Adaptive Systems

Assignment 2

Autor: Ebner Thomas (0831246), Nöhmer Stefan (0830668)
Datum: Graz, 15. Dezember 2011
Version.: alpha 1.0

1 MATLAB Problem 2.1

In den Abbildungen 1.1 bis 1.3 sind die Filterkoeffizienten des Adaptiven Filters dargestellt. Beim rauschfreien Fall in Abbildung 1.1 erkennt man, dass der Adaptive Filter die Koeffizienten des unbekannten Systems sehr schnell annähert. Da kein Rauschen hinzugefügt wird, werden diese Koeffizienten sehr gut angehähert.
In den Abbiludungen 1.2 und 1.3 ist sehr gut der Unterschied zwischen NLMS und LMS zu erkennen.
Beim LMS nähern sich die Koeffizienten etwas schneller als beim NLMS an. Der Grund hierfür ist, dass beim NLMS μ durch die Norm des tapped input Vektors angepasst wird. Bei einer Varianz des Eingangssignals (white noise) von 1 und einer Filterordnung von 4, wird μ im Mittel durch 4 dividiert.
Durch das kleinere effektive μ nähert sich der NLMS etwas langsamer den Koeffizienten an. Allerdings ist durch das kleinere μ der Excess-Error kleiner (siehe Vorlesung vom 18.11). Somit enthalten die Koeffizienten weniger Fluktuation (weniger Rauschen). Die Abbildungen 1.2 und 1.3 spiegeln sehr gut die geschilderten zusammenhänge wider.

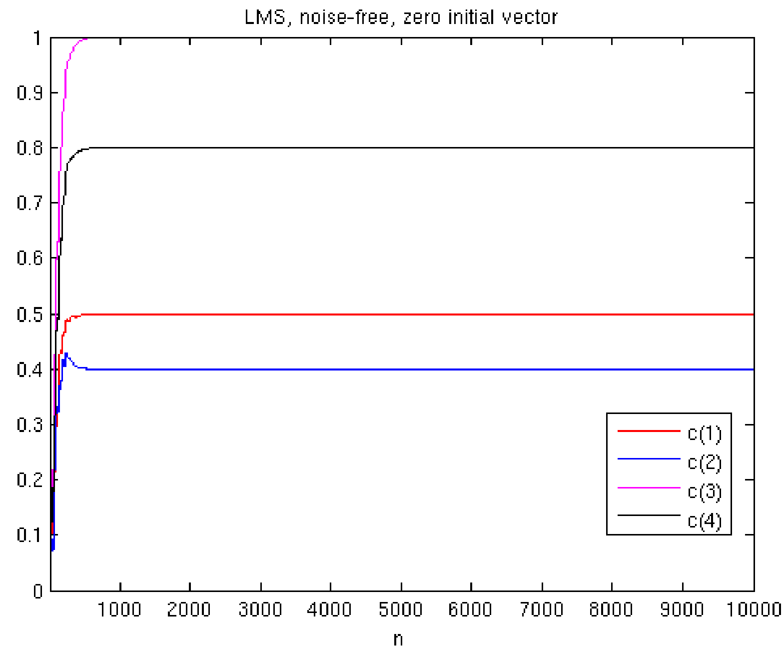

Abbildung 1.1: LMS, $\mu = 0.01$, zero-mean white noise input signal with unit variance

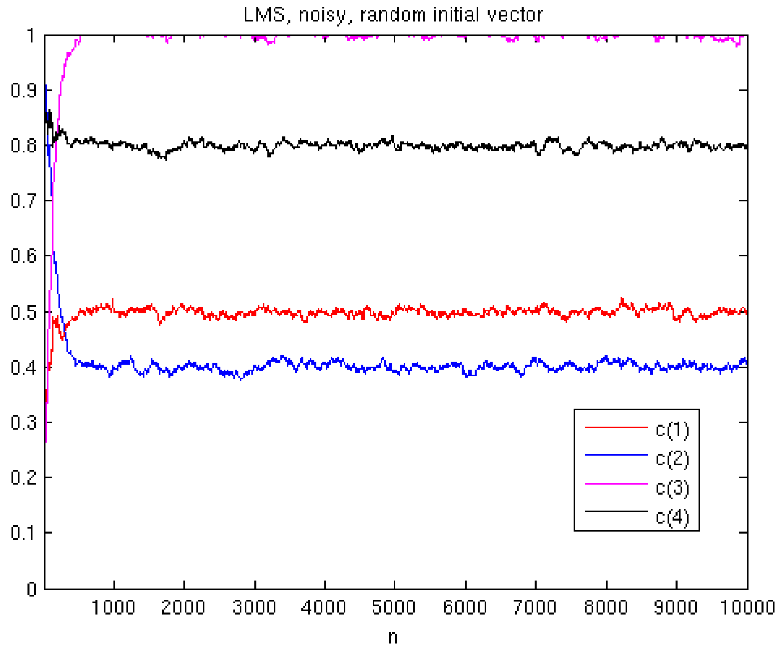

Abbildung 1.2: LMS, $\mu = 0.01$, zero-mean white noise input signal with unit variance, and additional white noise

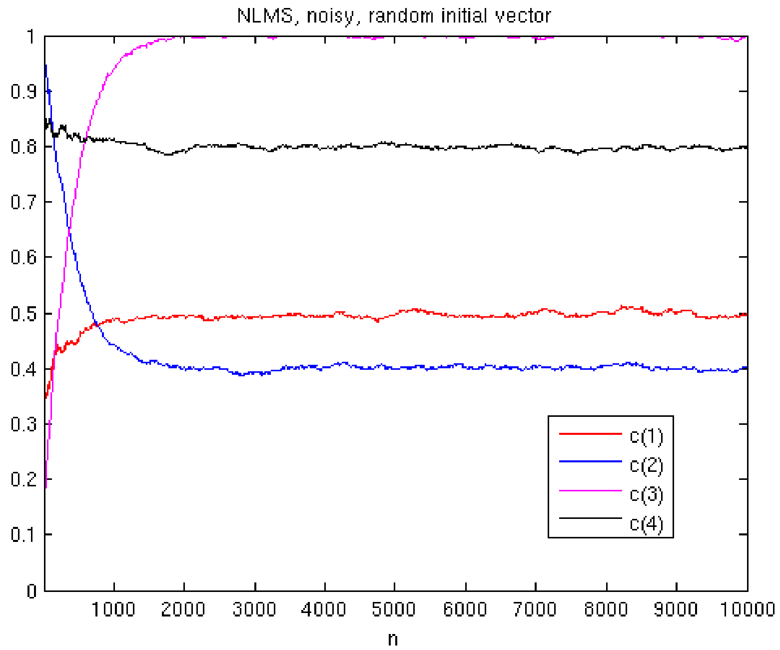

Abbildung 1.3: NLMS, $\mu = 0.01$, zero-mean white noise input signal with unit variance, and additional white noise

2 Analytic Problem 1.2

a) Wie in der Übung am 25.10.2011 hergeleitet kann die Kostenfunktion wie folgt angeschrieben werden:

$$J(\boldsymbol{c}) = \sigma_v^2 - 2\boldsymbol{c}^T\boldsymbol{p} + \boldsymbol{c}^T\boldsymbol{R}_{xx}\boldsymbol{c} \tag{2.1}$$

Die Gleichung 2.1 kann wie folgt umformuliert werden:

$$J(\boldsymbol{c}) = \sigma_v^2 - \boldsymbol{p}^T\boldsymbol{R}_{xx}^{-1}\boldsymbol{p} + (\boldsymbol{c} - \boldsymbol{R}_{xx}^{-1}\boldsymbol{p})^T\boldsymbol{R}_{xx}(\boldsymbol{c} - \boldsymbol{R}_{xx}^{-1}\boldsymbol{p}) \tag{2.2}$$

Beweis: Durch Ausmultiplizieren der Klammern gelangt man wieder auf die Gleichung 2.1:

$$\begin{aligned}
J(\boldsymbol{c}) &= \sigma_v^2 - \boldsymbol{p}^T\boldsymbol{R}_{xx}^{-1}\boldsymbol{p} + (\boldsymbol{c}^T - \boldsymbol{p}\boldsymbol{R}_{xx}^{-1})(\boldsymbol{R}_{xx}\boldsymbol{c} - \boldsymbol{R}_{xx}\boldsymbol{R}_{xx}^{-1}\boldsymbol{p}) && (2.3)\\
&= \sigma_v^2 - \boldsymbol{p}^T\boldsymbol{R}_{xx}^{-1}\boldsymbol{p} + \boldsymbol{c}^T\boldsymbol{R}_{xx}\boldsymbol{c} - \boldsymbol{p}^T\boldsymbol{R}_{xx}^{-1}\boldsymbol{R}_{xx}\boldsymbol{c} - \boldsymbol{c}^T\boldsymbol{p} + \boldsymbol{p}^T\boldsymbol{R}_{xx}^{-1}\boldsymbol{p} && (2.4)\\
&= \sigma_v^2 + \boldsymbol{c}^T\boldsymbol{R}_{xx}\boldsymbol{c} - \boldsymbol{p}^T\boldsymbol{c} - \boldsymbol{c}^T\boldsymbol{p} && (2.5)\\
&= \sigma_v^2 + \boldsymbol{c}^T\boldsymbol{R}_{xx}\boldsymbol{c} - 2\boldsymbol{c}^T\boldsymbol{p} && (2.6)
\end{aligned}$$

In der Gleichung 2.2 kommt der Ausdruck $\boldsymbol{c} - \boldsymbol{R}_{xx}^{-1}\boldsymbol{p}$ vor. Dieser Ausdruck entspricht genau dem Misalignment-Vector, da $\boldsymbol{R}_{xx}^{-1}\boldsymbol{p}$ der Wiener Hopf-Solution entspricht und somit die optimale Lösung darstellt.
Somit kann die Kostenfunktion in Abhängigkeit von $\boldsymbol{v}$ (Misalignment-Vektor) ausgedrückt werden:

$$J(\boldsymbol{c}) = \sigma_v^2 - \boldsymbol{p}^T\boldsymbol{R}_{xx}^{-1}\boldsymbol{p} + \boldsymbol{v}^T\boldsymbol{R}_{xx}\boldsymbol{v} \tag{2.7}$$

Anhand dieser Gleichung erkennt man, dass der vordere Teil ($\sigma_v^2 - \boldsymbol{p}^T\boldsymbol{R}_{xx}^{-1}\boldsymbol{p}$) unabhängig vom Misalignment-Vektor ist und somit das Minimum der Kostenfunktion(J_{min}) darstellt.

b) Die Autokorrelationsmatrix $\boldsymbol{R}_{xx}$ kann mittels der Eigenwerte/Eigenvektoren wie folgt zerlegt werden:

$$\boldsymbol{R}_{xx} = \boldsymbol{Q}\boldsymbol{\Delta}\boldsymbol{Q}^T \tag{2.8}$$

Wobei die Matrix $\boldsymbol{\Delta}$ eine Diagonalmatrix ist, welche die Eigenwerte von $\boldsymbol{R}_{xx}$ enthält. Die Matrix $\boldsymbol{Q}$ enthält alle Eigenvektoren.
Diese Beziehung kann in die Gleichung 2.7 eingesetzt werden und man erhält:

$$J(\boldsymbol{c}) = J_{min} + \boldsymbol{v}^T\boldsymbol{Q}\boldsymbol{\Delta}\boldsymbol{Q}^T\boldsymbol{v} \tag{2.9}$$

Fügt man nun noch folgende Substituion ein: $\tilde{\boldsymbol{v}} = \boldsymbol{Q}^T\boldsymbol{v}$, so erhält man eine Gleichung für die Kostenfunktion bei der die einzelnen Komponenten von $\tilde{\boldsymbol{v}}$ entkoppelt sind:

$$J(\boldsymbol{c}) = J_{min} + \tilde{\boldsymbol{v}}^T\boldsymbol{\Delta}\tilde{\boldsymbol{v}} \tag{2.10}$$

Die Gleichung 2.10 in Matrixschreibweise kann nun wie folgt in eine Summe umgeschrieben werden:

$$J(\boldsymbol{c}) = J_{min} + \sum_{k=1}^{N} \tilde{v}_k^2 \lambda_k \tag{2.11}$$

c) Wie in der Übung vom 22.11.2011 gezeigt wurde, verhalten sich die einzelnen Komponenten von $\tilde{\boldsymbol{v}}$ wie folgt:

$$\tilde{v}_k[n] = (1 - \mu\lambda_k)^n \tilde{v}_k[0] = \tilde{v}_k[0] e^{-n/\tau_k} \tag{2.12}$$

Diese Gleichung in Gleichung 2.11 eingesetzt ergibt:

$$J(\boldsymbol{c}) = J_{min} + \sum_{k=1}^{N} \tilde{v}_k^2[0] \lambda_k e^{-2n/\tau_k} \tag{2.13}$$

d) Die Zeitkonstanten können aus der Gleichung 2.12 ermittelt werden: Daraus ergibt sich für τ_k (wie auch in der Übung bereits hergeleitet) folgendes:

$$\tau_k = \frac{-1}{log|1 - \mu\lambda_k|} \approx \frac{1}{\mu\lambda_k} \tag{2.14}$$

$$J(\boldsymbol{c}) = J_{min} + \sum_{k=1}^{N} \tilde{v}_k^2[0] \lambda_k e^{-2n\mu\lambda_k} \tag{2.15}$$

e) White noise with unit variance =¿ $R_{xx} = \boldsymbol{I}$. Die Eigenwerte einer Diagonalmatrix entsprechen genau den Werten in der Diagonale. Somit sind beide Eigenwerte = 1. Die Eigenvektoren sind $[10]^T$ und $[01]^T$. Die Eigenvektoren in die Matrix $\boldsymbol{Q}$ eingetragen ergibt:

$$\boldsymbol{Q}^T = \boldsymbol{Q} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.16}$$

$$\tilde{\boldsymbol{v}}[0] = \boldsymbol{Q}^T \boldsymbol{v}[0] = \begin{pmatrix} -2 \\ -1 \end{pmatrix} \tag{2.17}$$

Diese Werte in die Gleichung für die Kostenfunktion eingesetzt ergibt:

$$J(\boldsymbol{c}) = J_{min} + 4e^{-2n\mu} + e^{-2n\mu} \tag{2.18}$$

3 MATLAB Problem 2.3

a) In den Plots ist deutlich ersichtlich, dass μ direkt in die Zeitkonstanten, also in die Konvergenzgeschwindigkeit einfließt. Im Falle von zero-mean White-Noise sind alle Eigenwerte der Autokorrelationsmatrix 1. Somit ergeben sich die Zeitkonstanten theoretisch: $\approx \frac{1}{\mu\lambda_k} = \frac{1}{\mu}$. Diese Formel stimmte auch sehr gut mit den aus den Plots ermittelten Werten für τ_k überein. Wie in der Abbildung 3.4 zu sehen, ist der Algorithmus bei $\mu = 1$ im Falle des LMS instabil. Der Misalignmentvector divergiert und der MSE wird immer größer. Beim NLMS und $\mu = 1$ ist keine deutliche Konvergenz des Misalignmentvectors ersichtlich. Der Misalignmentvector divergiert allerdings auch nicht so wie beim LMS(ohne Normalisierung). Der Grund hierfür ist, dass μ durch $\boldsymbol{x}[n]$ dividiert wird und somit etwas kleiner ist. μ ist aber dennoch zu groß um die Koeffizienten vernünftig anzunähern.
Für ein größeres μ konvergiert der MSE wesentlich schneller gegen sein Minimum, allerdings ist auch der Excess-Error etwas größer.
Beim NLMS wird μ durch $|\boldsymbol{x}[n]|^2$ dividiert. Bei einer Varianz von 1 und einer Filterordnung $N = 3$ ist der tapped input Vektor $\boldsymbol{x}[n]$ 3 Elemente groß. Somit ergibt sich ein Erwartungswert für $\boldsymbol{x}[n]$ von 3. Aus diesem Grund sind die Zeitkonstanten für diese Eingangsvarianz um den Faktor 3 größer. Der entscheidende Vorteil beim LMS ist jetzt, dass die Konvergenzgeschwindigkeit nicht mehr von der Eingangsvarianz abhängt. Diese ist im nächsten Abschnitt zu sehen.

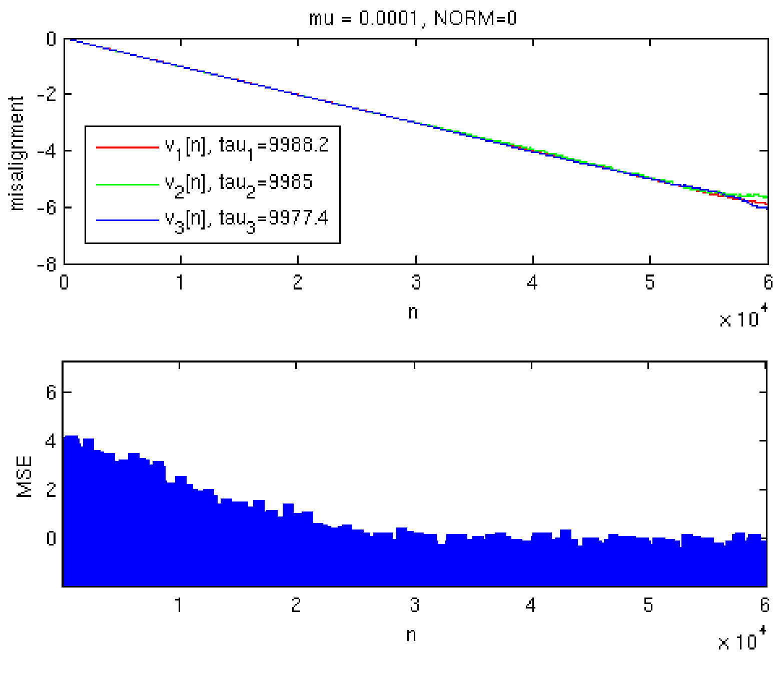

Abbildung 3.1: LMS, $\mu = 0.0001$, zero-mean unit variance white noise input

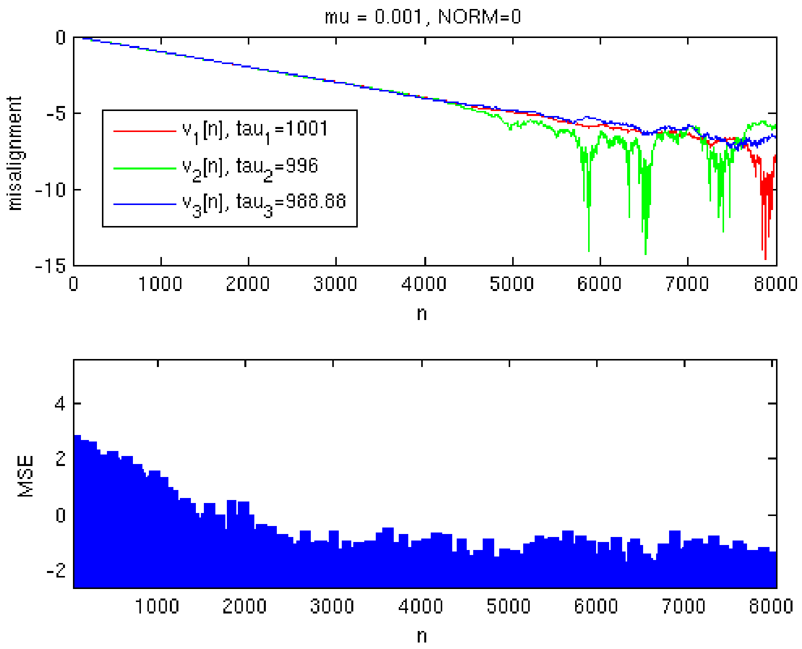

Abbildung 3.2: LMS, $\mu = 0.001$, zero-mean unit variance white noise input

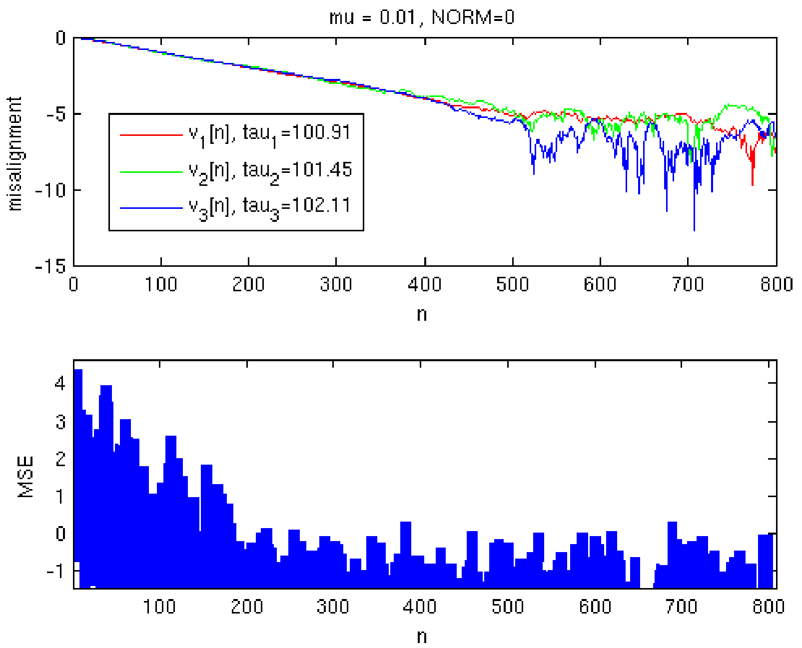

Abbildung 3.3: LMS, $\mu = 0.01$, zero-mean unit variance white noise input

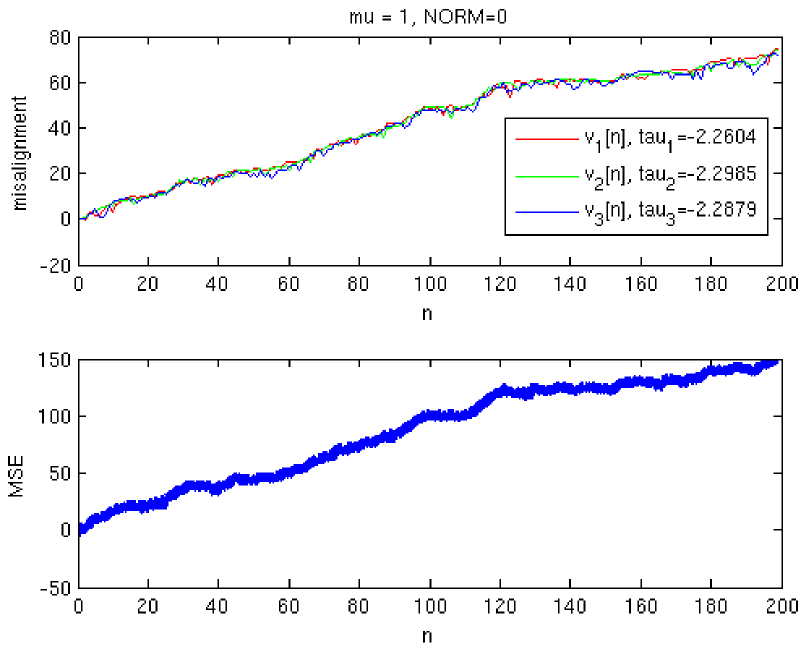

Abbildung 3.4: LMS, $\mu = 1$, zero-mean unit variance white noise input

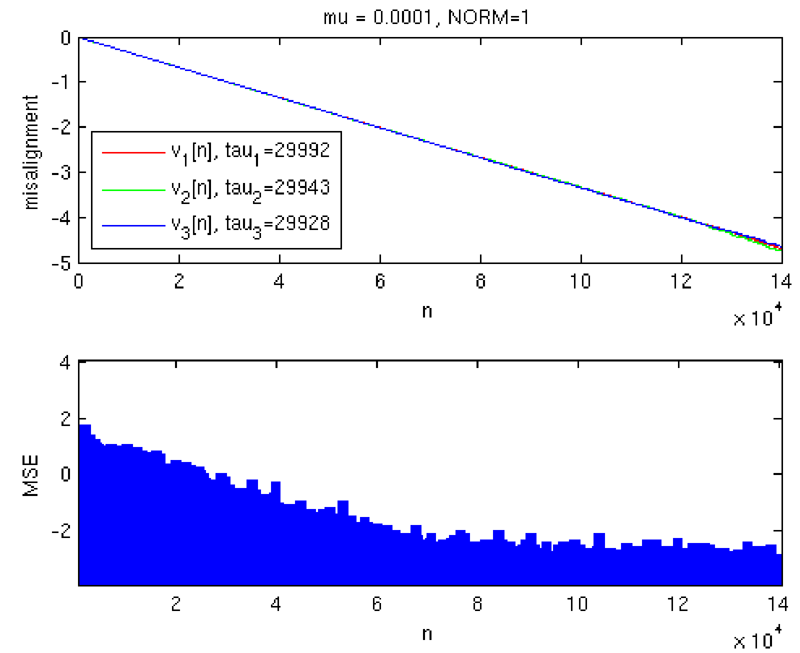

Abbildung 3.5: NLMS, $\mu = 0.0001$, zero-mean unit variance white noise input

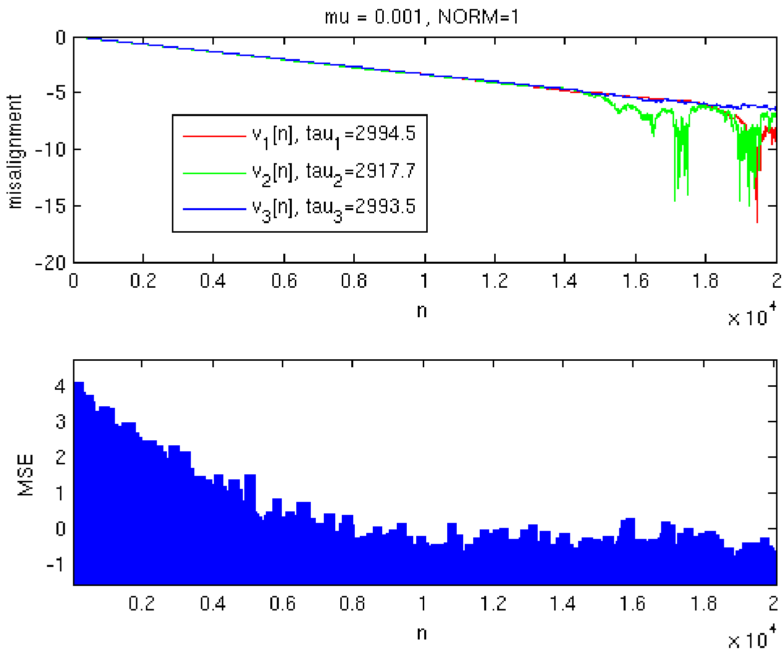

Abbildung 3.6: NLMS, $\mu = 0.001$, zero-mean unit variance white noise input

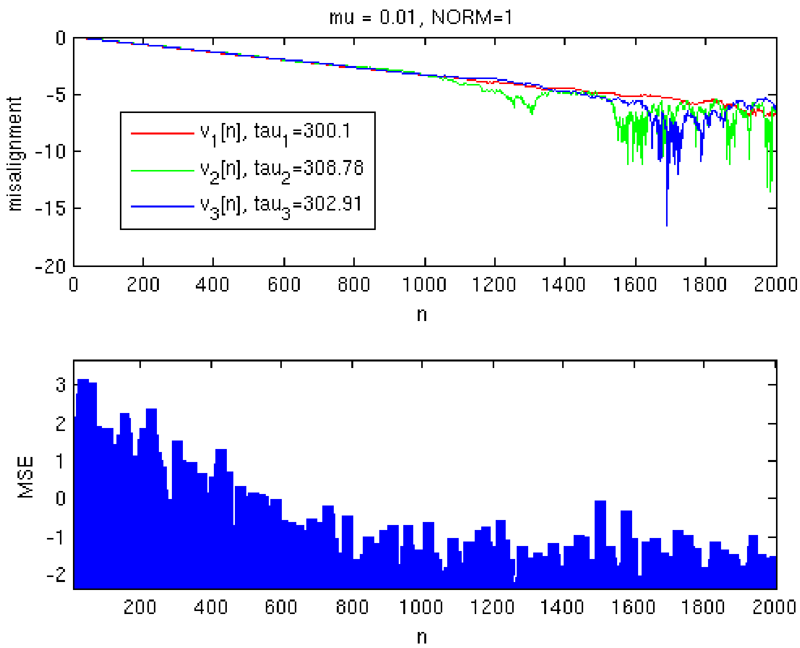

Abbildung 3.7: NLMS, $\mu = 0.01$, zero-mean unit variance white noise input

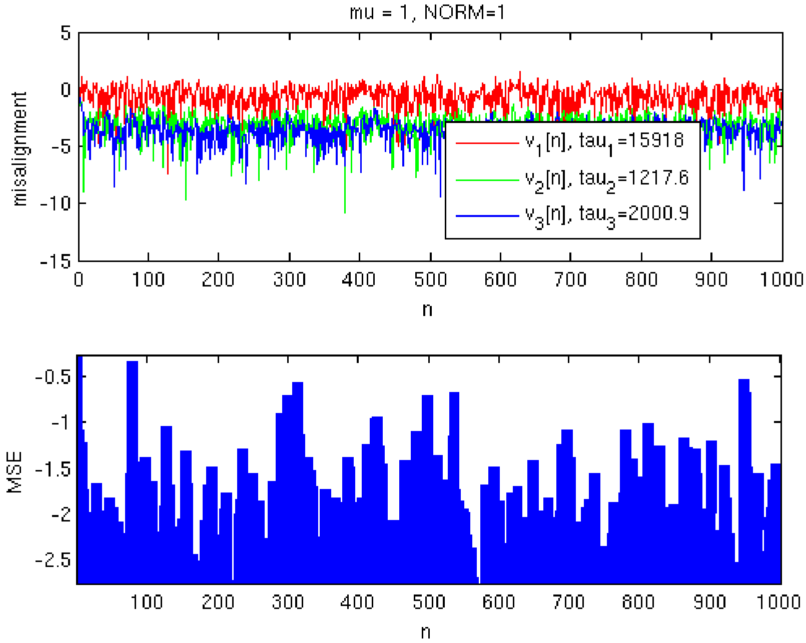

Abbildung 3.8: NLMS, $\mu = 1$, zero-mean unit variance white noise input

b) In den Plots wird ersichtlich, dass die Konvergenzgeschwindigkeit des LMS von der Varianz des Eingangssingals abhängt. Somit Konvergiert dieser Algorithmus bei einer größeren Varianz von $\boldsymbol{x}[n]$ schneller, bzw. bei einer sehr geringen Eingangsvarianz nur sehr gering. Um das ganze noch mit zahlen zu belegen, betrachten wir zunächst wieder die Formel für die Zeitkonstanten: $\tau_k \approx \frac{1}{\mu\lambda_k}$. Da es sich wieder um weißes Rauschen handelt, sind alle Eigenwerte λ_k gleich und entsprechen der Varianz. Somit ergibt sich bei einer Eingangsvarianz von 1.5 theoretisch eine Zeitkonstante von $1/(1.5\mu) \approx 0.666/\mu$. Diese Werte können sehr gut anhand der Plots abgelesen werden.
Beim NLMS hat die Eingangsvarianz keinen Einfluss da μ entsprechend der Eingangsvarianz angepasst wird. Dies ist sehr gut anhand der berechneten Zeitkonstanten in den Abbildungen 3.10 und 3.12 zu sehen.

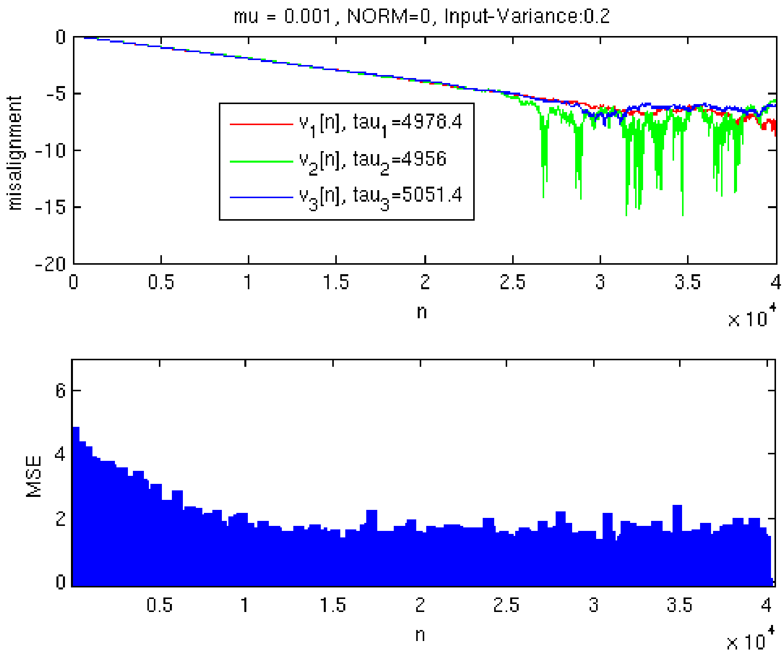

Abbildung 3.9: LMS, $\mu = 0.001$, zero-mean white noise input with variance=0.2

c) Da das Eingangssignal jetzt kein Weißes Rauschen mehr ist, sind die Eigenwerte von $\boldsymbol{R}_{xx}$ nicht mehr gleich. Dadurch konvergieren die Komponenten von v unterschiedlich schnell.
Der MSE nimmt kontinuierlich ab, bis er den noise-floor erreicht (MMSE + excess-error).

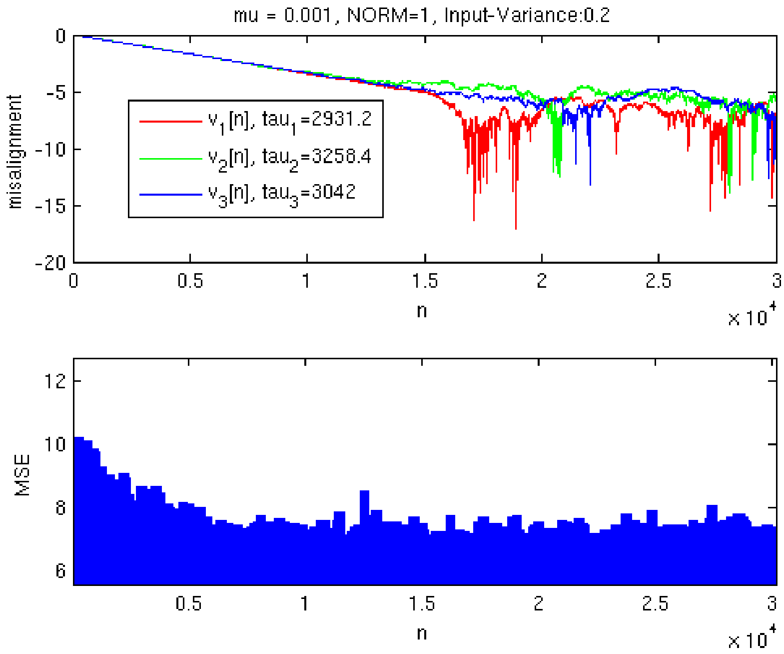

Abbildung 3.10: NLMS, $\mu = 0.001$, zero-mean white noise input with variance=0.2

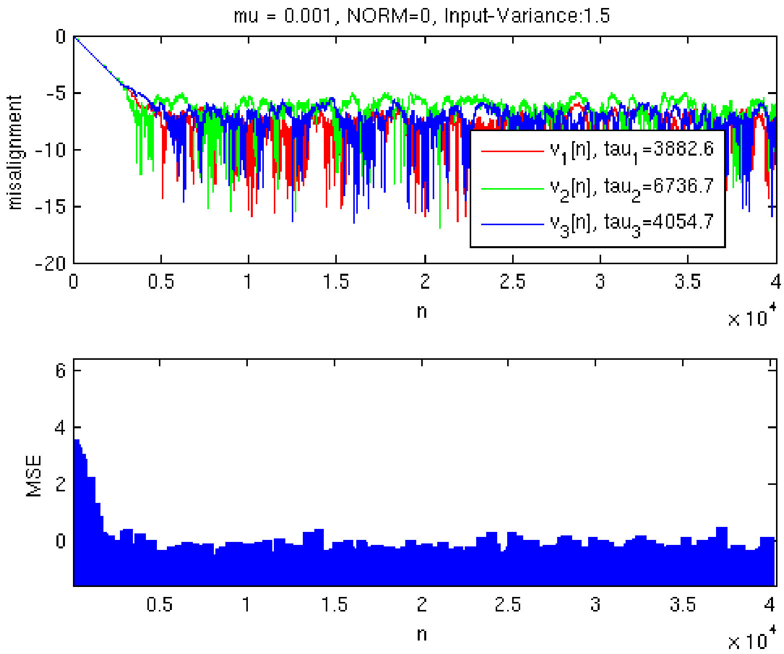

Abbildung 3.11: LMS, $\mu = 0.001$, zero-mean white noise input with variance=1.5

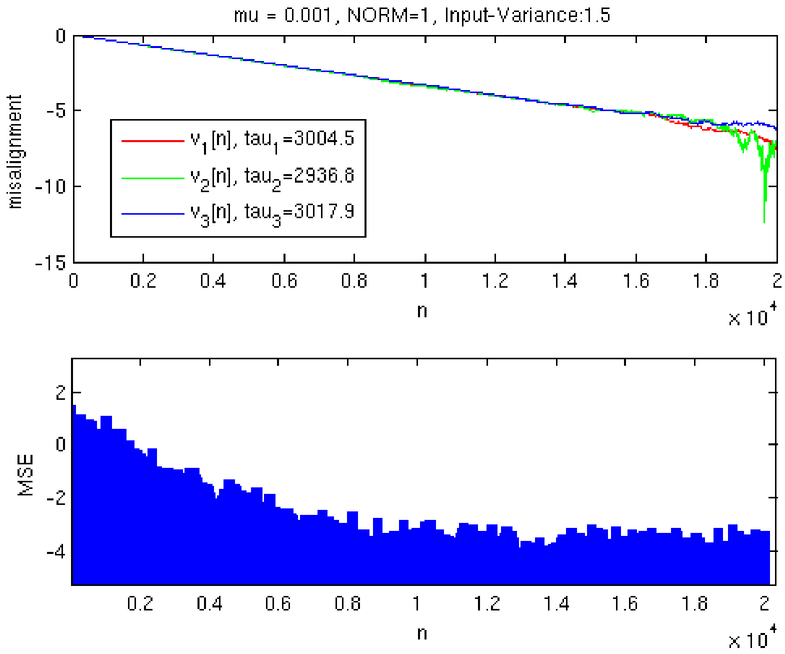

Abbildung 3.12: NLMS, $\mu = 0.001$, zero-mean white noise input with variance=1.5

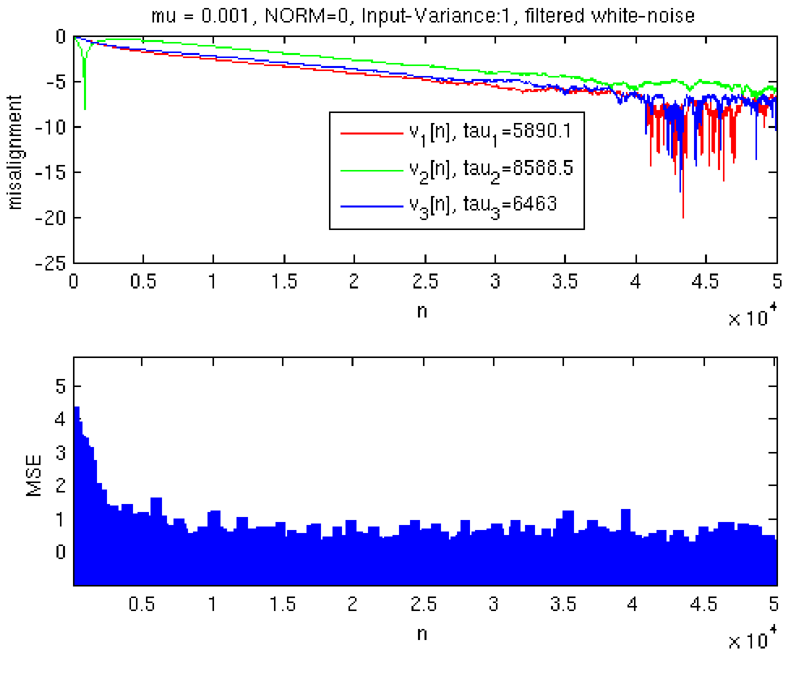

Abbildung 3.13: LMS, $\mu = 0.001$, zero-mean white noise input with variance=1, filtered

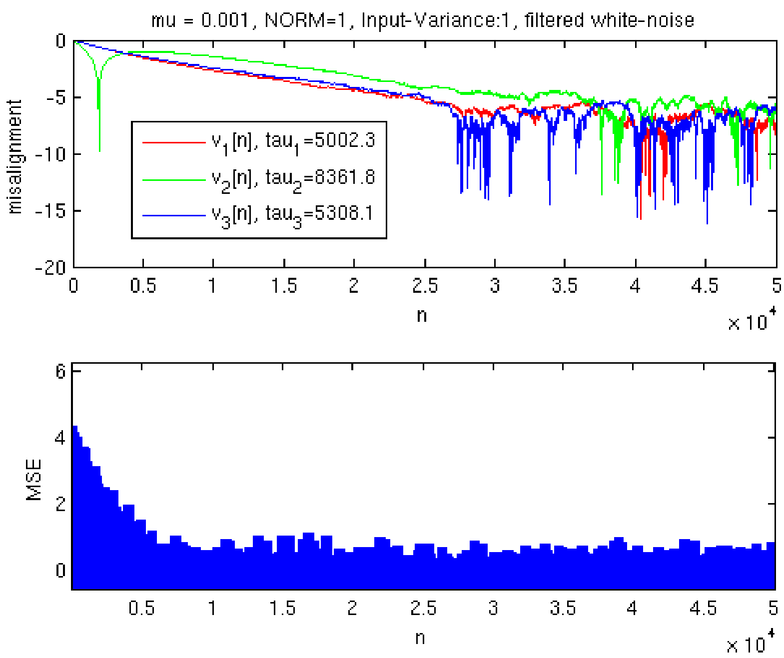

Abbildung 3.14: NLMS, $\mu = 0.001$, zero-mean white noise input with variance=1, filtered

d) Der MSE nimmt kontinuierlich ab, bis er den noise-floor erreicht (MMSE + excess-error).

e) Berechnung des Excess Errors: Wie in der Vorlesung vom 18.11.2011 hergeleitet ergibt sich der Excess Error wie folgt:

$$M \cdot MMSE \tag{3.1}$$

Wobei M als Misadjustment bezeichnet wird

$$M = \frac{\mu||\boldsymbol{x[n]}||^2}{2 - \mu||\boldsymbol{x[n]}||^2} \tag{3.2}$$

Wobei $\boldsymbol{x[n]}||^2$ Der Varianz des Eingangssingals entspricht.
Das Minimum der Kostenfunktion(=MMSE) (siehe Gleichung 2.7 wurde wie folgt ermittelt:

$$MMSE = \sigma_v^2 - \boldsymbol{p}^T \boldsymbol{R}_{xx}^{-1} \boldsymbol{p} \tag{3.3}$$

Formt man nun die Wiener Hopf Solution auf $\boldsymbol{p}$ um so erhält man: $\boldsymbol{p} = \boldsymbol{R}_{xx}\boldsymbol{h}$. In die Gleichung 3.3 eingesetzt ergibt das (für white noise!!!):

$$MMSE = \sigma_v^2 - \boldsymbol{h}^T \boldsymbol{R}_{xx}^T \boldsymbol{R}_{xx}^{-1} \boldsymbol{R}_{xx} \boldsymbol{h} = \sigma_v^2 - \boldsymbol{h}^T \boldsymbol{R}_{xx}^T \boldsymbol{h} = \sigma_v^2 - ||h||\sigma_x^2 \tag{3.4}$$

$$excesserror = MMSE \cdot M \tag{3.5}$$

Nun ergeben sich folgende Werte für den Excesserror in dB:

Eingangs-Varianz σ_x^2	$\mu = 0.0001$	$\mu = 0.001$	$\mu = 0.01$	$\mu = 1$
1	-44.31	-34.31	-24.31	-4.31
1.5	-40.7786	-30.7786	-20.7786	-0.7786
0.2	-58.4873	-48.4873	-38.4873	-18.4873

Tabelle 3.1: Ermittelten Werte für den Excess-Error in dB beim LMS

Eingangs-Varianz σ_x^2	$\mu = 0.0001$	$\mu = 0.001$	$\mu = 0.01$	$\mu = 1$
1	-49.0873	-39.0873	-29.0873	-9.0873
1.5	-45.5498	-35.5498	-25.5498	-5.5498
0.2	-63.2585	-53.2585	-43.2585	-23.2585

Tabelle 3.2: Ermittelten Werte für den Excess-Error in dB beim NLMS

Wie bereits erklärt nimmt der Excesserror bei einem kleinerem μ ab. Bei einer kleineren Eingangsvarianz ist der Excesserror auch kleiner. Da beim NLMS das effektive μ verkleinert wird, ist bei dieser Variante des LMS der Excesserror etwas geringer.

3.1 Analytical Problem 2.4: Average Behaviour of the LMS

Zur Berechnung wurde die Struktur aus Abbildung 2 in der Angabe verwendet. Das Eingangssignal $x[n]$ ist *white*, laut der Angabe zur Abbildung 2 ist das Noise $\nu[n]$ ebenfalls *white Gaussian noise*, die beiden sind also nicht korreliert und *white*.
Die Lernregel des LMS lautet:

$$\mathbf{c}[n] = \mathbf{c}[n-1] + \mu e[n]\mathbf{x}[n] \tag{3.6}$$

Wir gehen von Konvergenz *on average* aus:

$$E\{\mathbf{c}[n]\} = E\{\mathbf{c}[n-1]\} = E\{\mathbf{c}_\infty\} \tag{3.7}$$

Der Fehler $e[n]$ ergibt sich zu:

$$e[n] = -\mathbf{v}^T[n-1]\mathbf{x}[n] + \nu[n] \tag{3.8}$$

Setzen wir den Fehler in die Lernregel für LMS ein und wenden den Erwartungswert-Operator an, können wir die Lösung für $\mathbf{c}_\infty$ berechnen:

$$\begin{aligned}
E\{\mathbf{c}[n]\} &= E\{\mathbf{c}[n-1]\} + \mu E\{e[n]\mathbf{x}[n]\} && (3.9)\\
&= E\{\mathbf{c}[n-1]\} - \mu E\left\{\mathbf{v}^T[n-1]\mathbf{x}[n]\mathbf{x}[n] + \nu[n]\mathbf{x}[n]\right\} && (3.10)\\
&= E\{\mathbf{c}[n-1]\} - \mu E\left\{\mathbf{v}^T[n-1]\mathbf{x}[n]\mathbf{x}[n]\right\} - \mu E\{\nu[n]\mathbf{x}[n]\} && (3.11)\\
&= E\{\mathbf{c}[n-1]\} - \mu E\left\{\left(\mathbf{c}^T[n-1] - \mathbf{h}^T\right)\mathbf{x}[n]\mathbf{x}[n]\right\} - \mu E\{\nu[n]\mathbf{x}[n]\} && (3.12)\\
&= E\{\mathbf{c}[n-1]\} - \mu E\left\{\mathbf{c}^T[n-1]\mathbf{x}[n]\mathbf{x}[n] - \mathbf{h}^T\mathbf{x}[n]\mathbf{x}[n]\right\} - \mu E\{\nu[n]\mathbf{x}[n]\} && (3.13)\\
&= E\{\mathbf{c}[n-1]\} - \mu E\left\{\underbrace{\mathbf{x}[n]\mathbf{x}^T[n]}_{\mathbf{R}_{xx}}\mathbf{c}[n-1] - \underbrace{\mathbf{x}[n]\mathbf{x}^T[n]}_{\mathbf{R}_{xx}}\mathbf{h}\right\} - \mu \underbrace{E\{\nu[n]\mathbf{x}[n]\}}_{\text{both WGN, uncorrelated} \Rightarrow 0} && (3.14)\\
&= E\{\mathbf{c}[n-1]\} - \mu\mathbf{R}_{xx}E\{\mathbf{c}[n-1]\} + \mu\mathbf{R}_{xx}\mathbf{h} && (3.15)\\
\Rightarrow E\{\mathbf{c}_\infty\} &= E\{\mathbf{c}_\infty\} - \mu\mathbf{R}_{xx}E\{\mathbf{c}_\infty\} + \mu\mathbf{R}_{xx}\mathbf{h} && (3.16)\\
&= (\mathbf{I} - \mu\mathbf{R}_{xx})\,E\{\mathbf{c}_\infty\} + \mu\mathbf{R}_{xx}\mathbf{h} && (3.17)
\end{aligned}$$

Diese Gleichung wird nun auf $E\{\mathbf{c}_\infty\}$ umgeformt:

$$\begin{aligned}
E\{\mathbf{c}_\infty\} &= (\mathbf{I} - \mu\mathbf{R}_{xx})\,E\{\mathbf{c}_\infty\} + \mu\mathbf{R}_{xx}\mathbf{h} && (3.18)\\
\mu\mathbf{R}_{xx}E\{\mathbf{c}_\infty\} &= \mu\mathbf{R}_{xx}\mathbf{h} && (3.19)\\
E\{\mathbf{c}_\infty\} &= \mathbf{R}_{xx}^{-1}\mathbf{R}_{xx}\mathbf{h} && (3.20)\\
E\{\mathbf{c}_\infty\} &= \mathbf{h} && (3.21)
\end{aligned}$$

Die Lösung ist also die gleiche wie die optimale MSE-Lösung.

4 Listings

4.1 (N)LMS

```
function [y,e,c] = n_lms(x,d,N,mu,NORM,c0)
% INPUTS:
% x ...... input signal vector
% d ...... desired output signal (of same length as x)
% N ...... number of filter coefficients
% mu .,... step size parameter
% NORM ... set "1" for NLMS (bias=0), "0" for LMS
% c0 ..... initial coefficient vector (optional; default all zeros)
% OUTPUTS:
% y ...... output signal vector (of same length as x)
% e ...... error signal vector (of same length as x)
% c ...... coefficient matrix (N rows, number of columns = length of x)


x = x(:);
d = d(:);

if(nargin < 6)
    c0 = zeros(N,1);
end

y = zeros(length(x),1);
e = zeros(length(x),1);
c = zeros(N,length(x));


c0 = c0(:);

c_last = c0;

for n = 1:length(x)
    x_taped = x(1:n);
    if(length(x_taped) <= N)
        x_taped = [zeros(N-length(x_taped),1); x_taped];
    else
        x_taped = x_taped(end-N+1:end);
    end

    x_taped = flipud(x_taped);

    y(n) = c_last'*x_taped;
    e(n) = d(n)-y(n);

    if(NORM == 1)
        factor = norm(x_taped)^2;
    else
        factor = 1;
    end

    c(:,n) = c_last + mu*e(n)*x_taped./factor;
    c_last = c(:,n);
end




end
```

4.2 Performancevergleich (N)LMS

```
clc;
close all;
clear;


h = [0.6;0.2;0.4];
noise_variance = 0.008;
N = 3;
M = 30;



for NORM = 0:1
    for mu = [0.001, 0.01, 1]

        if NORM == 0
            switch(mu)
                case 0.0001
                    Ns = 60000; % number of samples
                case 0.001
                    Ns = 8000;
                case 0.01
                    Ns = 800;
                case 1
                    Ns = 200;
            end
        else
            switch(mu)
                case 0.0001
                    Ns = 140000; % number of samples
                case 0.001
                    Ns = 20000;
                case 0.01
                    Ns = 2000;
                case 1
                    Ns = 1000;
            end
        end

        c = zeros(N, Ns, M);
        y = zeros(Ns, M);
        e = zeros(Ns, M);

        for m = 1:M
            x = randn(1,Ns);
            d = filter(h,1,x);
            d = d + sqrt(noise_variance)*randn(size(d));

            [y(:,m),e(:,m),c(:,:,m)] = n_lms(x, d, N, mu, NORM, zeros(N, 1));

        end

        y_mean = mean(y, 2);
        e_mean = mean(e, 2);
        c_mean = mean(c, 3);

        v_mean = c_mean - repmat(h, 1, Ns);

        %% plots:

        log_v_mean1 = real(log(v_mean(1,:)./v_mean(1,1)));
        log_v_mean2 = real(log(v_mean(2,:)./v_mean(2,1)));
        log_v_mean3 = real(log(v_mean(3,:)./v_mean(3,1)));


        log_e_mean = log(e_mean.^2 ./ e_mean(1)^2);

        figure;
        subplot(2,1,1);
        plot(0:(Ns-1), log_v_mean1, 'r-');
        hold on;
```

```
            plot(0:(Ns-1), log_v_mean2, 'g-');
            plot(0:(Ns-1), log_v_mean3, 'b-');



            poly1 = polyfit(0:(Ns/2), log_v_mean1(1:(Ns/2+1)),1);
            poly2 = polyfit(0:(Ns/2), log_v_mean2(1:(Ns/2+1)),1);
            poly3 = polyfit(0:(Ns/2), log_v_mean3(1:(Ns/2+1)),1);

            tau1 = -1/poly1(1);
            tau2 = -1/poly2(1);
            tau3 = -1/poly3(1);

            legend(['v_1[n], tau_1=', num2str(tau1,5)], ['v_2[n], tau_2=',
                                          num2str(tau2,5)], ['v_3[n], tau_3=',
                                          num2str(tau3,5)]);
            title(['mu = ', num2str(mu), ', NORM=', num2str(NORM)]);
            xlabel('n');
            ylabel('misalignment');

            subplot(2,1,2);
            plot(0:(Ns-1), log_e_mean, 'LineWidth', 5);
            xlabel('n');
            ylabel('MSE');
        end
    end

    %%
    mu = 0.001;

    for NORM = [0:1]
        for x_var = [0.2, 1.5]

            if NORM == 0
                switch(x_var)
                    case 0.2
                        Ns = 40000;
                    case 1.5
                        NS = 6000;
                end
            else
                switch(x_var)
                    case 0.2
                        Ns = 30000;
                    case 1.5
                        Ns = 20000;
                end
            end


            c = zeros(N, Ns, M);
            y = zeros(Ns, M);
            e = zeros(Ns, M);

            for m = 1:M
                x = sqrt(x_var)*randn(1,Ns);
                d = filter(h,1,x);
                d = d + sqrt(noise_variance)*randn(size(d));

                [y(:,m),e(:,m),c(:,:,m)] = n_lms(x, d, N, mu, NORM, zeros(N, 1));

            end

            y_mean = mean(y, 2);
            e_mean = mean(e, 2);
            c_mean = mean(c, 3);

            v_mean = c_mean - repmat(h, 1, Ns);

            %% plots:

            log_v_mean1 = real(log(v_mean(1,:)./v_mean(1,1)));
            log_v_mean2 = real(log(v_mean(2,:)./v_mean(2,1)));
            log_v_mean3 = real(log(v_mean(3,:)./v_mean(3,1)));
```

```
        log_e_mean = log(e_mean.^2 ./ e_mean(1)^2);

        figure;
        subplot(2,1,1);
        plot(0:(Ns-1), log_v_mean1, 'r-');
        hold on;
        plot(0:(Ns-1), log_v_mean2, 'g-');
        plot(0:(Ns-1), log_v_mean3, 'b-');


        %for calculation of tau just use half of the valuee
        %(just use the beggining of the curves:
        poly1 = polyfit(0:(Ns/2), log_v_mean1(1:(Ns/2+1)),1);
        poly2 = polyfit(0:(Ns/2), log_v_mean2(1:(Ns/2+1)),1);
        poly3 = polyfit(0:(Ns/2), log_v_mean3(1:(Ns/2+1)),1);

        tau1 = -1/poly1(1);
        tau2 = -1/poly2(1);
        tau3 = -1/poly3(1);

        legend(['v_1[n], tau_1=', num2str(tau1,5)], ['v_2[n], tau_2=',
                                    num2str(tau2,5)], ['v_3[n], tau_3=',
                                    num2str(tau3,5)]);
        title(['mu = ', num2str(mu), ', NORM=', num2str(NORM), ',
                                    Input-Variance:', num2str(x_var)]);
        xlabel('n');
        ylabel('misalignment');

        subplot(2,1,2);
        plot(0:(Ns-1), log_e_mean, 'LineWidth', 5);
        xlabel('n');
        ylabel('MSE');
    end
end




%%
mu = 0.001;

for NORM = [0,1]
    for x_var = [1]

        if NORM == 0
            Ns = 50000;
        else
            Ns = 50000;
        end


        c = zeros(N, Ns, M);
        y = zeros(Ns, M);
        e = zeros(Ns, M);

        for m = 1:M
            x = sqrt(x_var)*randn(1,Ns);

            % filter noise:
            x = filter([0.5 0.5], 1, x);

            d = filter(h,1,x);
            d = d + sqrt(noise_variance)*randn(size(d));

            [y(:,m),e(:,m),c(:,:,m)] = n_lms(x, d, N, mu, NORM, zeros(N, 1));

        end

        y_mean = mean(y, 2);
        e_mean = mean(e, 2);
        c_mean = mean(c, 3);

```

```
        v_mean = c_mean - repmat(h, 1, Ns);

        %% plots:

        log_v_mean1 = real(log(v_mean(1,:)./v_mean(1,1)));
        log_v_mean2 = real(log(v_mean(2,:)./v_mean(2,1)));
        log_v_mean3 = real(log(v_mean(3,:)./v_mean(3,1)));


        log_e_mean = log(e_mean.^2 ./ e_mean(1)^2);

        figure;
        subplot(2,1,1);
        plot(0:(Ns-1), log_v_mean1, 'r-');
        hold on;
        plot(0:(Ns-1), log_v_mean2, 'g-');
        plot(0:(Ns-1), log_v_mean3, 'b-');


        %for calculation of tau just use half of the valuee
        %(just use the beggining of the curves:
        poly1 = polyfit(0:(Ns/2), log_v_mean1(1:(Ns/2+1)),1);
        poly2 = polyfit(0:(Ns/2), log_v_mean2(1:(Ns/2+1)),1);
        poly3 = polyfit(0:(Ns/2), log_v_mean3(1:(Ns/2+1)),1);

        tau1 = -1/poly1(1);
        tau2 = -1/poly2(1);
        tau3 = -1/poly3(1);

        legend(['v_1[n], tau_1=', num2str(tau1,5)], ['v_2[n], tau_2=',
                                  num2str(tau2,5)], ['v_3[n], tau_3=',
                                  num2str(tau3,5)]);
        title(['mu = ', num2str(mu), ', NORM=', num2str(NORM), ',
                                  Input-Variance:', num2str(x_var), ',
                                  filtered white-noise']);
        xlabel('n');
        ylabel('misalignment');

        subplot(2,1,2);
        plot(0:(Ns-1), log_e_mean, 'LineWidth', 5);
        xlabel('n');
        ylabel('MSE');
    end
end
```