p]\} &= E\left\{\frac{1}{P}\sum_{n=0}^{P-1} x[n]\right\} \\
&= E\left\{\frac{1}{P}\Big(x[0] \; + \; x[1] + \; \ldots \; + x[P-1]\Big)\right\} \\
&= \frac{1}{P}\Big(E\{x[0]\} + E\{x[1]\} + \; \ldots \; + E\{x[P-1]\}\Big) \\
&= \frac{1}{P}\sum_{n=0}^{P-1} E\{x[n]\} \\
&= \frac{1}{P}\sum_{n=0}^{P-1} \mu_x \\
&= \frac{1}{P}\cdot P \cdot \mu_x \\
&= \mu_x
\end{aligned}$$

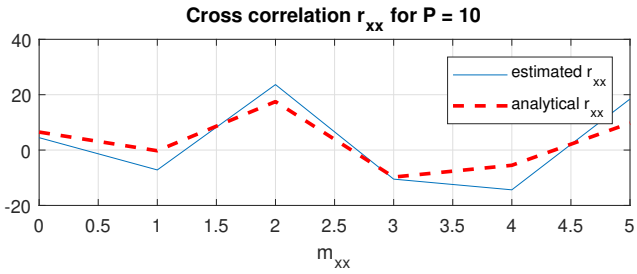
$$\begin{aligned}
E\{\hat{r}_{xy}[k,P]\} &= E\left\{\frac{1}{P}\sum_{n=0}^{P-1} x[n+k]\cdot y[n]\right\} \\
&= \frac{1}{P}\sum_{n=0}^{P-1} E\Big\{\; x[n+k]\cdot y[n] \;\Big\} \\
&= \frac{1}{P}\cdot P \cdot r_{xy}[k] \\
&= r_{xy}[k]
\end{aligned}$$

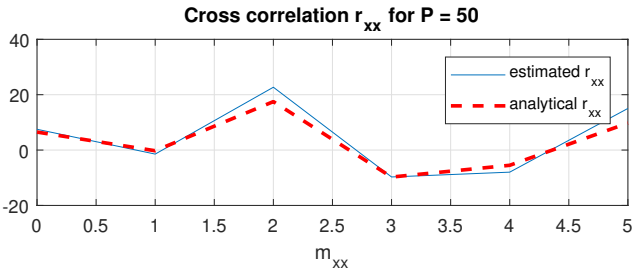
Task b)
Now a Matlab function shall be written, which estimates the auto- and cross correlation. The Matlab Code (`cross_correlation()`) can be found in the zip file and is also appended at the end of this chapter.

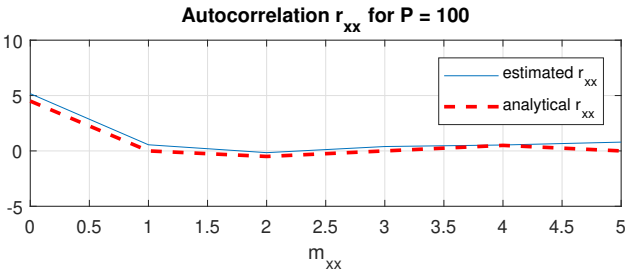
Task c)
The third task makes use of the just written function and calculates the correlations for different windows sizes P. The plots are given below.

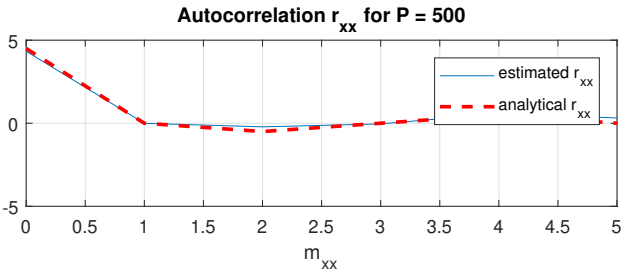

Autocorrelation r_xx for P = 10
estimated r_xx
analytical r_xx
m_xx

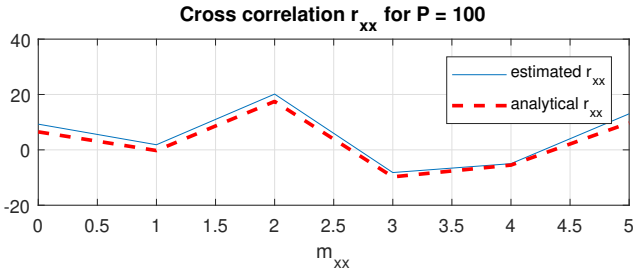

Autocorrelation r_xx for P = 50
estimated r_xx
analytical r_xx
m_xx

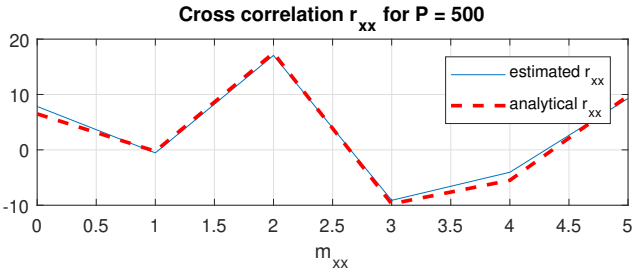

Cross correlation r_xx for P = 10
estimated r_xx
analytical r_xx
m_xx

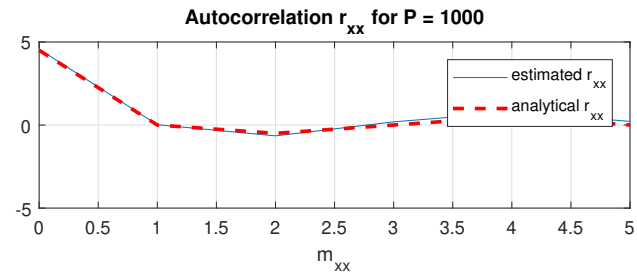

Cross correlation r_xx for P = 50
estimated r_xx
analytical r_xx
m_xx

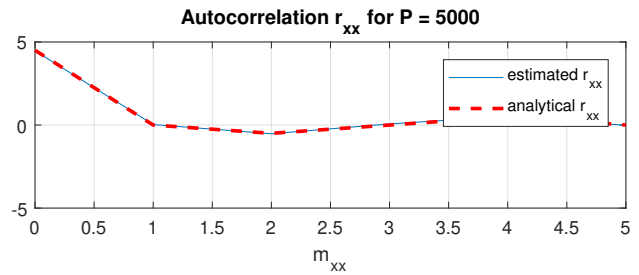

Autocorrelation r_xx for P = 100
estimated r_xx
analytical r_xx
m_xx

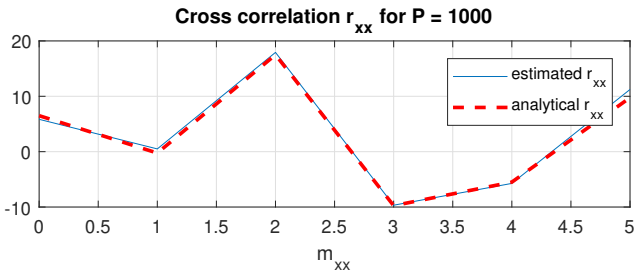

Autocorrelation r_xx for P = 500
estimated r_xx
analytical r_xx
m_xx

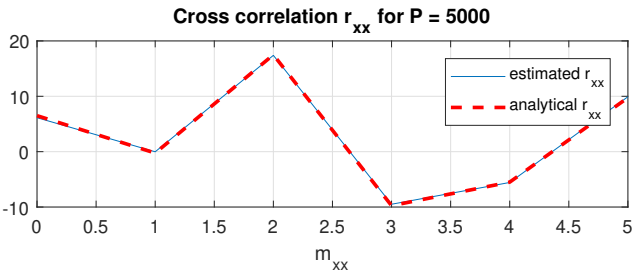

Cross correlation r_xx for P = 100
estimated r_xx
analytical r_xx
m_xx


Cross correlation r_xx for P = 500
estimated r_xx
analytical r_xx
m_xx


Autocorrelation r_xx for P = 1000
estimated r_xx
analytical r_xx
m_xx


Autocorrelation r_xx for P = 5000
estimated r_xx
analytical r_xx
m_xx


Cross correlation r_xx for P = 1000
estimated r_xx
analytical r_xx
m_xx


Cross correlation r_xx for P = 5000
estimated r_xx
analytical r_xx
m_xx

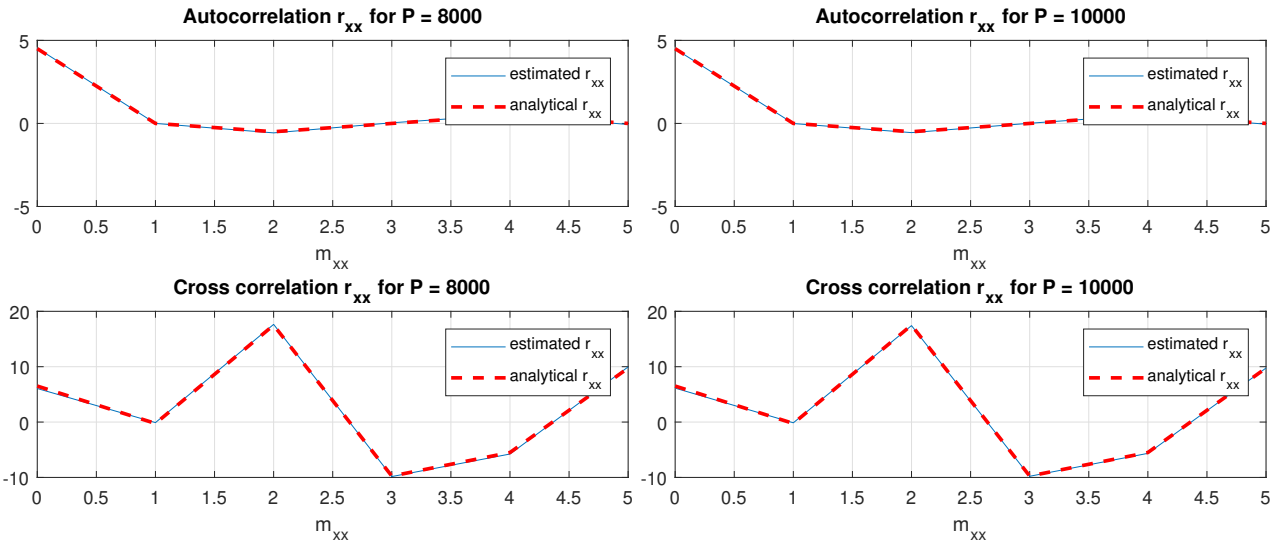


Figure 1: Auto- and cross correlation for different window sizes P

The first plot with window size $P = 10$ deviates quite a lot from the analytical solutions. The estimated correlations (which are computed by time average and not ensemble average, as the expectation operator does it) need more samples to get accurate values.

For $P = 50$ the estimated solutions look a lot better. Auto- as well as cross-correlation deviates less from the analytical solution but there is still room for improvement.

For $P = 100$ and $P = 500$ a small error can still be seen, but this error distinguishes for $P = 1000$ and above. The estimated and analytical solutions look identical.

Hence, with increasing window size P, the accuracy of the estimator increases. This was validated for several different realizations, which are not depicted in this document, but can easily be checked by running the Matlab code.

2.1 Matlab Code

Main File

```
close all
clear all
clc

mkdir Figures

load data.mat

%analytical solution
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


%estimated solution
N = length(h);

for ii = 1:2% amount of different realizations to be plottet

 RNG = round(rand()*size(X,1)); %for each iteration choose a random realization

  for P = [10, 50, 100, 500, 1000, 5000, 8000, 10000]


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

      figure
         subplot(2,1,1)
              plot(mxx, rxx)
              grid on
              xlabel('m_{xx}')
              title( ['Autocorrelation r_{xx} for P = ' num2str(P)] )
              hold on
              plot(mxx, rxx_ref(mxx), '--r', 'LineWidth', 2);
              hold off
              legend('estimated r_{xx}', 'analytical r_{xx}')

         subplot(2,1,2)
              plot(mdx, rdx) %rdx = p
              grid on
              xlabel('m_{xx}')
              title( ['Cross correlation r_{xx} for P = ' num2str(P)] )
              hold on
              plot(mdx, p_ref, '--r', 'LineWidth', 2);
              hold off
```

```
            legend('estimated r_{xx}', 'analytical r_{xx}')


        if ii == 1   %only save for first realization
            saveas(gcf,['Figures/P=' num2str(P)], 'epsc')
        end

 end
end

 %With increasing window size P, the estimated values for rxx and rdx get
 %closer and closer to the analytical solution

 function saveas(~,~,~)
    disp('Figure not saved')
 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)
```

1 Analytical Problem 2.3 - LMS Algorithm

(a)

$$\begin{aligned}
\underline{c}[n] &= \underline{c}[n-1] \; + \; \mu e[n]\underline{x}[n] \\
E[\underline{c}[n]] &= E[\underline{c}[n-1]] = E[\underline{c}_\infty] \\
e[n] &= \underline{h}^T\underline{x}[n] - \underline{c}^T\underline{x}[n] + \omega[n]
\end{aligned}$$

$$\begin{aligned}
\underline{c}[n] &= \underline{c}[n-1] + \mu\,(\underline{h}^T\,\underline{x}[n])\,\underline{x}[n] + \mu\,\omega[n]\,\underline{x}[n] - \mu\,(\underline{c}[n]^T\,\underline{x}[n])\,\underline{x}[n] \\
\underline{c}[n] &= \underline{c}[n-1] + \mu\,\underline{x}[n]\,\underline{h}^T\,\underline{x}[n] + \mu\,\omega[n]\,\underline{x}[n] - \mu\,\underline{x}[n]\,\underline{c}[n]^T\,\underline{x}[n] \\
\underline{c}[n] &= \underline{c}[n-1] + \mu\,\underline{x}[n]\,\underline{x}[n]^T\,\underline{h} + \mu\,\omega[n]\,\underline{x}[n] - \mu\,\underline{x}[n]\,\underline{x}[n]^T\,\underline{c}[n]
\end{aligned}$$

$$\begin{aligned}
E[\underline{c}[n]] &= E[\,\underline{c}[n-1] + \mu\,\underline{x}[n]\,\underline{x}[n]^T\,\underline{h} + \mu\,\omega[n]\,\underline{x}[n] - \mu\,\underline{x}[n]\,\underline{x}[n]^T\,\underline{c}[n]\,] \\
\cancel{E[\underline{c}[n]]} &= \cancel{E[\,\underline{c}[n-1]]} + \mu\,E[\underline{x}[n]\,\underline{x}[n]^T\,]\,\underline{h} \\
&\quad + \mu\,E[\,\omega[n]\,\underline{x}[n]\,] - \mu\,E[\,\underline{x}[n]\,\underline{x}[n]^T\,]\,\underline{c}[n] \\
\mu\,E[\,\underline{x}[n]\,\underline{x}[n]^T\,]\,\underline{c}[n] &= \mu\,E[\underline{x}[n]\,\underline{x}[n]^T\,]\,\underline{h} + \mu\,E[\,\omega[n]\,\underline{x}[n]\,] \\
E[\,\underline{x}[n]\,\underline{x}[n]^T\,]\,\underline{c}[n] &= E[\underline{x}[n]\,\underline{x}[n]^T\,]\,\underline{h} + E[\,\omega[n]\,\underline{x}[n]\,] \\
\underline{R}_{xx}\,E[\underline{c}[n]] &= \underline{R}_{xx}\,\underline{h} + 0 \\
E[\underline{c}[n]] &= \underline{h}
\end{aligned}$$

$$\begin{aligned}
\underline{c}[n] &= (1-\mu\alpha)\underline{c}[n-1] \; + \; \mu e[n]\underline{x}[n] \\
E[\underline{c}[n]] &= E[\underline{c}[n-1]] = E[\underline{c}_\infty] \\
e[n] &= \underline{h}^T\underline{x}[n] - \underline{c}^T\underline{x}[n] + \omega[n]
\end{aligned}$$

$$\begin{aligned}
\underline{c}[n] &= (1-\mu\alpha)\underline{c}[n-1] + \mu\,(\underline{h}^T\,\underline{x}[n])\,\underline{x}[n] + \mu\,\omega[n]\,\underline{x}[n] - \mu\,(\underline{c}[n]^T\,\underline{x}[n])\,\underline{x}[n] \\
\underline{c}[n] &= (1-\mu\alpha)\underline{c}[n-1] + \mu\,\underline{x}[n]\,\underline{h}^T\,\underline{x}[n] + \mu\,\omega[n]\,\underline{x}[n] - \mu\,\underline{x}[n]\,\underline{c}[n]^T\,\underline{x}[n] \\
\underline{c}[n] &= \underline{c}[n-1] \; - \; \mu\alpha\,\underline{c}[n-1] + \mu\,\underline{x}[n]\,\underline{x}[n]^T\,\underline{h} + \mu\,\omega[n]\,\underline{x}[n] \\
&\quad - \mu\,\underline{x}[n]\,\underline{x}[n]^T\,\underline{c}[n]
\end{aligned}$$

$$\begin{aligned}
E[\underline{c}[n]] &= E[\,\underline{c}[n-1] - \mu\alpha\,\underline{c}[n-1] + \mu\,\underline{x}[n]\,\underline{x}[n]^T\,\underline{h} \\
&\quad + \mu\,\omega[n]\,\underline{x}[n] - \mu\,\underline{x}[n]\,\underline{x}[n]^T\,\underline{c}[n]\,] \\
\cancel{E[\underline{c}[n]]} &= \cancel{E[\,\underline{c}[n-1]]} - \mu\alpha\,E[\,\underline{c}[n-1]] + \mu\,E[\underline{x}[n]\,\underline{x}[n]^T\,]\,\underline{h} \\
&\quad + \mu\,E[\,\omega[n]\,\underline{x}[n]\,] - \mu\,E[\,\underline{x}[n]\,\underline{x}[n]^T\,]\,E[\underline{c}[n]] \\
\mu\,E[\,\underline{x}[n]\,\underline{x}[n]^T\,]\,E[\underline{c}[n]] + \mu\alpha\,E[\,\underline{c}[n-1]] &= \mu\,E[\underline{x}[n]\,\underline{x}[n]^T\,]\,\underline{h} + \mu\,E[\,\omega[n]\,\underline{x}[n]\,] \\
E[\,\underline{x}[n]\,\underline{x}[n]^T\,]\,E[\underline{c}[n]] + \alpha\,E[\,\underline{c}[n-1]] &= E[\underline{x}[n]\,\underline{x}[n]^T\,]\,\underline{h} + E[\,\omega[n]\,\underline{x}[n]\,] \\
\underline{R}_{xx}\,E[\underline{c}[n]] + \alpha\underbrace{E[\underline{c}[n-1]]}_{=E[\underline{c}[n]]} &= \underline{R}_{xx}\,\underline{h} + 0 \\
E[\underline{c}[n]] &= (\underline{R}_{xx} + \alpha\,\underline{I})^{-1}(\underline{R}_{xx}\,\underline{h})
\end{aligned}$$

(b) Through the normalization of the LMS the step size parameter gets independent of the energy of the input signal.
α is a small positive constant to avoid division by zero.

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.

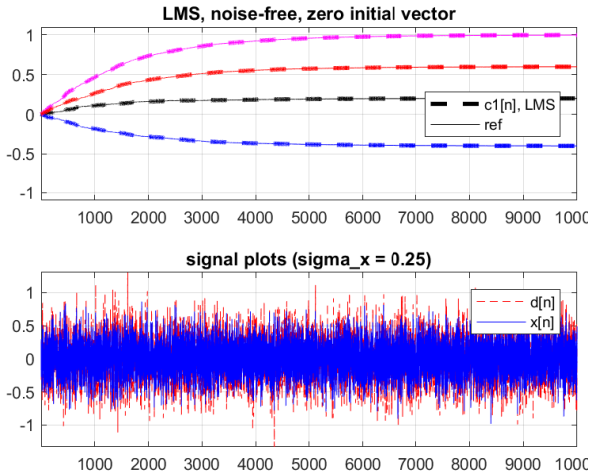


Figure 1: Coefficients for LMS

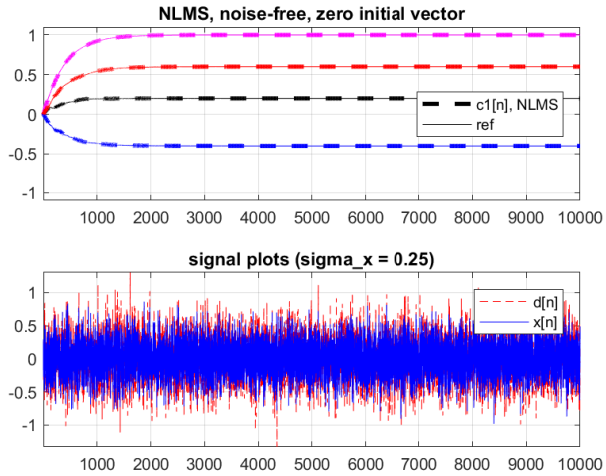


Figure 2: Coefficients for NLMS

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.

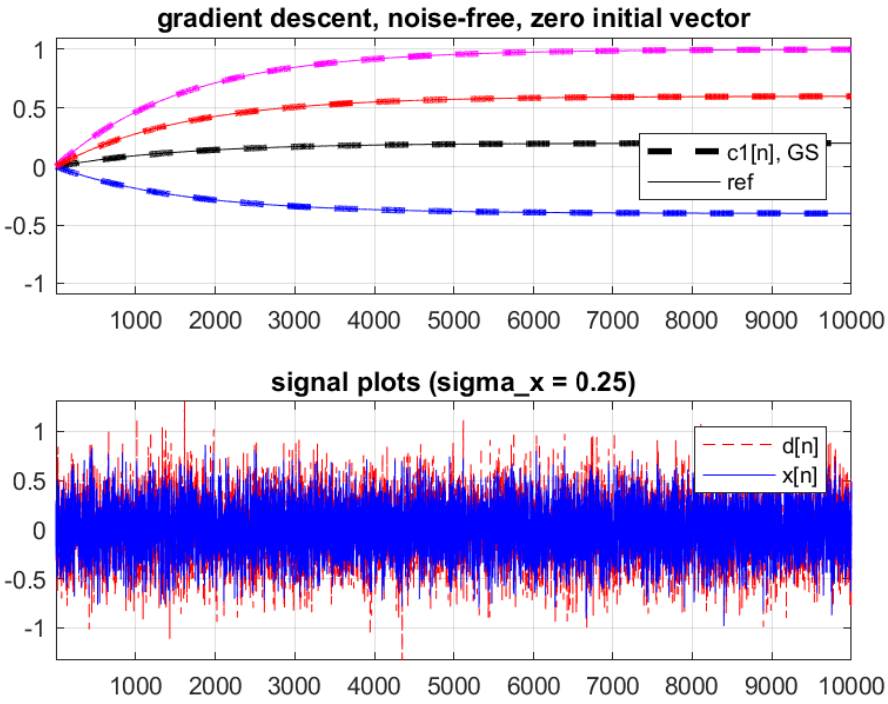


Figure 3: Coefficients for gradient descent

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

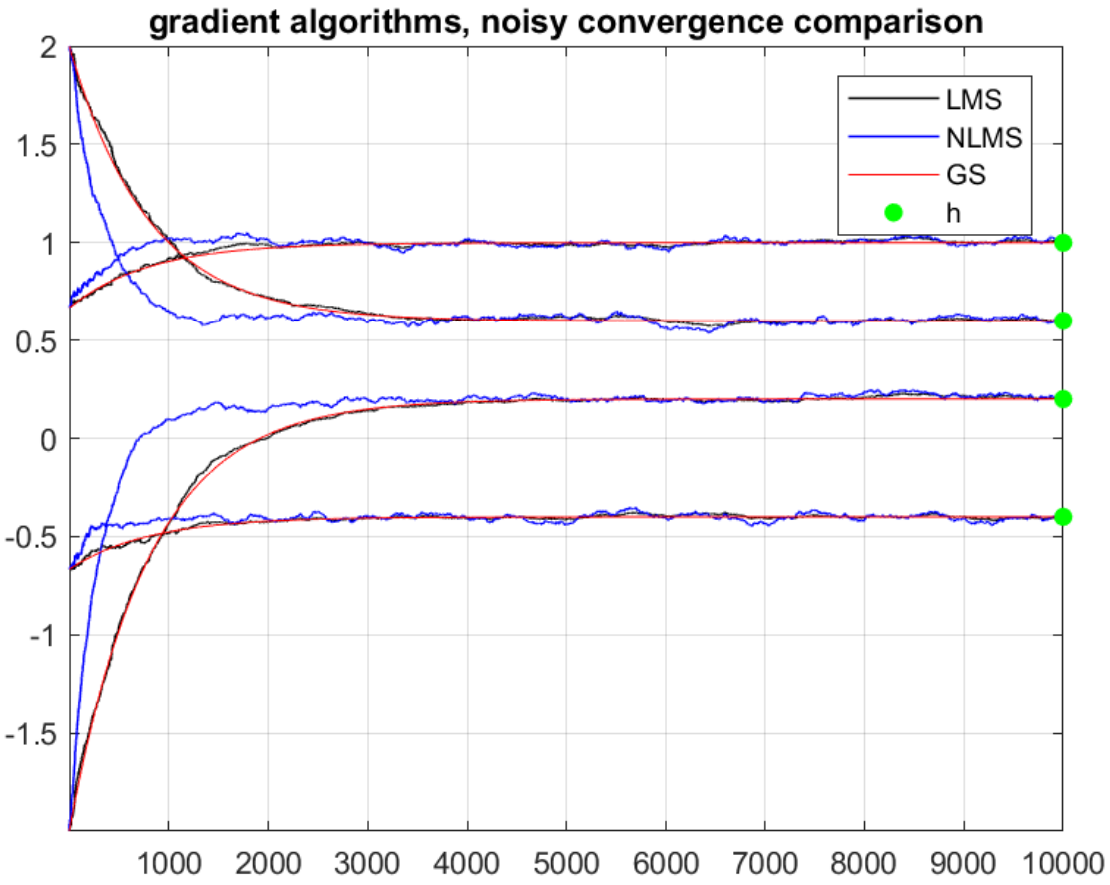


Figure 4: Comparsion of gradient algorithms with noise

LMS: The additional noise causes the LMS to fluctuate a l[20.1]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 environm[20.2]. 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');
```

```
55
56 figure(1)
57 subplot(2,1,1), hold on, grid on, box on
58 plot(c_lms(1,:),'--k','LineWidth',2.5), plot(ref_lms.c_lms(1,:),'k','LineWidth'
       ,0.5)
59 plot(c_lms(2,:),'--b','LineWidth',2.5), plot(ref_lms.c_lms(2,:),'b','LineWidth'
       ,0.5)
60 plot(c_lms(3,:),'--m','LineWidth',2.5), plot(ref_lms.c_lms(3,:),'m','LineWidth'
       ,0.5)
61 plot(c_lms(4,:),'--r','LineWidth',2.5), plot(ref_lms.c_lms(4,:),'r','LineWidth'
       ,0.5)
62 legend('c1[n], LMS','ref','Location','east')
63 xlim([1 Nx]), ylim([-1.1 1.1])
64 title('LMS, noise-free, zero initial vector')
65 subplot(2,1,2), hold on, grid on, box on
66 plot(d,'--r'), plot(x,'b')
67 axis tight
68 legend('d[n]','x[n]')
69 title(sprintf('signal plots (sigma_x = %2.2f)',sigma_x),'Interpreter','none')
70
71 saveas(gcf,'Figures/LMS', 'epsc')
72
73
74 %% NLMS, noise-free, zero initial vector (with reference)
75 c0 = zeros(1,Nc);      %initialize with zero vector
76
77 mu = 0.01;
78 alpha = 0;
79 [~,~,c_nlms]  = lms_algorithm(x,d,Nc,mu,alpha,1); %normalized LMS
80
81 % save('./ref_nlms.mat','c_nlms')
82 ref_nlms = load('./ref_nlms.mat');
83
84 figure(2)
85 subplot(2,1,1), hold on, grid on, box on
86 plot(c_nlms(1,:),'--k','LineWidth',2.5), plot(ref_nlms.c_nlms(1,:),'k','
       LineWidth',0.5)
87 plot(c_nlms(2,:),'--b','LineWidth',2.5), plot(ref_nlms.c_nlms(2,:),'b','
       LineWidth',0.5)
88 plot(c_nlms(3,:),'--m','LineWidth',2.5), plot(ref_nlms.c_nlms(3,:),'m','
       LineWidth',0.5)
89 plot(c_nlms(4,:),'--r','LineWidth',2.5), plot(ref_nlms.c_nlms(4,:),'r','
       LineWidth',0.5)
90 legend('c1[n], NLMS','ref','Location','east')
91 xlim([1 Nx]), ylim([-1.1 1.1])
92 title('NLMS, noise-free, zero initial vector')
93 subplot(2,1,2), hold on, grid on, box on
94 plot(d,'--r'), plot(x,'b')
95 axis tight
96 legend('d[n]','x[n]')
97 title(sprintf('signal plots (sigma_x = %2.2f)',sigma_x),'Interpreter','none')
98
99 saveas(gcf,'Figures/NLMS', 'epsc')
100
101 %% NLMS, noise-free, zero initial vector (with reference)
102 mu = 0.01;
103 alpha = 0;
104
```

```
105 Rxx = sigma_x^2*eye(Nc); %x is white noise?
106 p = Rxx*h;
107
108 [~,~,c_gs]  = gd_algorithm(x,d,Nc,mu,Rxx,p); %gradient descent
109
110 % save('./ref_gd.mat','c_gd')
111 ref_gd = load('./ref_gd.mat');
112
113 figure(3)
114 subplot(2,1,1), hold on, grid on, box on
115 plot(c_gs(1,:),'--k','LineWidth',2.5), plot(ref_gd.c_gs(1,:),'k','LineWidth'
        ,0.5)
116 plot(c_gs(2,:),'--b','LineWidth',2.5), plot(ref_gd.c_gs(2,:),'b','LineWidth'
        ,0.5)
117 plot(c_gs(3,:),'--m','LineWidth',2.5), plot(ref_gd.c_gs(3,:),'m','LineWidth'
        ,0.5)
118 plot(c_gs(4,:),'--r','LineWidth',2.5), plot(ref_gd.c_gs(4,:),'r','LineWidth'
        ,0.5)
119 legend('c1[n], GS','ref','Location','east')
120 xlim([1 Nx]), ylim([-1.1 1.1])
121 title('gradient descent, noise-free, zero initial vector')
122 subplot(2,1,2), hold on, grid on, box on
123 plot(d,'--r'), plot(x,'b')
124 axis tight
125 legend('d[n]','x[n]')
126 title(sprintf('signal plots (sigma_x = %2.2f)',sigma_x),'Interpreter','none')
127
128 saveas(gcf,'Figures/GD', 'epsc')
129
130 %% algorithm comparison: noisy, random intial vector (no references)
131 clear
132 load('./data_rng2.mat')
133
134 Rxx = sigma_x^2*eye(Nc);
135 p = Rxx*h;
136
137 c0 = linspace(-2,2,Nc); %pseudo-random initialization
138
139 mu = 0.02;
140 alpha = 0.1;
141 [~,~,c_lms]  = lms_algorithm(x,d,Nc,mu,alpha,0,c0);
142 [~,~,c_nlms] = lms_algorithm(x,d,Nc,mu,alpha,1,c0);
143 [~,~,c_gs]   = gd_algorithm(x,d,Nc,mu,Rxx,p,c0); %gradient search
144
145 figure(4), hold on, grid on, box on
146 hlms = plot(c_lms.','k','LineWidth',0.75);
147 hnlms = plot(c_nlms.','b','LineWidth',0.75);
148 hgs = plot(c_gs.','r');
149 htrue = scatter(Nx*ones(Nh,1),h,'g','o','filled');
150 axis tight
151 legend([hlms(1),hnlms(1),hgs(1),htrue],{'LMS','NLMS','GS','h'})
152 title('gradient algorithms, noisy convergence comparison')
153
154 saveas(gcf,'Figures/comparison', 'epsc')
155
156 % PLOT DESCRIPTION
157 %
158 % LMS:
```

```
159 % The additional noise causes the LMS to flucatute a lot it looks like it
160 % will never truly converge to h. In the problem class we derived, that it
161 % converges to h (if it converges in the first place) for noise free
162 % enviroment. BUT if there is some noise, e[n] wont be 0 in the optimum
163 % point, hence it never truly converges
164 %
165 % NMLS:
166 % Due to the independet step size to signal energy the NMLS overshoots the
167 % LMS and GS, especially in the beginning and like the LMS it looks like it
168 % can never truly converge if there is some noise.
169 %
170 % GS:
171 % Gradient search creates a smooth looking curve with no fluctuation what so
172 % ever and converges towards h really nicely. It looks like the noise does
173 % not effect the algorithm. The downside of GS is the fact that it needs the
174 % values of cross corellation p and Autocorrelationmatrix Rxx, which are
175 % often not known.
176 %
177 % LMS and NMLS algorithms assume ergodic process, which allow to replace
178 % expectation operator by an time average of one WSS realization for mean
179 % 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

```

```
54      %change mu depending on chosen OPTS (standard or normalized LMS)
55      if OPTS == 1 %normalized LMS
56          mu_calc = mu/(alpha + x_tap'*x_tap); %only the energy of the observed
                current signal
57      end
58
59      c(:,n) = c(:,n-1) + mu_calc*conj(e(n))*x_tap;
60
61  end
62
63  %now delete the first entries of y,e and c which are zero, to keep the time
64  %indices in order
65  y(1:N-1) = [];
66  e(1:N-1) = [];
67  c(:,1:N-1) = [];
68
69  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) = [];

```

```
52 end
```

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.

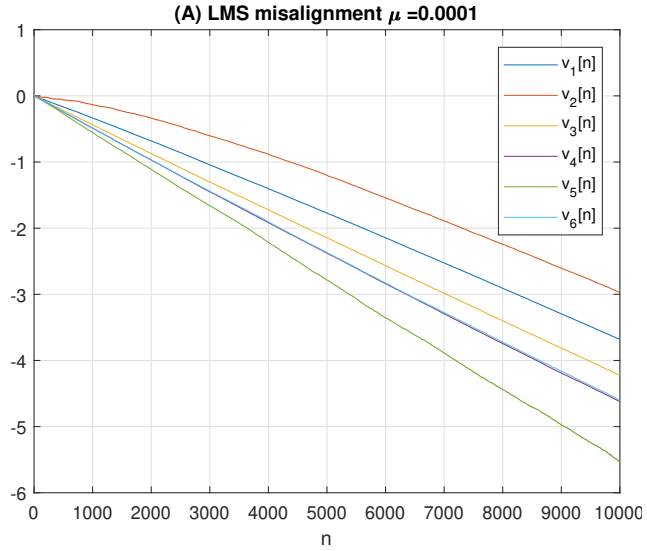
The first formula is:

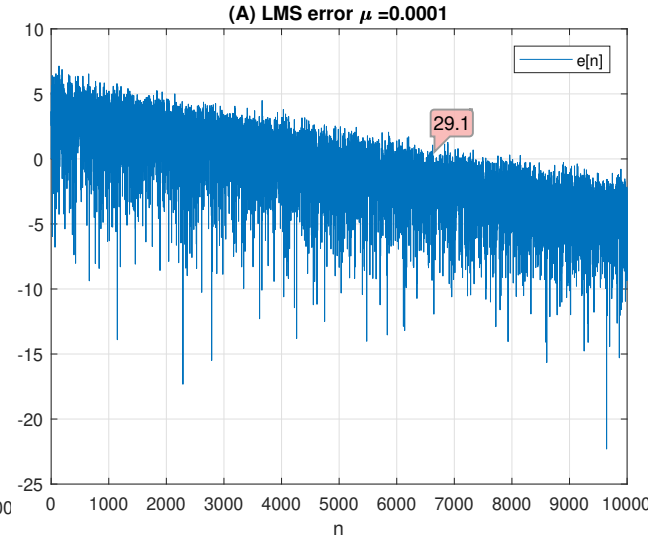
$$\ln\left(\frac{|\mathbf{E}\{v_k[n]\}|}{|\mathbf{E}\{v_k[0]\}|}\right) \tag{1}$$

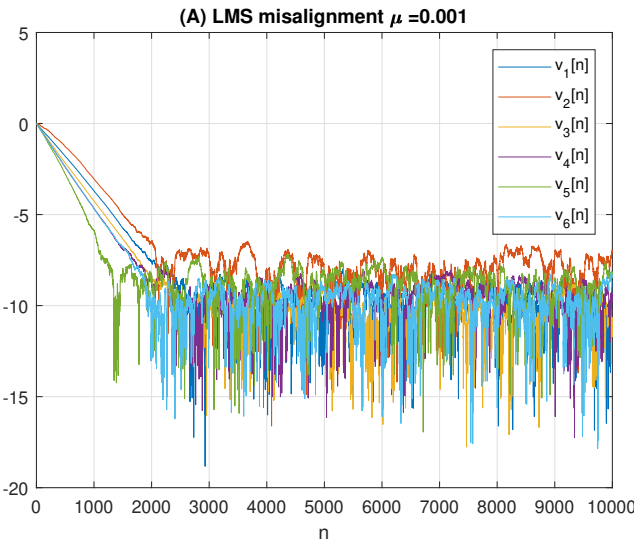
and the second one:

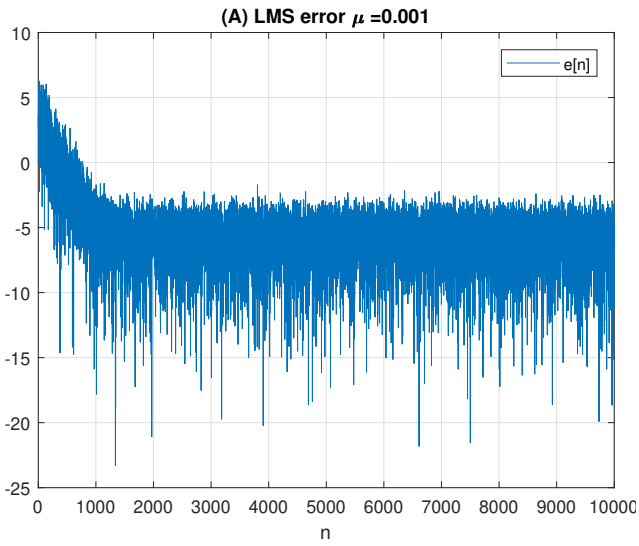
$$\ln\left(\frac{\mathbf{E}\{e^2[n]\}}{\mathbf{E}\{e^2[0]\}}\right) \tag{2}$$

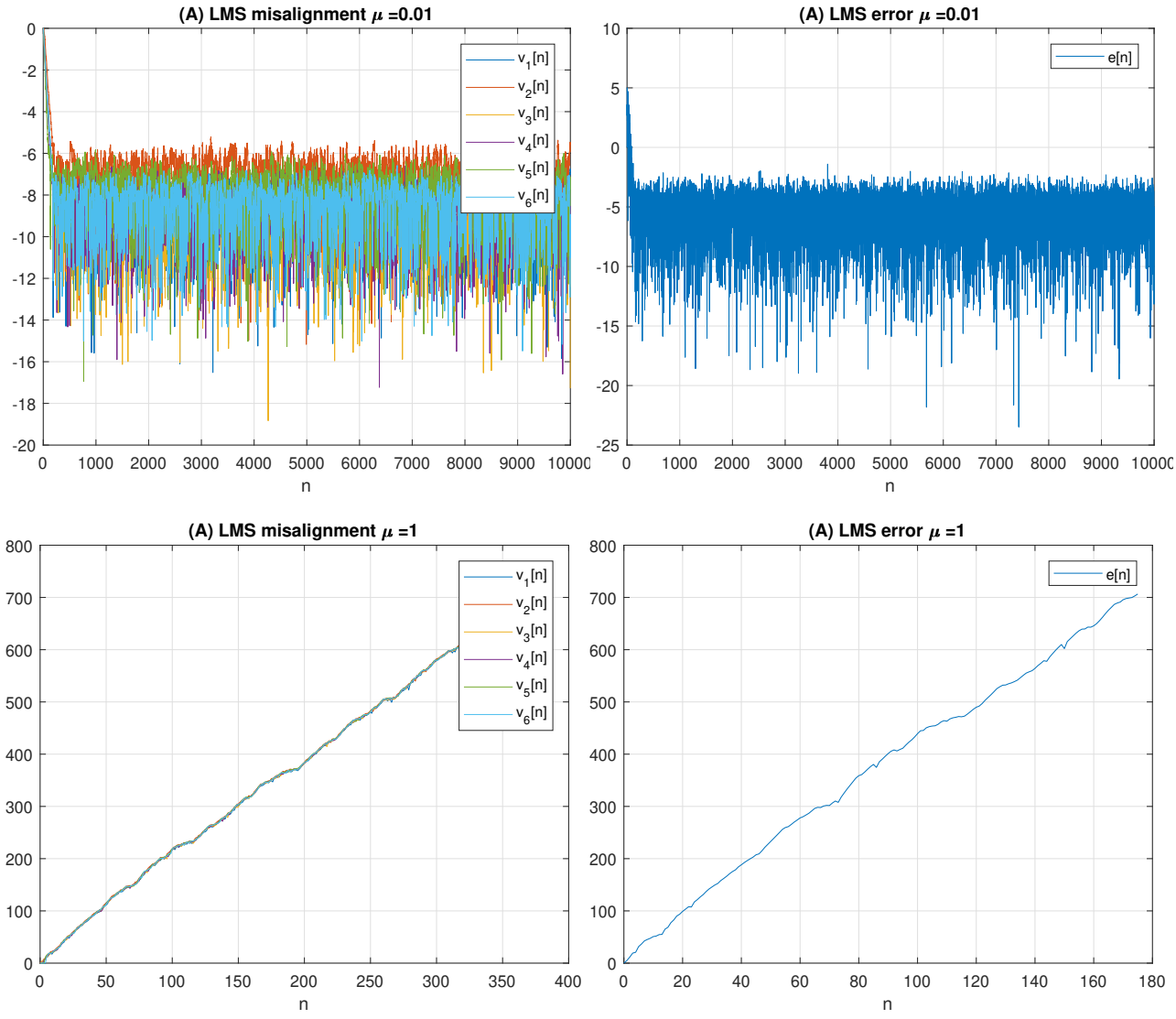
The plots for varying step size of the LMS algorithm:











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:

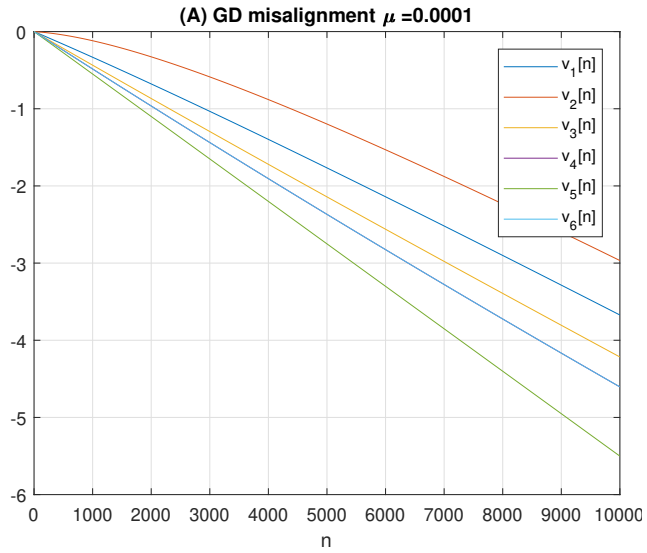
LMS for $\mu = 0.0001$:
The misalignment coefficients keep decreasing within the plot as well the error and the given amount of samples is not enough to reach the point of convergence.

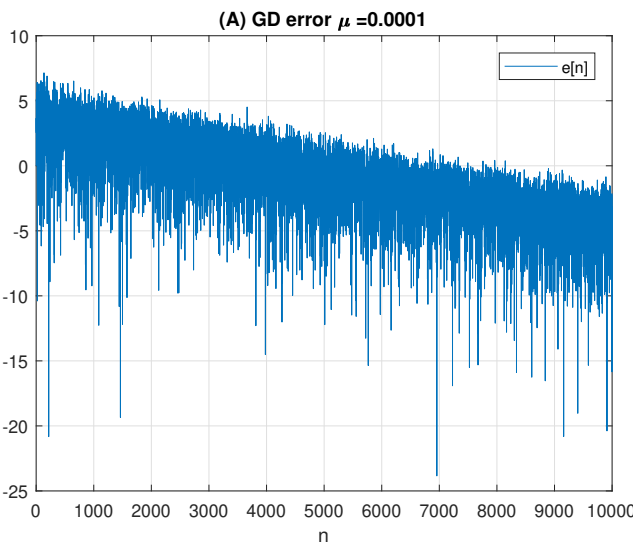
LMS for $\mu = 0.001$:
This time the misalignment coefficients do converge, since after about 2000 samples the coefficients and the error don't keep decreasing anymore. The step size is big enough to allow convergence within the time(i.e. samples).

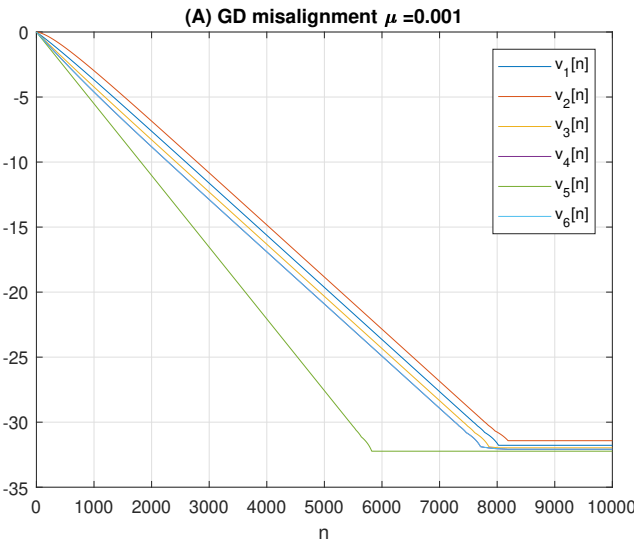
LMS for $\mu = 0.01$:
Both plots reached their 'converged' state much earlier now due to the higher step size μ and there is no sign of divergence due to the large step size.

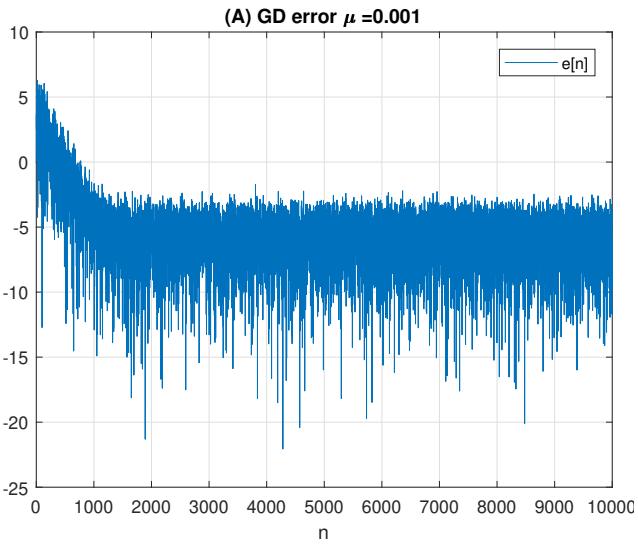
LMS for $\mu = 1$:
Now the step size increased dramatically and the LMS algorithm does not converge anymore. The value of the gradient from the start point is too big, hence we overshoot the valley of the cost function by far and end up at point with an even bigger gradient. This keeps repeating until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

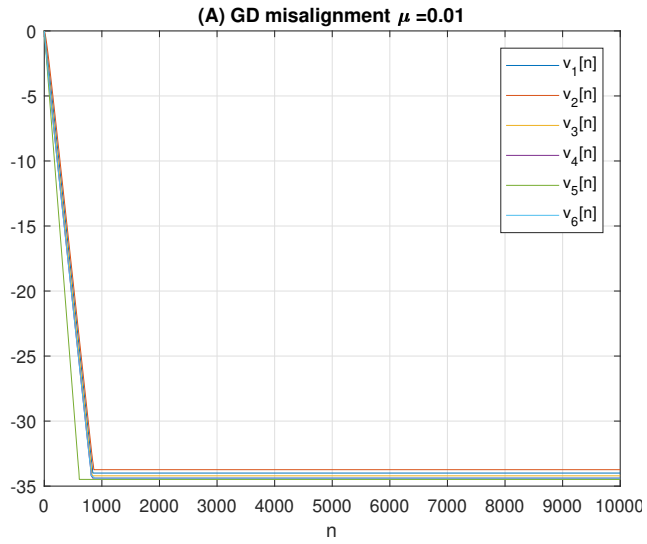
The plots for varying step size of the GD algorithm:

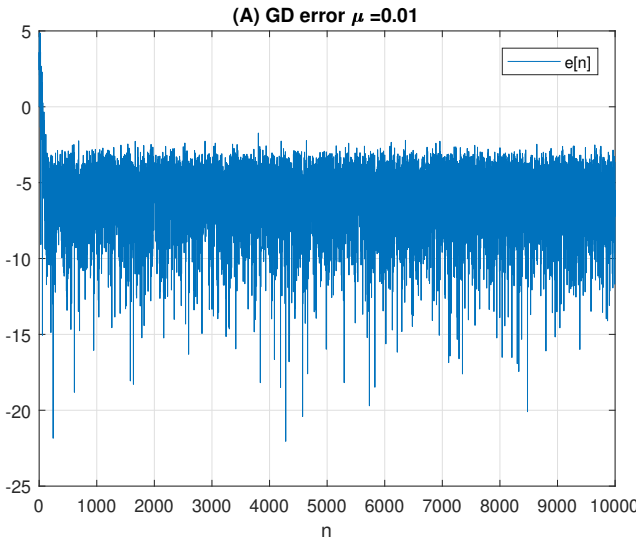


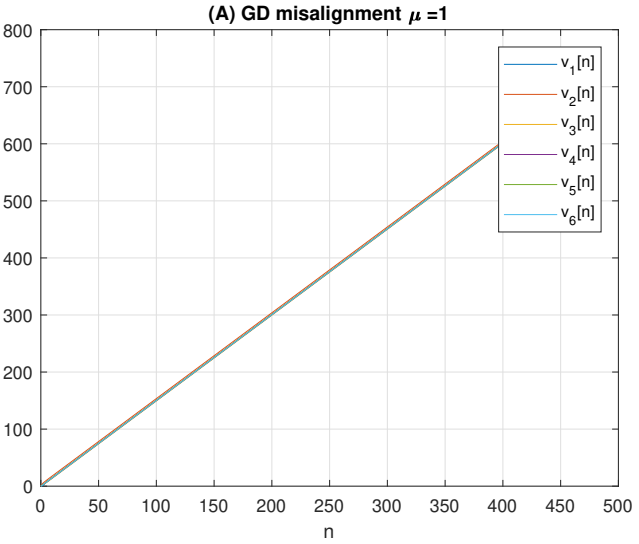


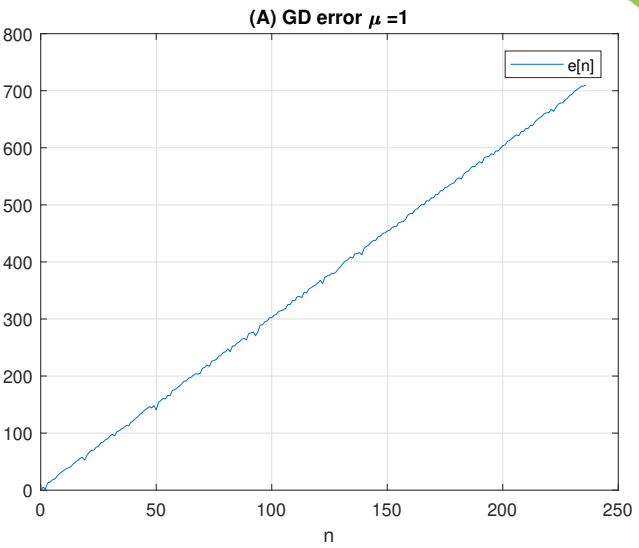












Discussion of the GD plots:

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

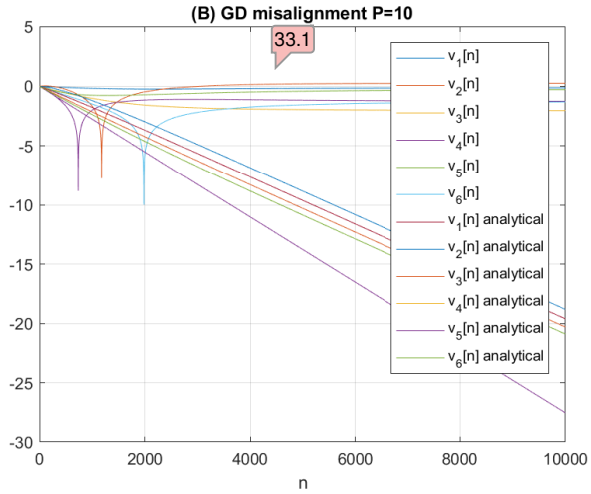
GD for $\mu = 0.001$:
Increasing the step size helps the algorithm to 'converge' much faster and in the GD case the misalignment coefficients get much smaller than in the LMS case. And after 'convergence' there is no random fluctuations for the misalignment, they show a straight line. It should also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

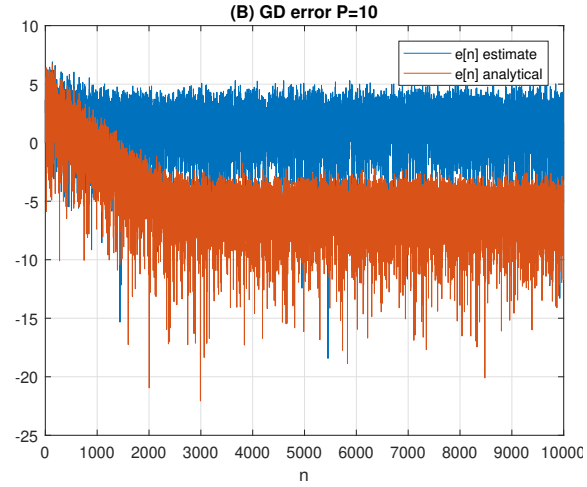
GD for $\mu = 0.01$:
Similar to LMS, the algorithm converges much faster than before.

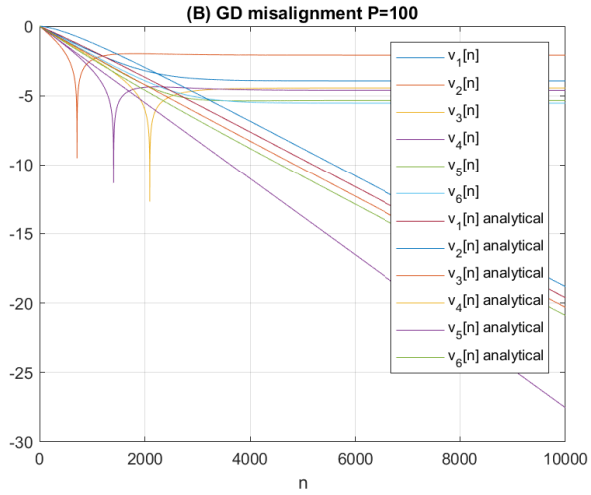
GD for $\mu = 1$:
Similar to LMS, the algorithm diverges.

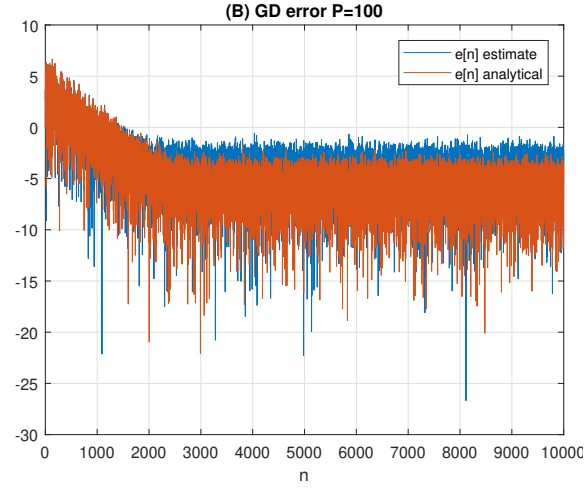
Task b)

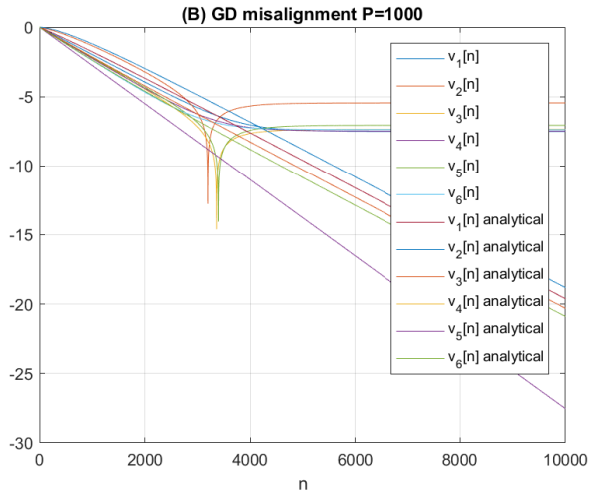
Now the effect of amount of samples should be investigated for a given stepsize $\mu = 0.0005$ using the GD algorithm. Each plot displays the values, once for the analytical solutions for **Rxx** and **p** and once for the estimated statistics.

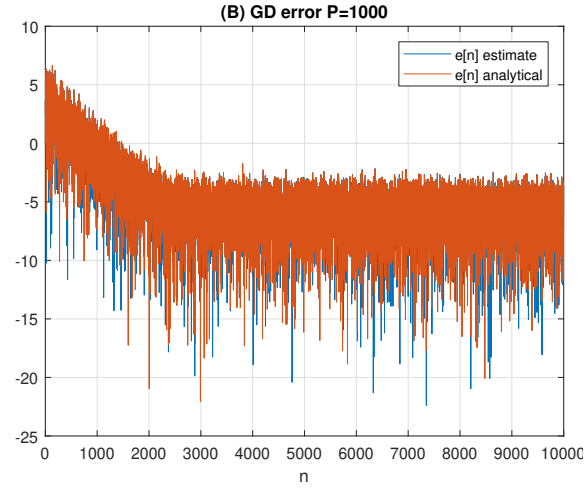






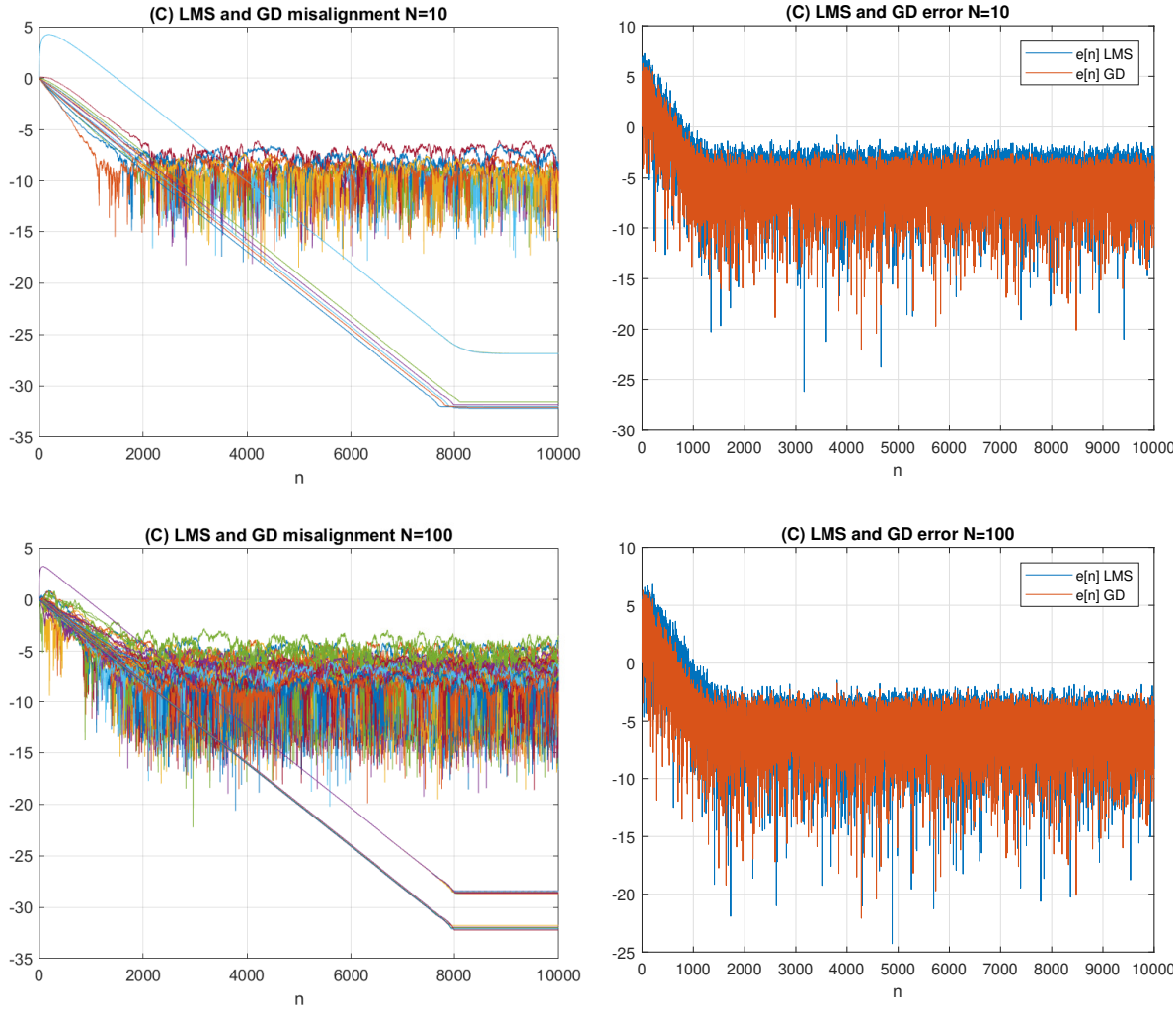






From the plots we can see, the lower the sample size P, the worse the misalignment coefficients and the error is. The smaller sample size causes the sample correlations **Rxx** and **p** to be less precise and therefore also GD is less precise. It should be noted, that the analytical solution does not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

34.1

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.

35.1

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
```

```
53
54          saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')
55
56
57      figure
58          plot(n,e_plot)
59          title(['(A) LMS error \mu =' num2str(mu)])
60          legend('e[n]')
61          xlabel('n')
62          grid on
63
64          saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')
65
66  end %end for mu
67
68
69
70  %-------------------------
71  %Gradient Search
72
73
74  %get Rxx and p
75  h = [2, -0.5, 4, -2, -1, 2].';
76
77  A = 1;
78  theta = pi/2;
79  sigma_u = sqrt(4);
80
81  rxx_ref = @(k) A^2/2 * cos(theta*k) + sigma_u^2 * kroneckerDelta(sym(k));
82  Rxx_ref = double(toeplitz(rxx_ref(0:length(h)-1))); %double convertes the sym
        data to double precision numbers
83
84  p_ref = Rxx_ref * h; %c_MSE = h = Rxx^-1 * p
85
86
87  v = zeros(N,size(X,2),size(X,1));
88  e = zeros(size(X,1), size(X,2));
89
90  for mu = mu_list
91
92      for r = 1:size(X,1) %r...realization index
93          x = X(r,:);
94          d = D(r,:);
95
96          [~,e(r,:),c] = gd_algorithm(x,d,N,mu,Rxx_ref,p_ref);
97
98          v(:,:,r) = c - h;
99      end
100
101     %compute the expectation by averaging over different realizations
102     e_expect = mean(e,1); %mean along the 1 dimension (rows get added)
103     v_expect = mean(v,3); %mean along the third dimension (2D matrixes with row
            and col get added)
104
105
106     v_plot = log(abs(v_expect)./abs(v_expect(:,1)));
107     e_plot = log(e_expect.^2/e_expect(1).^2);
108
```

```
109
110     n = 0:size(X,2)-1;
111
112     figure
113         plot(n,v_plot)
114         title(['(A) GD misalignment \mu =' num2str(mu)])
115         legend('v_1[n]', 'v_2[n]', 'v_3[n]', 'v_4[n]', 'v_5[n]', 'v_6[n]')
116         xlabel('n')
117         grid on
118
119         saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')
120
121
122     figure
123         plot(n,e_plot)
124         title(['(A) GD error \mu =' num2str(mu)])
125         legend('e[n]')
126         xlabel('n')
127         grid on
128
129         saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')
130
131 end %end for mu
132
133 % ------------------------------------------------------------------------
134 % Task b)
135
136 mu = 0.0005;
137 P_list = [10, 100, 1000];
138 % P_list = [];
139
140 r = 1; %used realization
141 N = length(h);
142
143 for P = P_list
144
145     [rxx, mxx] = cross_correlation(X(r,:),X(r,:),P,N);
146     [rdx, mdx] = cross_correlation(D(r,:),X(r,:),P,N);
147
148     Rxx = toeplitz(rxx);
149     p = rdx;
150
151
152     for r = 1:size(X,1) %r...realization index
153         x = X(r,:);
154         d = D(r,:);
155
156         %use estimated statistics
157         [~,e(r,:),c] = gd_algorithm(x,d,N,mu,Rxx,p);
158         v(:,:,r) = c - h;
159
160         %use analytical statistics
161         [~,e_an(r,:),c_an] = gd_algorithm(x,d,N,mu,Rxx_ref,p_ref);
162         v_an(:,:,r) = c_an - h;
163     end
164
165     %compute the expectation by averaging over different realizations
166     e_expect = mean(e,1); %mean along the 1 dimension (rows get added)
```

```
167     v_expect = mean(v,3); %mean along the third dimension (2D matrixes with row
            and col get added)
168
169     e_expect_an = mean(e_an,1); %mean along the 1 dimension (rows get added)
170     v_expect_an = mean(v_an,3); %mean along the third dimension (2D matrixes
            with row and col get added)
171
172
173     v_plot = log(abs(v_expect)./abs(v_expect(:,1)));
174     e_plot = log(e_expect.^2/e_expect(1).^2);
175
176     v_plot_an = log(abs(v_expect_an)./abs(v_expect_an(:,1)));
177     e_plot_an = log(e_expect_an.^2/e_expect_an(1).^2);
178
179
180     n = 0:size(X,2)-1;
181
182     figure
183         plot(n,v_plot)
184         hold on
185         plot(n,v_plot_an)
186         title(['(B) GD misalignment P=' num2str(P)])
187         legend('v_1[n]', 'v_2[n]', 'v_3[n]', 'v_4[n]', 'v_5[n]', 'v_6[n]', ...
188                'v_1[n] analytical', 'v_2[n] analytical', 'v_3[n] analytical', '
                    v_4[n] analytical', 'v_5[n] analytical', 'v_6[n] analytical')
189         xlabel('n')
190         grid on
191
192         saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')
193
194
195     figure
196         plot(n,e_plot)
197         hold on
198         plot(n,e_plot_an)
199         title(['(B) GD error P=' num2str(P)])
200         legend('e[n] estimate', 'e[n] analytical')
201         xlabel('n')
202         grid on
203
204         saveas(gcf,['Figures/' own_strrep(gca)], 'epsc')
205
206
207
208 end
209
210
211 %Seen from plots: with increasing window size, the gradient descend
212 %convergers better and better, the misalignment gets lower
213
214
215 % ------------------------------------------------------------------------
216 % Task c)
217
218
219 mu = 0.001;
220 alpha = 0;
221 OPTS = 0;
```

```
222
223 N_list = [10, 100];
224 % N_list = [];
225
226 for N = N_list
227
228     %reset
229     v_LMS = [];
230     v_GD = [];
231
232     %random initialization for c0
233     c0 = rand(N,1);
234
235     A = 1;
236     theta = pi/2;
237     sigma_u = sqrt(4);
238
239     rxx_ref = @(k) A^2/2 * cos(theta*k) + sigma_u^2 * kroneckerDelta(sym(k));
240     Rxx_ref = double(toeplitz(rxx_ref(0:N-1))); %double convertes the sym data
            to double precision numbers
241
242     h_calc = [h; zeros(N-length(h),1)];
243
244     p_ref = Rxx_ref * h_calc; %c_MSE = h = Rxx^-1 * p
245
246     for r = 1:size(X,1) %r...realization index
247         x = X(r,:);
248         d = D(r,:);
249
250         [~,e_LMS(r,:),c_LMS] = lms_algorithm(x,d,N,mu,alpha,OPTS,c0); %compute e
                for each realization
251         [~,e_GD(r,:),c_GD] = gd_algorithm(x,d,N,mu,Rxx_ref,p_ref);
252
253         v_LMS(:,:,r) = c_LMS - h_calc; %compute v for each realization
254         v_GD(:,:,r) = c_GD - h_calc; %compute v for each realization
255
256     end
257
258
259     %compute the expectation by averaging over different realizations
260     e_expect_LMS = mean(e_LMS,1); %mean along the 1 dimension (rows get added)
261     v_expect_LMS = mean(v_LMS,3); %mean along the third dimension (2D matrixes
            with row and col get added)
262
263     e_expect_GD = mean(e_GD,1); %mean along the 1 dimension (rows get added)
264     v_expect_GD = mean(v_GD,3); %mean along the third dimension (2D matrixes
            with row and col get added)
265
266
267     v_plot_LMS = log(abs(v_expect_LMS)./abs(v_expect_LMS(:,1)));
268     e_plot_LMS = log(e_expect_LMS.^2/e_expect_LMS(1).^2);
269
270     v_plot_GD = log(abs(v_expect_GD)./abs(v_expect_GD(:,1)));
271     e_plot_GD = log(e_expect_GD.^2/e_expect_GD(1).^2);
272
273
274     n = 0:size(X,2)-1;
275
```

```
    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) = [];

```

```
52 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) = [];

```

```
52 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

```

```
54      %change mu depending on chosen OPTS (standard or normalized LMS)
55      if OPTS == 1 %normalized LMS
56          mu_calc = mu/(alpha + x_tap'*x_tap); %only the energy of the observed
                current signal
57      end
58
59      c(:,n) = c(:,n-1) + mu_calc*conj(e(n))*x_tap;
60
61  end
62
63  %now delete the first entries of y,e and c which are zero, to keep the time
64  %indices in order
65  y(1:N-1) = [];
66  e(1:N-1) = [];
67  c(:,1:N-1) = [];
68
69  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
```

1 Problem 2.6 - Bonus Wiener-Hopf Solution

(a)

$$
\begin{aligned}
J_{MSE(\underline{c})} &= E[|e[n]|^2] = E[e[n]\ e[n]^*] \\
J_{MSE(\underline{c})} &= E[(y[n]-d[n])\ (y[n]^*-d[n]^*) \\
J_{MSE(\underline{c})} &= E[y[n]\ y[n]^* - d[n]\ y[n]^* - d[n]^*\ y[n] + d[n]\ d[n]^*] \\
J_{MSE(\underline{c})} &= E[\underline{c}^H\ \underline{x}[n]\ (\underline{c}^T\ \underline{x}[n]^*)^T - d[n]\ (\underline{c}^T\ \underline{x}[n]^*)^T - d[n]^*\ (\underline{c}^H\ \underline{x}[n])^T + d[n]\ d[n]^*\ ] \\
J_{MSE(\underline{c})} &= E[\underline{c}^H\ \underbrace{\underline{x}[n]\ \underline{x}[n]^H}_{R_{xx}}\ \underline{c}] - E[\underbrace{d[n]\ \underline{x}[n]^H}_{p^H}\ \underline{c}] - E[\underbrace{d[n]^*\ \underline{x}[n]^T}_{p^T}\ \underline{c}^*] + E[\underbrace{d[n]\ d[n]^*}_{\sigma_d^2}] \\
J_{MSE(\underline{c})} &= \underline{c}^H\ \underline{R}_{xx}\underline{c} - \underline{p}^H\ \underline{c} - \underline{p}^T\ \underline{c}^* + \sigma_d^2
\end{aligned}
$$

Minimize $\rightarrow$ $derivate(\underline{\nabla}_{\underline{c}})$

$$
\underline{\nabla}_{\underline{c}} = \begin{pmatrix} \frac{\partial}{\partial\ c_1} \\ \vdots \\ \frac{\partial}{\partial\ c_k} \\ \vdots \\ \frac{\partial}{\partial\ c_N} \end{pmatrix} \ with\ \frac{\partial}{\partial\ c_k} = \frac{\partial}{\partial\ a_k} + j\ \frac{\partial}{\partial\ b_k} \text{ for complex } c_k
$$

$$
\underline{\nabla}_{\underline{c}}\ J_{MSE(\underline{c})} = \underline{\nabla}_{\underline{c}}[\underline{c}^H\ \underline{R}_{xx}\underline{c} - \underline{p}^H\ \underline{c} - \underline{p}^T\ \underline{c}^* + \underbrace{\sigma_d^2}_{=0}]
$$

$$
\underline{\nabla}_{\underline{c}}[\underline{p}^H\ \underline{c}]^T = \underline{\nabla}_{\underline{c}}[\underline{c}^T\ \underline{p}^*] = \underline{\nabla}_{\underline{c}}[\sum_{k=0}^{N-1} c_k\ p_k^*]
$$

$$
\underline{\nabla}_{\underline{c}}[\underline{p}^H\ \underline{c}]^T = \underline{\nabla}_{\underline{c}}\sum_k (a_k + j\ b_k)\ p_k*
$$

$$
\underline{\nabla}_{\underline{c}}[\underline{p}^H\ \underline{c}]^T = \begin{pmatrix} \frac{\partial\ \sum_k (a_k+j\ b_k)\ p_k*}{\partial\ a_0} + j\ \frac{\partial\ \sum_k (a_k+j\ b_k)\ p_k*}{\partial\ b_0} \\ \vdots \\ \vdots \\ \frac{\partial\ \sum_k (a_k+j\ b_k)\ p_k*}{\partial\ a_{N-1}} + j\ \frac{\partial\ \sum_k (a_k+j\ b_k)\ p_k*}{\partial\ b_{N-1}} \end{pmatrix}
$$

$$
\underline{\nabla}_{\underline{c}}[\underline{p}^H\ \underline{c}]^T = \begin{pmatrix} p_0^* + j\ (j\ p_0^*) \\ \vdots \\ \vdots \\ p_{N-1}^* + j\ (j\ p_{N-1}^*) \end{pmatrix}
$$

$$
\underline{\nabla}_{\underline{c}}[\underline{p}^H\ \underline{c}]^T = \begin{pmatrix} p_0^* - p_0^* \\ \vdots \\ \vdots \\ p_{N-1}^* - p_{N-1}^* \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} = \underline{0}
$$

$$\underline{\nabla}_{\underline{c}}[\underline{p}^T\ \underline{c}*]^T = \underline{\nabla}_{\underline{c}}[\underline{c}^H\ \underline{p}] = \underline{\nabla}_{\underline{c}}[\sum_{k=0}^{N-1} c_k^*\ p_k]$$

$$\underline{\nabla}_{\underline{c}}[\underline{p}^T\ \underline{c}*]^T = \underline{\nabla}_{\underline{c}}[\sum_{k}(a_k - j\ b_k)\ p_k]$$

$$\underline{\nabla}_{\underline{c}}[\underline{p}^T\ \underline{c}*]^T = \begin{pmatrix} \frac{\partial \sum_k (a_k - j\ b_k)\ p_k}{\partial\ a_0} + j\ \frac{\partial \sum_k (a_k - j\ b_k)\ p_k}{\partial\ b_0} \\ \vdots \\ \vdots \\ \frac{\partial \sum_k (a_k - j\ b_k)\ p_k}{\partial\ a_{N-1}} + j\ \frac{\partial \sum_k (a_k - j\ b_k)\ p_k}{\partial\ b_{N-1}} \end{pmatrix}$$

$$\underline{\nabla}_{\underline{c}}[\underline{p}^T\ \underline{c}*]^T = \begin{pmatrix} p_0 - j\ (j\ p_0) \\ \vdots \\ \vdots \\ p_{N-1} - j\ (j\ p_{N-1}) \end{pmatrix}$$

$$\underline{\nabla}_{\underline{c}}[\underline{p}^T\ \underline{c}*]^T = \begin{pmatrix} p_0 + p_0 \\ \vdots \\ \vdots \\ p_{N-1} + p_{N-1} \end{pmatrix} = 2\ \underline{p}$$

51.1

$$\nabla_{\underline{c}}[\underline{c}^H \underline{R}_{xx} \underline{c}]$$

$$\underline{R}_{xx}^T = \underline{R}_{xx} = \begin{pmatrix} r_{00} & r_{01} & \dots & r_{0i} \\ r_{10} & r_{11} & & \\ \vdots & & \ddots & \\ r_{j0} & & & r_{ji} \end{pmatrix}$$

$$\begin{aligned} \underline{c}^H \underline{R}_{xx} \underline{c} &= \sum_{j=0}^{N-1} c_j^* \sum_{i=0}^{N-1} r_{ji}\, c_i \\ &= \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} c_j^*\, r_{ji}\, c_i \\ &= \sum_{i=0}^{N-1} |c_i|^2\, r_{ii} + \sum_{j=0}^{N-1} \sum_{i=0;i\neq j}^{N-1} c_j^*\, r_{ji}\, c_i \end{aligned}$$

$$\frac{\partial}{\partial\, c_k}\,[\underline{c}^H \underline{R}_{xx} \underline{c}] = \frac{\partial}{\partial\, c_k}\,(\sum_{i=0}^{N-1} |c_i|^2\, r_{ii} + \sum_{j=0}^{N-1} \sum_{i=0;i\neq j}^{N-1} c_j^*\, r_{ji}\, c_i)$$

$$\begin{aligned} |c_i| &= (a_i + j\, b_i)\,(a_i - j\, b_i) = a_i^2 + b_i^2 \\ \frac{\partial}{\partial\, c_k} &= \frac{\partial}{\partial\, a_k} + j\, \frac{\partial}{\partial\, b_k} \text{ for complex } c_k \\ \frac{\partial}{\partial\, c_k}(\sum_{i=0}^{N-1} |c_i|^2\, r_{ii}) &= \frac{\partial \sum_i (a_i^2 + b_i^2)\, r_{ii}}{\partial\, a_k} + j\, \frac{\partial \sum_i (a_i^2 + b_i^2)\, r_{ii}}{\partial\, b_k} \\ &= 2\, a_k\, r_{kk} + j\, 2\, b_k\, r_{kk} \end{aligned}$$

$$\begin{aligned} \frac{\partial}{\partial\, c_k}(\sum_{j=0}^{N-1} \sum_{i=0;i\neq j}^{N-1} c_j^*\, r_{ji}\, c_i) &= (\frac{\partial}{\partial\, a_k} + j\, \frac{\partial}{\partial\, b_k})(\sum_{j=0}^{N-1} \sum_{i=0;i\neq j}^{N-1} r_{ji}\,(a_i + j\, b_i)\,(a_j - j\, b_j)) \\ &= \sum_{j=0}^{N-1} \sum_{i=0;i\neq j}^{N-1} r_{jk}\,(1+0)\,(a_j - j\, b_j) + j\, r_{jk}\,(0+j)(a_j - j\, b_j) \\ &\qquad + r_{ki}\,(1-0)\,(a_i + j\, b_i) + j\, r_{ki}\,(0-j)(a_i + j\, b_i) \\ &= \sum_{i=0}^{N-1} r_{ik}\,(a_i - \cancel{j\, b_i}) + r_{ik}\, j^2\,(\cancel{a_i} - j\, b_i) + r_{ik}\,(a_i + \cancel{j\, b_i}) - j^2\,(\cancel{a_i} + j\, b_i) r_{ik} \\ &= \sum_{i=0}^{N-1} r_{ik}\, a_i + r_{ik}\, j\, b_i + r_{ik}\, a_i + r_{ik}\, j\, b_i \\ &= \sum_{i=0}^{N-1} 2\, r_{ik}\, a_i + 2\, j\, r_{ik}\, b_i \\ &= \sum_{i=0}^{N-1} 2\, r_{ik}(a_i + j\, b_i) = \sum_{i=0}^{N-1} 2\, r_{ik} c_i = 2\, \underline{R}_{xx}\, \underline{c} \end{aligned}$$

$$\nabla_{\underline{c}}[\underline{c}^H \underline{R}_{xx} \underline{c}] = 2\, \underline{R}_{xx}\, \underline{c}$$

$$\begin{aligned} \nabla_{\underline{c}} J_{MSE(\underline{c})} &= 2\, \underline{R}_{xx}\, \underline{c} - 0 - 2\, \underline{p} + 0 \\ &= \cancel{2}\, \underline{R}_{xx}\, \underline{c} - \cancel{2} \underline{p} \stackrel{!}{=} 0 \\ \Rightarrow\ \underline{c}_{MSE} &= \underline{R}_{xx}^{-1}\, \underline{p} \end{aligned}$$

(b) J_{min}

$$
\begin{aligned}
J_{min} = J_{MSE}(\underline{c}_{MSE}) &= \underline{c}_{MSE}^{H}\ \underline{R}_{xx}\ \underline{c}_{MSE} - \underline{p}^{H}\ \underline{c}_{MSE} - \underline{p}^{T}\ \underline{c}_{MSE}^{*} + \sigma_d^2 \\
&= (\underline{R}_{xx}^{-1}\ \underline{p})^{H}(\underbrace{\underline{R}_{xx}\underline{R}_{xx}^{-1}}_{\underline{I}})\ \underline{p} - \underline{p}^{H}\ \underline{R}_{xx}^{-1}\ \underline{p} - \underline{p}^{T}\ \underline{R}_{xx}^{-1}\ \underline{p} + \sigma_d^2 \\
&= \cancel{(\underline{R}_{xx}^{-1}\ \underline{p})^{H}}\ \underline{p} - \underline{p}^{H}\ \underline{R}_{xx}^{-1}\ \underline{p} - \cancel{(\underline{R}_{xx}^{-1}\ \underline{p})^{H}}\ \underline{p} + \sigma_d^2 \\
&= -\underline{p}^{H}\ \underline{R}_{xx}^{-1}\ \underline{p} + E[|d[n]|2] \\
&= -\underline{p}^{H}\ \underline{R}_{xx}^{-1}\ \underline{p} + E[(\underline{h}^{H}\ \underline{x}[n] + w[n])\ (\underline{h}^{H}\ \underline{x}[n] + w[n])^{*}] \\
&= -\underline{p}^{H}\ \underline{R}_{xx}^{-1}\ \underline{p} + E[\underline{h}^{H}\ \underline{x}[n]\ \underline{x}[n]^{H}\ \underline{h} + \underline{h}^{H}\ \underline{x}[n]\ w[n]^{*} + \underline{h}^{T}\ \underline{x}[n]^{*}\ w[n] + w[n]\ w[n]^{*}] \\
&= -\underline{p}^{H}\ \underline{R}_{xx}^{-1}\underline{p} + \underline{h}^{H}\ \underline{R}_{xx}\ \underline{h} + \sigma_w^2 \\
&= -\underline{p}^{H}\ \underline{R}_{xx}^{-1}\underline{p} + (\underline{R}_{xx}^{-1}\underline{p})^{H}\ \underbrace{\underline{R}_{xx}\ \underline{R}_{xx}^{-1}}_{\underline{I}}\underline{p} + \sigma_w^2 \\
&= -\cancel{\underline{p}^{H}\ \underline{R}_{xx}^{-1}}\ \underline{p} + \cancel{\underline{p}^{H}\ \underline{R}_{xx}^{-1}}\ \underline{p} + \sigma_w^2
\end{aligned}
$$

$$J_{min} = \sigma_w^2$$

APPENDIX

For completeness reasons the handwritten solutions of the analytical problems follow, although they were also typed in LaTeX before.

Analytical Problem 2.1

↳ problem class problem 1.14

a) $J(\underline{c}[n]) = E\{e^2[n]\} + \alpha \cdot \|\underline{c}[n]\|^2$ / from problem class sheet page 8; 1.7

$= E\{|d[n]|^2\} - 2(\underline{p}^H \cdot \underline{c}) + \underline{c}^H \cdot \underline{R}_{xx} \cdot \underline{c} + \alpha \cdot \|\underline{c}[n]\|^2$ / $\underline{c}^T \cdot \underline{c}$

$\nabla_{\underline{c}} J_{MSE}(\underline{c}[n]) = 0 - 2 \cdot \underline{p}^* + 2 \cdot \underline{R}_{xx} \cdot \underline{c}[n] + \alpha \cdot 2 \cdot \underline{c}[n]$ // assumption: $\underline{c} \in \mathbb{R}$, $\underline{p} \in \mathbb{R}$

$= \underline{2 \cdot (\underline{R}_{xx} \cdot \underline{c}[n] - \underline{p}) + 2 \cdot \alpha \cdot \underline{c}[n]}$

Gradient search: $-\nabla_{\underline{c}} J_{MSE}(\underline{c})\big|_{\underline{c}=\underline{c}[n-1]}$

$-\nabla_{\underline{c}} J_{MSE}(\underline{c})\big|_{\underline{c}=\underline{c}[n-1]} = -2 \cdot (\underline{R}_{xx} \cdot \underline{c}[n-1] - \underline{p}) - 2 \cdot \alpha \cdot \underline{c}[n-1]$ =: z

Update rule: $\underline{c}[n] = \underline{c}[n-1] + \tilde{\mu} \cdot ((\underline{p} - \underline{R}_{xx} \cdot \underline{c}[n-1]) - \alpha \cdot \underline{c}[n-1]) \cdot 2$ // $\mu = \tilde{\mu} \cdot 2$

$\underline{c}[n] = \underline{c}[n-1] - \mu \cdot \alpha \cdot c[n-1] + \mu \cdot (\underline{p} - \underline{R}_{xx} \cdot \underline{c}[n-1])$

$$\boxed{\underline{c}[n] = \underline{c}[n-1] \cdot (1 - \mu \cdot \alpha) + \mu \cdot (\underline{p} \cdot \underline{R}_{xx} \cdot \underline{c}[n-1])}$$

b) Where does it converge to?

$n \to \infty \Rightarrow c_\infty = ?$

$\underline{c}[n] = \underline{c}[n-1] \cdot (1 - \mu \cdot \alpha) + \mu \cdot (\underline{p} - \underline{R}_{xx} \cdot \underline{c}[n-1])$ // $n \to \infty$; $n-1 \to \infty$

$\underline{c}_\infty = \underline{c}_\infty \cdot (1 - \mu \cdot \alpha) + \mu \cdot (\underline{p} - \underline{R}_{xx} \cdot \underline{c}_\infty)$

$\underline{c}_\infty \cdot (1 - 1 + \mu \cdot \alpha) = \mu \cdot (\underline{p} - \underline{R}_{xx} \cdot \underline{c}_\infty)$

$\underline{c}_\infty \cdot \mu \cdot \alpha = \mu \cdot (\underline{p} - \underline{R}_{xx} \cdot \underline{c}_\infty)$

$\underline{I} \cdot \underline{c}_\infty \cdot \alpha = \underline{p} - \underline{R}_{xx} \cdot \underline{c}_\infty$

$(\underline{R}_{xx} + \underline{I}\alpha) \cdot \underline{c}_\infty = \underline{p}$

$$\boxed{\underline{c}_\infty = (\underline{R}_{xx} + \underline{I}\alpha)^{-1} \cdot \underline{p}}$$ // not quiet the Wiener solution $\underline{c}_{opt}$

c) misalignment $\underline{v}[n]$

$\underline{v}[n] = \underline{c}[n] - \underline{c}_\infty$

$p = (\underline{R}_{xx} + \alpha) \cdot \underline{c}_\infty$

$\underline{c}[n] = (1-\mu\alpha)\cdot \underline{c}[n-1] + \mu\,(\underline{p} - \underline{R}_{xx}\cdot \underline{c}[n-1]) \quad | -\underline{c}_\infty$

$\underline{c}[n] - \underline{c}_\infty = (1-\mu\cdot\alpha)\cdot \underline{c}[n-1] - \underline{c}_\infty + (1-\mu\cdot\alpha)\cdot \underline{c}_\infty + \mu\,(\underline{p} - \underline{R}_{xx}\cdot \underline{c}[n-1]) \quad \underline{p} = \underline{R}_{xx}$

$\underline{v}[n] = (1-\mu\cdot\alpha)\cdot \underline{v}[n-1] - \underline{c}_\infty + \underline{c}_\infty - \mu\alpha\cdot\underline{c}_\infty + \mu\cdot(\underline{R}_{xx}\cdot\underline{c}_\infty + \alpha\,\underline{c}_\infty - \underline{R}_{xx}\cdot\underline{c}[n-1])$

$\underline{v}[n] = (1-\mu\cdot\alpha)\cdot \underline{v}[n-1] - \mu\alpha\,\underline{c}_\infty + \mu\alpha\,\underline{c}_\infty + \mu\cdot\underline{R}_{xx}\cdot(-\underline{v}[n-1])$

$\underline{v}[n] = (1-\mu\cdot\alpha)\cdot \underline{v}[n-1] - \mu\cdot\underline{R}_{xx}\,\underline{v}[n-1]$

$$\boxed{\underline{v}[n] = \left((1-\mu\cdot\alpha)\cdot\underline{I} - \mu\cdot\underline{R}_{xx}\right)\cdot\underline{v}[n-1]} \quad // \underline{v}[n] = \left[(1-\mu\cdot\alpha)\cdot\underline{I} - \mu\cdot\underline{R}_{xx}\right]^n\cdot\underline{v}[0]$$

d) decoupling

$\underline{v}[n] = \left[(1-\mu\cdot\alpha)\cdot\underline{I} - \mu\cdot\underline{R}_{xx}\right]\cdot\underline{v}[n-1] =$

Eigendecomposition: $\underline{R}_{xx} = \underline{Q}\cdot\underline{\Lambda}\cdot\underline{Q}^H$

$\underline{Q}$ ... unitary matrix; $\underline{Q}^{-1} = \underline{Q}^H$ // conjugate matrix, NICHT conjugierte "$\underline{v}_0$"

$\underline{\Lambda}$ ... diagonal matrix consisting of eigenvalues; $\underline{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \dots, \lambda_{M-1})$

$\|\underline{Q}^H\cdot\underline{a}\|_2^2 = \underline{a}^H\cdot\underline{Q}\cdot\underline{Q}^H\cdot\underline{a} = \underline{a}^H\cdot\underline{I}\cdot\underline{a} = \|\underline{a}\|_2^2$ // $\underline{Q}$ matrix does not scale vector, it only rotates it

$\underline{v}[n] = \left[(1-\mu\cdot\alpha)\cdot\underline{I} - \mu\cdot\underline{Q}\cdot\underline{\Lambda}\,\underline{Q}^H\right]\cdot\underline{v}[n-1] \quad | \underline{Q}^H$ from left

$\underbrace{\underline{Q}^H\underline{v}[n]}_{\tilde{\underline{v}}[n]} = (1-\mu\cdot\alpha)\cdot\underline{Q}^H\cdot\underline{v}[n-1] - \underline{Q}^H\cdot\mu\cdot\underline{Q}\cdot\underline{\Lambda}\cdot\underline{Q}^H\cdot\underline{v}[n-1]$

$\tilde{\underline{v}}[n] = (1-\mu\cdot\alpha)\cdot\tilde{\underline{v}}[n-1] - \mu\cdot\underbrace{\underline{Q}^H\cdot\underline{Q}}_{\underline{I}}\cdot\underline{\Lambda}\cdot\tilde{\underline{v}}[n-1]$

$$\boxed{\tilde{\underline{v}}[n] = (1-\mu\cdot\alpha)\cdot\tilde{\underline{v}}[n-1] - \mu\cdot\underbrace{\underline{\Lambda}}_{\text{diagonal matrix}}\tilde{\underline{v}}[n-1] \quad // \text{decoupled}} \qquad \tilde{v}_1[n] = (1-\mu\cdot\alpha)\cdot\tilde{v}_1[n-1] - \mu\cdot\lambda_1\cdot\tilde{v}_1[n-1]$$

e) $\tilde{\underline{V}}[n] = [(1-\mu\cdot a)\cdot \underline{I} - \mu\cdot \underline{\Lambda}]\cdot \tilde{\underline{V}}[n-1] = [($

$= [(1-\mu\cdot a)\cdot \underline{I} - \mu\cdot \underline{\Lambda}]\cdot[(1-\mu\cdot a)\cdot \underline{I} - \mu\cdot \underline{\Lambda}]\cdot \tilde{\underline{V}}[n-2]$

$$\boxed{\tilde{\underline{V}}[n] = [(1-\mu\cdot a)\cdot \underline{I} - \mu\cdot \underline{\Lambda}]^n \cdot \tilde{\underline{V}}[0]}$$

f) assumption: a influences convergence time // What is meant by "find an expression of the time constant τ_i"?

$$\tilde{V}_i[n] = [(1-\mu\cdot a) - \mu\cdot\lambda_i]^n \cdot \tilde{V}_i[0]$$

$$|\tilde{V}_i[n]| = \underbrace{|(1-\mu\cdot a) - \mu\cdot\lambda_i|^n}_{\text{exponential decay: } |e^{-\frac{n}{\tau_i}}|} \cdot |\tilde{V}_i[0]| \quad // \tilde{V}_i[n] \overset{!}{=} 0 \text{ for } n\to\infty$$

$$\hookrightarrow |(1-\mu\cdot a) - \mu\cdot\lambda_i|^n = |e^{-\frac{n}{\tau_i}}| \quad // e>0\ \forall n$$

$$e^{-\frac{n}{\tau_i}} = |(1-\mu\cdot a) - \mu\cdot\lambda_i|^n \quad |\ln$$

$$-\frac{n}{\tau_i}\cdot \ln(e) = n\cdot \ln|(1-\mu\cdot a) - \mu\cdot\lambda_i|$$

$$\tau_i = -\frac{1}{\ln|(1-\mu\cdot a) - \mu\cdot\lambda_i|}$$

// convergence time can be easily changed w/o changing the step size (or eigenvalues)

if λ_i are different $\Rightarrow \tau_i$ is different for each component

Problem 2.2

a)

$$E\{\hat{\mu}[P]\} = E\left\{\frac{1}{P}\cdot\sum_{n=0}^{P-1} x[n]\right\} = E\left\{\frac{1}{P}\cdot\underbrace{[x[0]+x[1]+\dots+x[P-1]]}_{x[n]+x[n+1]+\dots+x[P-1]}\right\} =$$

$$= \frac{1}{P}\cdot\left[E\{x[0]\}+E\{x[1]\}+\dots+E\{x[P-1]\}\right] =$$

$$= \frac{1}{P}\cdot\sum_{n=0}^{P-1} E\{x[n]\} = \frac{1}{P}\cdot\sum_{n=0}^{P-1}\mu_x = \frac{1}{P}\cdot P\cdot\mu_x$$

$$= \underline{\underline{\mu_x}}$$

$$E\{\hat{r}_{xy}[k,P]\} = E\left\{\frac{1}{P}\cdot\sum_{n=0}^{P-1} x[n+k]\cdot y[n]\right\} =$$

$$= \frac{1}{P}\cdot\sum_{n=0}^{P-1} E\{x[n+k]\cdot y[n]\} =$$

$$= \frac{1}{P}\cdot\sum_{n=0}^{P-1} r_{xy}[k] = \frac{1}{P}\cdot P\cdot r_{xy}[k]$$

$$= \underline{\underline{r_{xy}[k]}}$$

2.6 Analytical Problem

$$J_{MSE}(\underline{c}) = E\{|e[n]|^2\} = E\{e[n]\cdot e^*[n]\} = \quad // e[n] = y[n] - d[n]\ ;\ (a-b)^* = a^* - b^*$$

$$= E\{(y[n] - d[n])\cdot(y^*[n] - d^*[n])\} =$$

$$(a\cdot b)^* = a^*\cdot b^*$$

$$= E\{y[n]\,y^*[n] - d[n]\,y^*[n] - d^*[n]\,y[n] + d[n]\,d^*[n]\} \quad // y[n] = \underline{c}^H\cdot\underline{x}[n]\ ;\ \text{don't sub } d[n]$$

$$= E\{\underline{c}^H\cdot\underline{x}[n]\cdot(\underline{c}^T\cdot\underline{x}^*[n])^T - d[n]\cdot(\underline{c}^T\cdot\underline{x}^*[n])^T - d^*[n]\cdot(\underline{c}^H\cdot\underline{x}[n])^T + d[n]\,d^*[n]\} =$$

$$= E\{\underline{c}^H\cdot\underline{x}[n]\cdot\underline{x}^H[n]\cdot\underline{c}\} - E\{d[n]\cdot\underline{x}^H[n]\cdot\underline{c}\} - E\{d^*[n]\cdot\underline{x}^T[n]\cdot\underline{c}^*\} + E\{d[n]\,d^*[n]\} =$$

$$= \underline{c}^H\cdot\underline{R}_{xx}\cdot\underline{c} - \underline{p}^H\cdot\underline{c} - \underline{p}^T\cdot\underline{c}^* + \sigma_d^2$$

Minimize → derivate $(\nabla_{\underline{c}})$ $\qquad \nabla_{\underline{c}} = \begin{bmatrix} \frac{\partial}{\partial c_0} \\ \frac{\partial}{\partial c_1} \\ \vdots \\ \frac{\partial}{\partial c_{N-1}} \end{bmatrix}$ with $\frac{\partial}{\partial c_k} = \frac{\partial}{\partial a_k} + j\cdot\frac{\partial}{\partial b_k}$ for complex c_k

$$\nabla_{\underline{c}}\, J_{MSE}(\underline{c}) = \nabla_{\underline{c}}\left[\underline{c}^H\cdot\underline{R}_{xx}\cdot\underline{c} - \underline{p}^H\cdot\underline{c} - \underline{p}^T\cdot\underline{c}^* + \sigma_d^2\right]$$

(scalar; not necessary; σ_d^2 → 0)

↳ $\nabla_{\underline{c}}[\underline{p}^H\cdot\underline{c}]^T = \nabla_{\underline{c}}[c^T\cdot p^*] = \nabla_{\underline{c}}\left[\sum_{k=0}^{N-1} c_k\cdot p_k^*\right] =$

$$= \nabla_{\underline{c}}\sum_k (a_k + j\,b_k)\cdot p_k^* =$$

↳ $\nabla_{\underline{c}} = \begin{bmatrix} \frac{\partial}{\partial a_0} + j\cdot\frac{\partial}{\partial b_0} \\ \vdots \\ \frac{\partial}{\partial a_{N-1}} + j\cdot\frac{\partial}{\partial b_{N-1}} \end{bmatrix}$

$$= \begin{bmatrix} \left(\frac{\partial}{\partial a_0} + j\cdot\frac{\partial}{\partial b_0}\right)\cdot\sum_k (a_k + j\,b_k)\cdot p_k^* \\ \vdots \\ \left(\frac{\partial}{\partial a_{N-1}} + j\cdot\frac{\partial}{\partial b_{N-1}}\right)\cdot\sum_k (a_k + j\,b_k)\cdot p_k^* \end{bmatrix} = \begin{bmatrix} \frac{\partial\sum_k (a_k + j\,b_k)\cdot p_k^*}{\partial a_0} + j\cdot\frac{\partial\sum_k (a_k + j\,b_k)\cdot p_k^*}{\partial b_0} \\ \vdots \\ \frac{\partial\sum_k (a_k + j\,b_k)\cdot p_k^*}{\partial a_{N-1}} + j\cdot\frac{\partial\sum_k (a_k + j\,b_k)\cdot p_k^*}{\partial b_{N-1}} \end{bmatrix} =$$

$$= \begin{bmatrix} p_0^* + j\cdot(j\cdot p_0^*) \\ \vdots \\ p_{N-1}^* + j\cdot(j\cdot p_{N-1}^*) \end{bmatrix} = \begin{bmatrix} p_0^* - 1\cdot p_0^* \\ \vdots \\ p_{N-1} - 1\cdot p_{N-1} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\boxed{\nabla_{\underline{c}}[\underline{p}^H\cdot\underline{c}] = \underline{0}}$$

$\hookrightarrow \underline{\nabla}_{\underline{c}}[\overbrace{\underline{p}^T \cdot \underline{c}^*}^{\text{scalar}}]^T = \underline{\nabla}_{\underline{c}}[\underline{c}^H \cdot \underline{p}] = \underline{\nabla}_{\underline{c}}[\sum_{k=0}^{N-1} c_k^* \cdot p_k]$ // $c_k^* = a_k - j \cdot b_k$

$$= \underline{\nabla}_{\underline{c}}[\sum_{k=0}^{N-1} (a_k - j \cdot b_k) \cdot p_k]$$

$$= \begin{bmatrix} \left(\frac{\partial}{\partial a_0} + j \cdot \frac{\partial}{\partial b_0}\right) \cdot \sum_k (a_k - j \cdot b_k) \cdot p_k \\ \vdots \\ \left(\frac{\partial}{\partial a_{N-1}} + j \cdot \frac{\partial}{\partial b_{N-1}}\right) \cdot \sum_k (a_k - j \cdot b_k) \cdot p_k \end{bmatrix} = \begin{bmatrix} p_0 + j \cdot (-j \cdot p_0) \\ \vdots \\ p_{N-1} + j \cdot (-j \cdot p_{N-1}) \end{bmatrix} =$$

$$= \begin{bmatrix} p_0 + p_0 \\ \vdots \\ p_{N-1} + p_{N-1} \end{bmatrix} = \underline{\underline{2 \cdot \underline{p}}}$$

$$\boxed{\underline{\nabla}_{\underline{c}}[\underline{p}^T \cdot \underline{c}^*] = 2 \cdot \underline{p}}$$

$\hookrightarrow \underline{\nabla}_{\underline{c}}[\underline{c}^H \cdot \underline{R}_{xx} \cdot \underline{c}]$

$$\underline{R}_{xx}^T = \underline{R}_{xx} = \begin{bmatrix} r_{00} & r_{01} & \cdots & r_{0i} \\ r_{10} & r_{11} & & \\ \vdots & & \ddots & \\ r_{j0} & & & r_{ji} \end{bmatrix}^{N-1}$$

j … row index

i … col index

$$\hookrightarrow \underline{c}^H \cdot \underline{R}_{xx} \cdot \underline{c} = \overbrace{\sum_{j=0}^{N-1} c_j^* \cdot \underbrace{\sum_{i=0}^{N-1} r_{ji} \cdot c_i}_{\text{sum rows}}}^{\text{sum cols}} =$$

$$= \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} c_j^* \cdot r_{ji} \cdot c_i = \overbrace{\sum_{i=0}^{N-1} |c_i|^2 \cdot r_{ii}}^{\text{main diagonal}} + \overbrace{\sum_{i=0}^{N-1} \sum_{\substack{j=0 \\ j \neq i}}^{N-1} r_{ji} \cdot c_i \cdot c_j^*}^{\text{everything but main diagonal}}$$

$$\frac{\partial}{\partial c_k}[\underline{c}^H \cdot \underline{R}_{xx} \cdot \underline{c}] = \frac{\partial}{\partial c_k}\left(\sum_{i=0}^{N-1} |c_i|^2 \cdot r_{ii} + \sum_{i=0}^{N-1} \sum_{\substack{j=0 \\ i \neq j}}^{N-1} r_{ji} \cdot c_i \cdot c_j^*\right.$$

$\hookrightarrow |c_i| = (a_i + j \cdot b_i) \cdot (a_i - j b_i) =$

$$= a_i^2 - j \cdot a_i b_i + j \cdot a_i \cdot b_i - j^2 \cdot b_i = \underline{\underline{a_i^2 + b_i^2}}$$

$\hookrightarrow \frac{\partial}{\partial c_k} = \frac{\partial}{\partial a_k} + j \cdot \frac{\partial}{\partial b_k}$

$\hookrightarrow \frac{\partial}{\partial c_k}\left(\sum_{i=0}^{N-1} |c_i|^2 \cdot r_{ii}\right) = \frac{\partial \sum_i (a_i^2 + b_i^2) \cdot r_{ii}}{\partial a_k} + j \cdot \frac{\partial \sum_i (a_i^2 + b_i^2) \cdot r_{ii}}{\partial b_k}$

$$\boxed{= 2 a_k \cdot r_{kk} + j \cdot 2 b_k \cdot r_{kk}}$$

↳ $\frac{2}{\partial a_k}\left(\sum_{i=0}^{N-1}\sum_{\substack{j=0\\ i\neq j}}^{N-1} r_{ji}\cdot c_i\cdot c_j^*\right) =$

$= \left(\frac{2}{\partial a_k} + j\cdot\frac{2}{\partial b_k}\right)\cdot\left(\sum_{i=0}^{N-1}\sum_{\substack{j=0\\ i\neq j}}^{N-1} r_{ji}\cdot(a_i+j\cdot b_i)\cdot(a_j-j\cdot b_j)\right) =$ // first derivate over i, then j

$= \sqrt{-1}$ and not the index

$= \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} r_{jk}\cdot(1+0)\cdot(a_j-j\cdot b_j) + j\cdot r_{jk}\cdot(0+j\cdot 1)\cdot(a_j-j\cdot b_j)$

$+ r_{ki}\cdot(a_i+j\cdot b_i)\cdot(1-0) + j\cdot r_{ki}\cdot(a_i+j\cdot b_i)\cdot(0-j\cdot 1) =$ // $r_{ki}=r_{ik}$

$= \sum_{i=0}^{N-1} r_{ik}\cdot(a_i-j\cdot b_i) + r_{ik}\cdot j^2\cdot(a_i-j\cdot b_i) + r_{ik}\cdot(a_i+j\cdot b_i) - j^2\cdot(a_i+j\cdot b_i)\cdot r_{ik} =$

$= \sum_{i=0}^{N-1} r_{ik}\cdot a_i + r_{ik}\cdot j\cdot b_i + r_{ik}\cdot a_i + r_{ik}\cdot j\cdot b_i$

$= \sum_{i=0}^{N-1} 2\cdot r_{ik}\cdot a_i + 2j\cdot r_{ik}\cdot b_i$

$= \sum_{i=0}^{N-1} 2\cdot r_{ik}\cdot(a_i+j\cdot b_i) = \sum_{i=0}^{N-1} 2\cdot r_{jk}\cdot c_i$

$= \sum_{i=0}^{N-1} 2\cdot r_{ik}\cdot c_i = \left[2\cdot \underline{R}_{xx}\cdot\underline{c}\right]_{k\text{-th row}}$ accidentally switched row and col, should hold true nonetheless

↳ $\boxed{\nabla_{\underline{c}}\left[\underline{c}^H\cdot\underline{R}_{xx}\cdot\underline{c}\right] = 2\cdot\underline{R}_{xx}\cdot\underline{c}}$

$\nabla_{\underline{c}} J_{MSE(\underline{c})} = 2\cdot\underline{R}_{xx}\cdot\underline{c} - 0 - 2\cdot\underline{p} + 0$

$= 2\cdot\underline{R}_{xx}\cdot\underline{c} - 2\underline{p} \overset{!}{=} 0$ // to find minimum

$\Rightarrow \boxed{\underline{c}_{MSE} = \underline{R}_{xx}^{-1}\cdot\underline{p}}$ // Wiener Hopf solution for complex numbers

b) J_{min}

$$J_{min} = J_{MSE}(c_{MSE}) = \underline{c}_{MSE}^H \cdot \underline{R}_x \cdot \underline{c}_{MSE} - \underline{p}^H \underline{c}_{MSE} - \underline{p}^T \cdot \underline{c}_{MSE}^* + \sigma_d^2$$

$$= (\underline{R}_x^{-1} \cdot \underline{p})^H \cdot (\underline{R}_x \cdot \underline{R}_x^{-1}) \underline{p} - \underline{p}^H \cdot \underline{R}_x^{-1} \cdot \underline{p} - (\underline{p}^T \cdot (\underline{R}_x^{-1} \cdot \underline{p}))^* + \sigma_d^2$$

($\underline{R}_x \cdot \underline{R}_x^{-1} = \underline{I}$; $(\cdot)^*$ = conj)

$$= \cancel{(\underline{R}_x^{-1} \cdot \underline{p})^H \underline{p}} - \underline{p}^H \cdot \underline{R}_x^{-1} \cdot \underline{p} - \cancel{(\underline{R}_x^{-1} \cdot \underline{p})^H \cdot \underline{p}} + \sigma_d^2$$

$$= -\underline{p}^H \cdot \underline{R}_x^{-1} \cdot \underline{p} + E\{|d[n]|^2\} = \quad // d[n] = \underline{h}^H \cdot \underline{x}[n] + w[n]$$

$$= -\underline{p}^H \cdot \underline{R}_x^{-1} \cdot \underline{p} + E\{(\underline{h}^H \cdot \underline{x}[n] + w[n]) \cdot (\underline{h}^H \cdot \underline{x}[n] + w[n])^*\}$$

$$= -\underline{p}^H \cdot \underline{R}_x^{-1} \cdot \underline{p} + E\{\underline{h}^H \underbrace{\underline{x}[n] \cdot \underline{x}[n]^H}_{E[\cdot]=R_x} \cdot \underline{h} + \underline{h}^H \underbrace{\underline{x}[n] \cdot w[n]^*}_{E[\cdot]=0} + \underline{h}^T \underbrace{\underline{x}[n]^* \cdot w[n]}_{E[\cdot]=0} + \underbrace{w[n] \cdot w[n]^*}_{E[\cdot]=\sigma_w^2}\}$$

$$= -\underline{p}^H \cdot \underline{R}_x^{-1} \cdot \underline{p} + \underline{h}^H \cdot \underline{R}_x \cdot \underline{h} + \sigma_w^2 \quad // \underline{h} = \underline{c}_{MSE} = \underline{R}_x^{-1} \cdot \underline{p}$$

$$= -\underline{p}^H \cdot \underline{R}_x^{-1} \cdot \underline{p} + (\underline{R}_x^{-1} \cdot \underline{p})^H \cdot \underbrace{\underline{R}_x \cdot \underline{R}_x^{-1}}_{\underline{I}} \cdot \underline{p} + \sigma_w^2$$

$$\hookrightarrow (\underline{R}_x^{-1} \cdot \underline{p})^H = \underline{p}^H \cdot (\underline{R}_x^{-1})^H \overset{R_x^H = R_x}{=} \underline{p}^H \cdot \underline{R}_x^{-1}$$

$$= \cancel{-\underline{p}^H \cdot \underline{R}_x^{-1} \cdot \underline{p}} + \cancel{\underline{p}^H \cdot \underline{R}_x^{-1} \cdot \underline{p}} + \sigma_w^2$$

$$\boxed{J_{min} = \sigma_w^2}$$

Index der Kommentare

10.1 You didn't answer the effects of alpha!!

With alpha you can control the stability of the filter. This is done by offsetting the eigenvalues of Rxx by alpha. You can think of this as adding additional white noise at the input with a variance of alpha, this is known as "prewhitening".

The instability of these algorithms often comes from numerical calculations. So, introducing alpha can help with that, but not always!

You get 2P out of 3P for this one.

20.1 The expectation of _c[n] converges to _h. Or in other words, the average of many different realizations (here we just used one realization) converges to _h.

20.2 Even if there is no noise, the LMS will "never truly converge".

29.1 -0.5P: You calculated the MSE incorrectly. You calculated
ln(E{e[n]}^2 / E{e[0]}^2)
instead of
ln(E{e[n]^2} / E{e[0]^2})
resulting in a more "noisy" MSE.

33.1 -0.5P: You used the same estimates for Rxx and p for all realizations, i.e. you calculated Rxx and p using the 1st realization and used this one for the adaption of all realizations.
The algorithm should be unstable for P = 10.

34.1 But only slightly.

35.1 -1P: How does this relate to the MSE?

51.1 clearly presented!