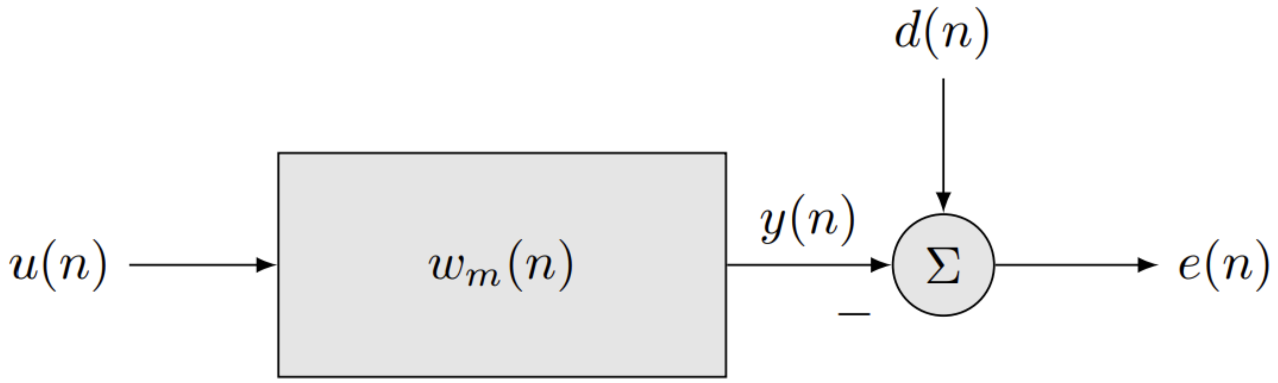
Recursive Least Squeares Adaptive Filters

Jesper Rindom Jensen, jrj@es.aau.dk

Advanced Signal Processing, Electronic Systems, Aalborg University

Adaptive filtering – Least Squares Solution

- Consider an adaptive filter setup:



- We introduce vectors containing the signal and filter values:

$$\mathbf{w}(n) = \begin{bmatrix} w_0(n) & w_1(n) & \cdots & w_{M-1}(n) \end{bmatrix}^T$$

$$\mathbf{u}(n) = \begin{bmatrix} u(n) & u(n-1) & \cdots & u(n-M+1) \end{bmatrix}^T$$

$$\mathbf{A}(n) = \begin{bmatrix} \mathbf{u}(1) & \mathbf{u}(2) & \cdots & \mathbf{u}(n) \end{bmatrix}^T$$

$$\mathbf{d}(n) = \begin{bmatrix} d(1) & d(2) & \cdots & d(n) \end{bmatrix}^T$$

$$\mathbf{e}(n) = \begin{bmatrix} e(1) & e(2) & \cdots & e(n) \end{bmatrix}^T$$

Adaptive filtering – Least Squares Solution

- With the vector notation, we have that

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{A}(n)\mathbf{w}(n)$$

- The LS cost function can then be written as

$$\begin{aligned} J_2(\mathbf{w}(n)) = \mathbf{e}^T(n)\mathbf{e}(n) &= (\mathbf{d}(n) - \mathbf{A}(n)\mathbf{w}(n))^T(\mathbf{d}(n) - \mathbf{A}(n)\mathbf{w}(n)) \\ &= \mathbf{d}^T(n)\mathbf{d}(n) + \mathbf{w}^T(n)\mathbf{A}^T(n)\mathbf{A}(n)\mathbf{w}(n) - 2\mathbf{w}^T(n)\mathbf{A}^T(n)\mathbf{d}(n) \end{aligned}$$

- Based on this, the optimal filter in the LS sense is

$$\mathbf{w}_o(n) = (\mathbf{A}^T(n)\mathbf{A}(n))^{-1}\mathbf{A}^T(n)\mathbf{d}(n)$$

- **Problem:** **A**(*n*) and **d**(*n*) keeps growing as *n* is increasing!

Weighted Least Squares

- New data should be assigned greater weight than old data

$$J_\beta(\mathbf{w}(n)) = \sum_{i=1}^{n} \beta(n,i)e^2(i) = \mathbf{e}^T(n)\mathbf{B}(n)\mathbf{e}(n)$$

- where

$$\mathbf{B}(n) = \begin{bmatrix} \beta(n,1) & 0 & \cdots & 0 \\ 0 & \beta(n,2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \beta(n,n) \end{bmatrix}$$

- Use the 5-step procedure to design the filters.

Filter design – step 1

- Construct the cost function

$$J_\beta(\mathbf{w}(n)) = \mathbf{e}^T(n)\mathbf{B}(n)\mathbf{e}(n) = (\mathbf{d}(n) - \mathbf{A}(n)\mathbf{w}(n))^T\mathbf{B}(n)(\mathbf{d}(n) - \mathbf{A}(n)\mathbf{w}(n))$$
$$= \mathbf{d}^T(n)\mathbf{B}(n)\mathbf{d}(n) + \mathbf{w}^T(n)\mathbf{\Phi}(n)\mathbf{w}(n) - 2\mathbf{w}^T(n)\boldsymbol{\varphi}(n)$$

- with

$$\mathbf{\Phi}(n) = \mathbf{A}^T(n)\mathbf{B}(n)\mathbf{A}(n)$$
$$\boldsymbol{\varphi}(n) = \mathbf{A}^T(n)\mathbf{B}(n)\mathbf{d}(n)$$

- We refer to $\mathbf{\Phi}(n)$ and $\boldsymbol{\varphi}(n)$ as the correlation matrix and the cross-correlation vector, respectively.
- They are scaled and weighted estimates of $\mathbf{R}_u$ and $\mathbf{r}_{ud}$

Filter design – step 2+3

- **Step 2:** Find the gradient

$$\mathbf{g}(\mathbf{w}(n)) = \frac{\partial J_\beta(\mathbf{w}(n))}{\partial \mathbf{w}(n)} = (\boldsymbol{\Phi}(n) + \boldsymbol{\Phi}^T(n))\mathbf{w}(n) - 2\boldsymbol{\varphi}(n) = 2\boldsymbol{\Phi}(n)\mathbf{w}(n) - 2\boldsymbol{\varphi}(n)$$

- **Step 3:** Solve **g**(**w**(*n*)) = **0** for **w**(*n*)

$$\mathbf{g}(\mathbf{w}(n)) = 2\boldsymbol{\Phi}(n)\mathbf{w}(n) - 2\boldsymbol{\varphi}(n)$$

$$\boldsymbol{\Phi}(n)\mathbf{w}(n) = \boldsymbol{\varphi}(n)$$ (weighted normal equations)

$$\mathbf{w}(n) = \boldsymbol{\Phi}^{-1}(n)\boldsymbol{\varphi}(n)$$ (if correlation matrix is invertible)

Filter design – step 4+5

- **Step 4**: Find the Hessian

$$\mathbf{H}(\mathbf{w}(n)) = 2\mathbf{\Phi}(n)$$

- Positive definite for all **w**(n) if **A**(n) is full rank and $\beta(n,i) > 0$ for all $n \geq i > 0$.

- **Step 5**: Implication
  - $J_\beta(\mathbf{w}(n))$ is a convex function, and
  - $\mathbf{w}_o(n) = \mathbf{\Phi}^{-1}(n)\boldsymbol{\varphi}(n)$ is the global minimiser
- Solution ($\mathbf{w}_o(n)$) often referred to as WLS solution.

Estimation of the (Cross-)Correlation

- Comparing weighted normal equations and Wiener-Hopf equations:

$$\hat{\mathbf{R}}_u(n) = c(n,\beta)\boldsymbol{\Phi}(n) = c(n,\beta)\mathbf{A}^T(n)\mathbf{B}(n)\mathbf{A}(n) = c(n,\beta)\sum_{i=1}^{n}\beta(n,i)\mathbf{u}(i)\mathbf{u}^T(i)$$

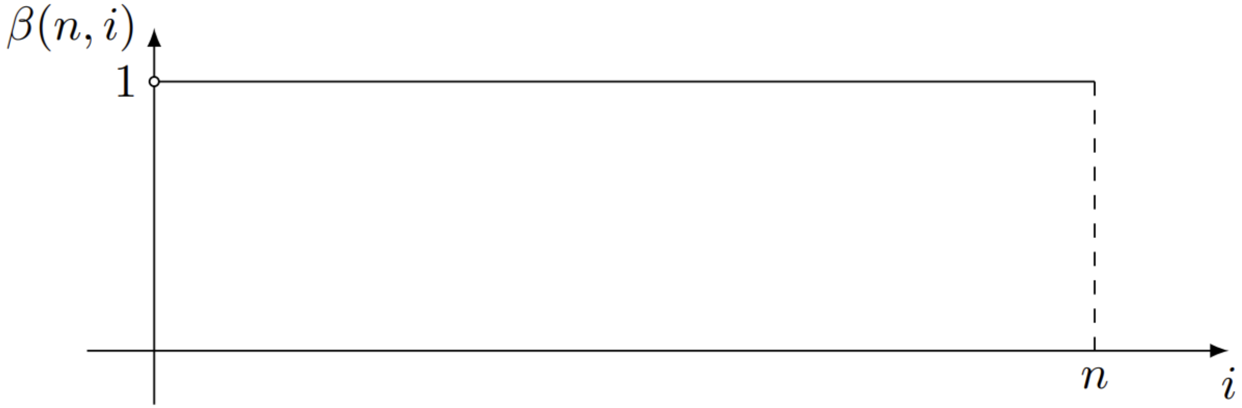
$$\hat{\mathbf{r}}_{ud}(n) = c(n,\beta)\boldsymbol{\varphi}(n) = c(n,\beta)\mathbf{A}^T(n)\mathbf{B}(n)\mathbf{d}(n) = c(n,\beta)\sum_{i=1}^{n}\beta(n,i)\mathbf{u}(i)d(i)$$

- Constant $c(n, \beta)$ depends on n and weighting $\beta(n,i)$.
- Can be selected so the (cross-)correlation estimates are unbiased

Growing Window Weight Function

- The **growing window** weight function

$$\beta(n,i) = \begin{cases} 1 & 0 < i \leq n \\ 0 & \text{otherwise} \end{cases}$$



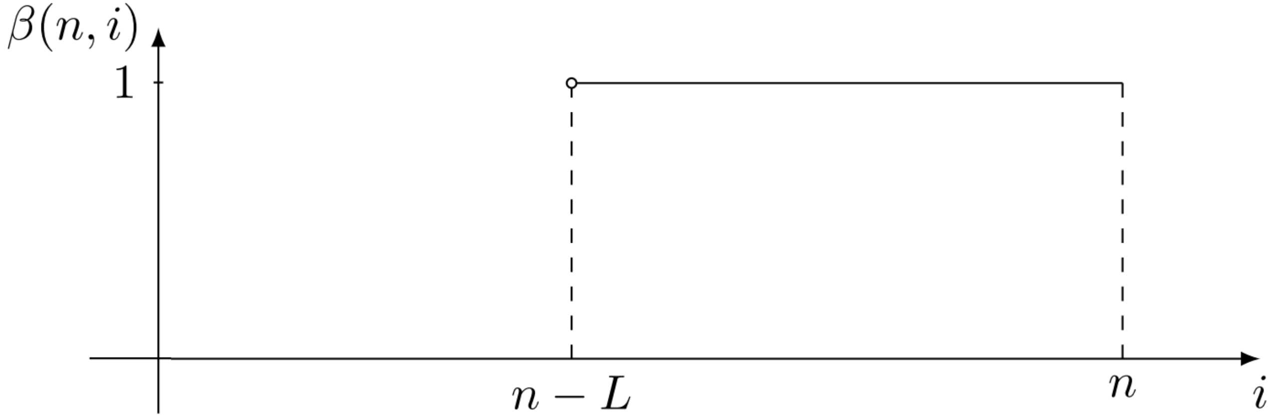
- Reduces WLS to the standard LS problem.
- In order to obtained unbiases estimates of $\mathbf{R}_u(n)$ and $\mathbf{r}_{ud}(n)$:

$$c(n,\beta) = \frac{1}{n}$$

Sliding Window Weight Function

- The **sliding window** weight function:

$$\beta(n,i) = \begin{cases} 1 & n-L < i \leq n \\ 0 & \text{otherwise} \end{cases}$$



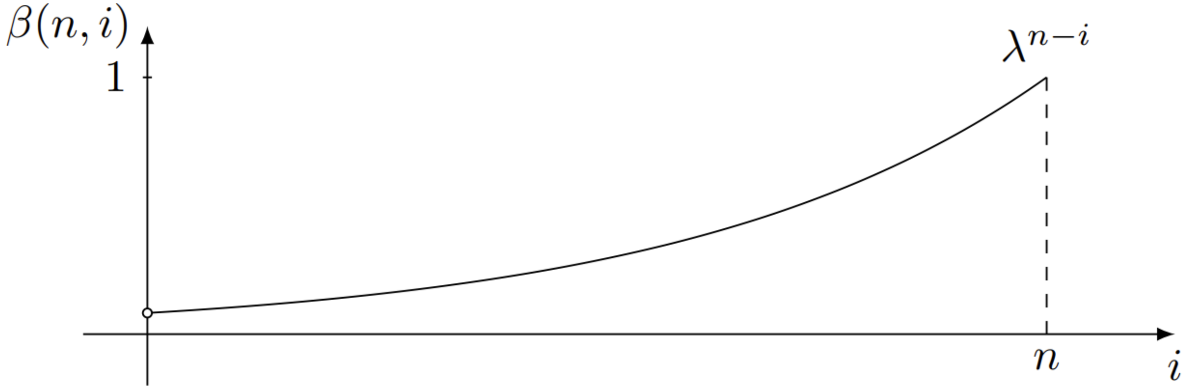
- Selecting L = n, the sliding window reduces to the growing window.
- In order to obtained unbiases estimates of $\mathbf{R}_u(n)$ and $\mathbf{r}_{ud}(n)$:

$$c(n,\beta) = \frac{1}{L}$$

Exponential Weight Function

- The **exponential window** weight function:

$$\beta(n,i) = \begin{cases} \lambda^{n-i} & 0 < i \le n \\ 0 & \text{otherwise} \end{cases}$$



- If we select $\lambda = 1$, the exponential window reduces to the growing window.
- In order to obtained unbiases estimates of $\mathbf{R}_u(n)$ and $\mathbf{r}_{ud}(n)$:

$$c(n,\beta) = \begin{cases} \frac{1-\lambda}{1-\lambda^n} & 0 < \lambda < 1 \\ \frac{1}{n} & \lambda = 1 \end{cases}$$

Recursive Least-Squares Algorithm (Exponential Weight Function)

- In an online algorithm, we solve the weighted normal equations for every time index *n*

$$\boldsymbol{\Phi}(n)\mathbf{w}(n) = \boldsymbol{\varphi}(n)$$

- Direct solution:

$$\mathbf{w}(n) = \boldsymbol{\Phi}^{-1}(n)\boldsymbol{\varphi}(n)$$

- Computationally complex since:
  1. **A**(*n*), **B**(*n*), and **d**(*n*) grows with n ➔ direction solution infeasible without finite window.
  2. Complexity scales cubically with filter order: $\mathcal{O}(M^3)$
- RLS algorithm **solves both problems**!

Recursive Computation of (Cross-)Correlation

- (Cross-)correlation estimates can be recursively updated by rewriting:

$$\begin{aligned}
\mathbf{\Phi}(n) &= \mathbf{A}^T(n)\mathbf{B}(n)\mathbf{A}(n) = \sum_{i=1}^{n} \lambda^{n-i}\mathbf{u}(i)\mathbf{u}^T(i) \\
&= \lambda^0\mathbf{u}(n)\mathbf{u}^T(n) + \sum_{i=1}^{n-1} \lambda^{n-i}\mathbf{u}(i)\mathbf{u}^T(i) = \mathbf{u}(n)\mathbf{u}^T(n) + \lambda\sum_{i=1}^{n-1} \lambda^{n-1-i}\mathbf{u}(i)\mathbf{u}^T(i) \\
&= \mathbf{u}(n)\mathbf{u}^T(n) + \lambda\mathbf{\Phi}(n-1)
\end{aligned}$$

Reduced computational complexity!

$$\begin{aligned}
\boldsymbol{\varphi}(n) &= \mathbf{A}^T(n)\mathbf{B}(n)\mathbf{d}(n) = \sum_{i=1}^{n} \lambda^{n-i}\mathbf{u}(i)d(i) \\
&= \lambda^0\mathbf{u}(n)d(n) + \sum_{i=1}^{n-1} \lambda^{n-i}\mathbf{u}(i)d(i) = \mathbf{u}(n)d(n) + \lambda\sum_{i=1}^{n-1} \lambda^{n-1-i}\mathbf{u}(i)d(i) \\
&= \mathbf{u}(n)d(n) + \lambda\boldsymbol{\varphi}(n-1)
\end{aligned}$$

Inversion of Correlation Matrix

- **Matrix inversion lemma (MIL)**

$$(\mathbf{X} + \mathbf{U}\mathbf{Y}\mathbf{V})^{-1} = \mathbf{X}^{-1} - \mathbf{X}^{-1}\mathbf{U}(\mathbf{Y}^{-1} + \mathbf{V}\mathbf{X}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{X}^{-1}$$

- By choosing...

$$\begin{aligned}
\mathbf{X} &= \lambda\mathbf{\Phi}(n-1) \\
\mathbf{U} &= \mathbf{u}(n) \\
\mathbf{V} &= \mathbf{u}^T(n) \\
\mathbf{Y} &= 1
\end{aligned}$$

- and invoking the MIL on the recursive update equation, we get...

$$\mathbf{\Phi}^{-1}(n) = \lambda^{-1}\mathbf{\Phi}^{-1}(n-1) - \lambda^{-2}\frac{\mathbf{\Phi}^{-1}(n-1)\mathbf{u}(n)\mathbf{u}^T(n)\mathbf{\Phi}^{-1}(n-1)}{1+\lambda^{-1}\mathbf{u}^T(n)\mathbf{\Phi}^{-1}(n-1)\mathbf{u}(n)}$$

Simplifying notation

- By introducing the notation…

$$\mathbf{P}(n) = \mathbf{\Phi}^{-1}(n)$$

$$\mathbf{k}(n) = \frac{\mathbf{P}(n-1)\mathbf{u}(n)}{\lambda + \mathbf{u}^T(n)\mathbf{P}(n-1)\mathbf{u}(n)}$$

- we may rewrite the recursive update of the inverse correlation matrix…

$$\mathbf{P}(n) = \lambda^{-1}[\mathbf{P}(n-1) - \mathbf{k}(n)\mathbf{u}^T(n)\mathbf{P}(n-1)]$$

$$\begin{aligned}\mathbf{k}(n) &= \lambda^{-1}[\mathbf{P}(n-1) - \mathbf{k}(n)\mathbf{u}^T(n)\mathbf{P}(n-1)]\mathbf{u}(n) \\ &= \mathbf{P}(n)\mathbf{u}(n)\end{aligned}$$

Recursive Filter Update

- From here, we can develop the recursive filter update as...

$$
\begin{aligned}
\mathbf{w}(n) &= \mathbf{P}(n)\boldsymbol{\varphi}(n) \\
&= \mathbf{P}(n)[\mathbf{u}(n)d(n) + \lambda\boldsymbol{\varphi}(n-1)] = \lambda\mathbf{P}(n)\mathbf{P}^{-1}(n-1)\mathbf{w}(n-1) + \mathbf{k}(n)d(n) \\
&= [\mathbf{P}(n-1) - \mathbf{k}(n)\mathbf{u}^T(n)\mathbf{P}(n-1)]\mathbf{P}^{-1}(n-1)\mathbf{w}(n-1) + \mathbf{k}(n)d(n) \\
&= \mathbf{w}(n-1) - \mathbf{k}(n)\mathbf{u}^T(n)\mathbf{w}(n-1) + \mathbf{k}(n)d(n) \\
&= \mathbf{w}(n-1) + \mathbf{k}(n)\xi(n)
\end{aligned}
$$

- where the *a priori* error is defined as...

$$\xi(n) = d(n) - \mathbf{u}^T(n)\mathbf{w}(n-1)$$

The RLS Algorithm

- Based on this, we can formulate the RLS algorithm:

$$\boldsymbol{\pi}(n) = \mathbf{P}(n-1)\mathbf{u}$$

$$\mathbf{k}(n) = \frac{\boldsymbol{\pi}(n)}{\lambda + \mathbf{u}^T(n)\boldsymbol{\pi}(n)}$$

$$\xi(n) = d(n) - \mathbf{u}^T(n)\mathbf{w}(n-1)$$

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{k}(n)\xi(n)$$

$$\mathbf{P}(n) = \lambda^{-1}[\mathbf{P}(n-1) - \mathbf{k}(n)\boldsymbol{\pi}^T(n)]$$

- Requires $O(M^2)$ operations to run one iteration for M-tap FIR filter.

Initialization

- **Biased** (decreasing bias for large *n*):

$$\mathbf{P}(0) = \delta^{-1}\mathbf{I}$$
$$\mathbf{w}(0) = \mathbf{0}$$
$$u(n) = 0, \quad \text{for } -M+1 < n < 1$$

- **Unbiased** (requires knowledge of future samples):

$$\mathbf{P}(0) = \left[\sum_{i=-M+1}^{0} \lambda^{-1}\mathbf{u}(i)\mathbf{u}^T(i)\right]^{-1}$$
$$\boldsymbol{\varphi}(0) = \sum_{i=-M+1}^{0} \lambda^{-i}\mathbf{u}(i)d(i)$$

Forgetting Factor vs. Window Length

- The forgetting factor can be interpreted as a window length:

$$\lim_{n\to\infty} \sum_{i=1}^{n} \lambda^{n-i} = \lim_{n\to\infty} \sum_{i=n-L_{\text{eff}}+1}^{n} 1 = L_{\text{eff}}$$

- For 0 < λ < 1, this leads to…

$$\lim_{n\to\infty} \sum_{i=1}^{n} \lambda^{n-1} = \lim_{n\to\infty} \sum_{k=0}^{n-1} \lambda^{k} = \lim_{n\to\infty} \frac{1-\lambda^{n}}{1-\lambda} = \frac{1}{1-\lambda}$$

- Thus, we have that…

$$L_{\text{eff}} = \frac{1}{1-\lambda}$$

Steady-State Analysis

- It can be shown that...

$$J_{\text{ex}} = J_2(\mathbf{w}(\infty)) - J_{\min} \approx J_{\min} \frac{(1-\lambda)M}{1+\lambda-(1-\lambda)M}$$ (EMSE)

$$\mathcal{M} = \frac{J_{\text{ex}}}{J_{\min}} \approx \frac{(1-\lambda)M}{1+\lambda-(1-\lambda)M}$$ (Misadjustment)

$$E[\|\Delta\mathbf{w}(\infty)\|^2] \approx J_{\text{ex}} \sum_{m=1}^{M} \frac{1}{\lambda_m}$$ (MSD)

Computational Cost

- Breakdown of the **computational cost** of the RLS algorithm

Term	$\times$	$+$ or $-$	$/$
$\boldsymbol{\pi}(n) = \boldsymbol{P}(n-1)\boldsymbol{u}(n)$	M^2	$M(M-1)$	
$\boldsymbol{k}(n) = \boldsymbol{\pi}(n)/(\lambda + \boldsymbol{u}^T(n)\boldsymbol{\pi}(n))$	M	M	M
$\xi(n) = d(n) - \boldsymbol{u}^T(n)\boldsymbol{w}(n-1)$	M	M	
$\boldsymbol{w}(n) = \boldsymbol{w}(n-1) + \boldsymbol{k}(n)\xi(n)$	M	M	
$\boldsymbol{P}(n) = (\boldsymbol{P}(n-1) - \boldsymbol{k}(n)\boldsymbol{\pi}^T(n))/\lambda$	M^2	M^2	M^2
Total	$2M^2 + 3M$	$2M^2 + 2M$	$M^2 + M$

Basic Adaptive Filtering Overview

Name	Algorithm	Cost	Mean-Square Stability	EMSE	Misadjustment	MSD
SD	$\boldsymbol{g}(\boldsymbol{w}(n)) = 2\boldsymbol{R}_u\boldsymbol{w}(n) - 2\boldsymbol{r}_{ud}(n)$ $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \frac{\mu}{2}\boldsymbol{g}(\boldsymbol{w}(n))$	$\mathcal{O}(M)$	$0 < \mu < \frac{2}{\lambda_{\max}}$	0	0	0
LMS	$e(n) = d(n) - \boldsymbol{u}^T(n)\boldsymbol{w}(n)$ $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu\boldsymbol{u}(n)e(n)$	$\mathcal{O}(M)$	$0 < \mu < \frac{2}{\mathrm{tr}(\boldsymbol{R}_u)}$	$\frac{\mu}{2}J_{\min}\mathrm{tr}(\boldsymbol{R}_u)$	$\frac{\mu}{2}\mathrm{tr}(\boldsymbol{R}_u)$	$\frac{\mu}{2}J_{\min}M$
NLMS	$e(n) = d(n) - \boldsymbol{u}^T(n)\boldsymbol{w}(n)$ $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{\beta}{\epsilon + \\|\boldsymbol{u}(n)\\|^2}\boldsymbol{u}(n)e(n)$	$\mathcal{O}(M)$	$0 < \beta < 2$	$\frac{\beta}{2}J_{\min}$	$\frac{\beta}{2}$	$\frac{\beta J_{\min}}{2\mathrm{tr}(\boldsymbol{R}_u)}$
APA	$\boldsymbol{e}(n) = \boldsymbol{d}(n) - \boldsymbol{U}^T(n)\boldsymbol{w}(n)$ $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \beta\boldsymbol{U}(n)[\epsilon\boldsymbol{I} + \boldsymbol{U}^T(n)\boldsymbol{U}(n)]^{-1}\boldsymbol{e}(n)$	$\mathcal{O}(MK^2)$	$0 < \beta < 2$	$\frac{\beta}{2}J_{\min}K$	$\frac{\beta}{2}K$	no simple expression
RLS	$\boldsymbol{\pi}(n) = \boldsymbol{P}(n-1)\boldsymbol{u}(n)$ $\boldsymbol{k}(n) = \frac{\boldsymbol{\pi}(n)}{\lambda + \boldsymbol{u}^T(n)\boldsymbol{\pi}(n)}$ $\xi(n) = d(n) - \boldsymbol{u}^T(n)\boldsymbol{w}(n-1)$ $\boldsymbol{w}(n+1) = \boldsymbol{w}(n-1) + \boldsymbol{k}(n)\xi(n)$ $\boldsymbol{P}(n) = \lambda^{-1}\left[\boldsymbol{P}(n-1) - \boldsymbol{k}(n)\boldsymbol{\pi}^T(n)\right]$	$\mathcal{O}(M^2)$	$0 < \lambda \leq 1$	$\frac{J_{\min}\frac{1-\lambda}{1+\lambda}M}{1 - \frac{1-\lambda}{1+\lambda}M}$	$\frac{\frac{1-\lambda}{1+\lambda}M}{1 - \frac{1-\lambda}{1+\lambda}M}$	$\frac{J_{\min}\frac{1-\lambda}{1+\lambda}M}{1 - \frac{1-\lambda}{1+\lambda}M}\sum_{m=1}^{M}\frac{1}{\lambda_m}$

Exercise 1

- Consider a situation, where you are listening to music in an environment with background noise. Luckily you have placed a microphone, which is able to pickup the background noise. Your job is now to design a filter, which can remove a significant part of the background noise from the noisy music.
- Implement and compare the performance of LMS, normalised LMS and RLS
- To get inspiration you can look at the MATLAB file Filters.m which provides an example solution.

Exercise 2

- Implement an RLS algorithm for system identification of a stationary signal. Play around with forgetting factor and filter orders. To get inspiration you can look at the MATLAB file sys_rd_rls.m which provides an example solution.