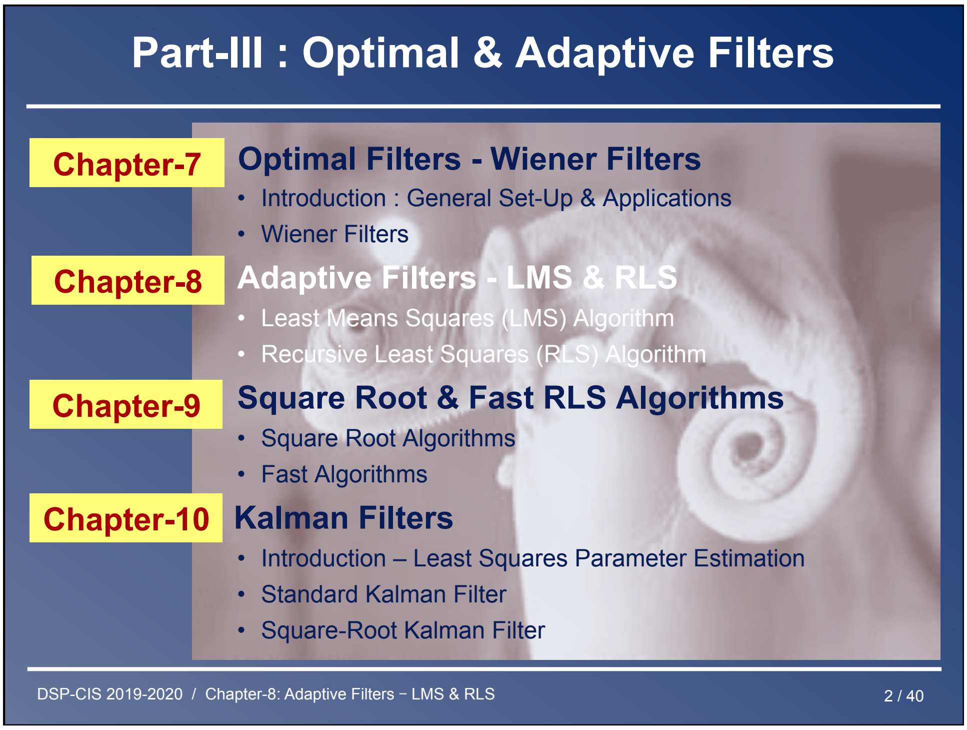
DSP-CIS

Part-III : Optimal & Adaptive Filters

Chapter-8 : Adaptive Filters − LMS & RLS

Marc Moonen
Dept. E.E./ESAT-STADIUS, KU Leuven
marc.moonen@kuleuven.be
www.esat.kuleuven.be/stadius/



Part-III : Optimal & Adaptive Filters

Chapter-7 **Optimal Filters - Wiener Filters**
- Introduction : General Set-Up & Applications
- Wiener Filters

Chapter-8 **Adaptive Filters - LMS & RLS**
- Least Means Squares (LMS) Algorithm
- Recursive Least Squares (RLS) Algorithm

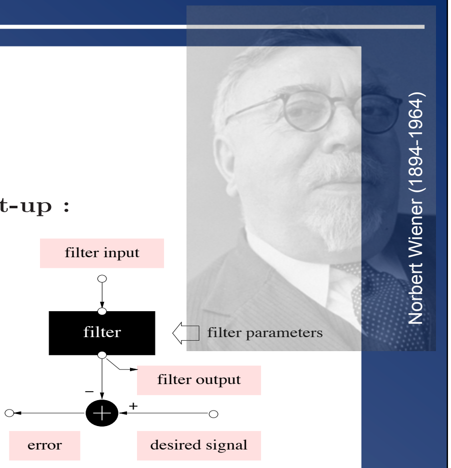
Chapter-9 **Square Root & Fast RLS Algorithms**
- Square Root Algorithms
- Fast Algorithms

Chapter-10 **Kalman Filters**
- Introduction – Least Squares Parameter Estimation
- Standard Kalman Filter
- Square-Root Kalman Filter

Recap 1/5

Prototype optimal filtering set-up :

Design filter such that for a given (i.e. 'statistical info available') input signal, filter output signal is 'optimally close' (to be defined) to a given 'desired output signal'.



Recap 2/5

1 Will use

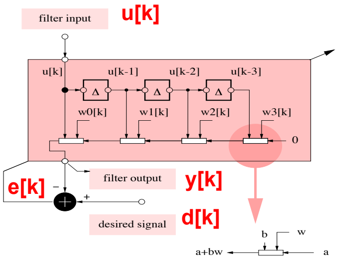
FIR filters (=tapped-delay line filter/'transversal' filter)

$$y_k = \sum_{l=0}^{L} w_l . u_{k-l} = \mathbf{w}^T . \mathbf{u}_k = \mathbf{u}_k^T . \mathbf{w}$$

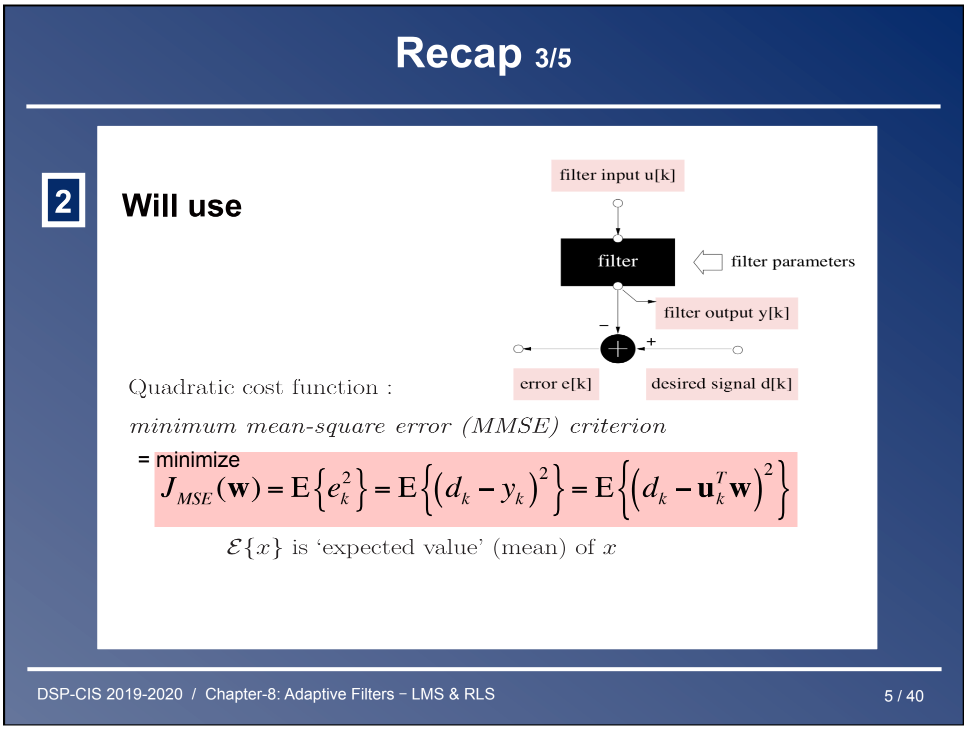
where

$$\mathbf{w}^T = \left[\begin{array}{cccc} w_0 & w_0 & \dots & w_L \end{array}\right]$$

$$\mathbf{u}_k^T = \left[\begin{array}{cccc} u_k & u_{k-1} & \dots & u_{k-L} \end{array}\right]$$



PS: Shorthand notation u_k=u[k], y_k=y[k], d_k=d[k], e_k=e[k],
Filter coefficients ('weights') are w_l (replacing b_l of previous chapters)
For adaptive filters w_l also have a time index w_l[k]



Recap 3/5

2 **Will use**

filter input u[k]

filter

filter parameters

filter output y[k]

−

+

error e[k]

desired signal d[k]

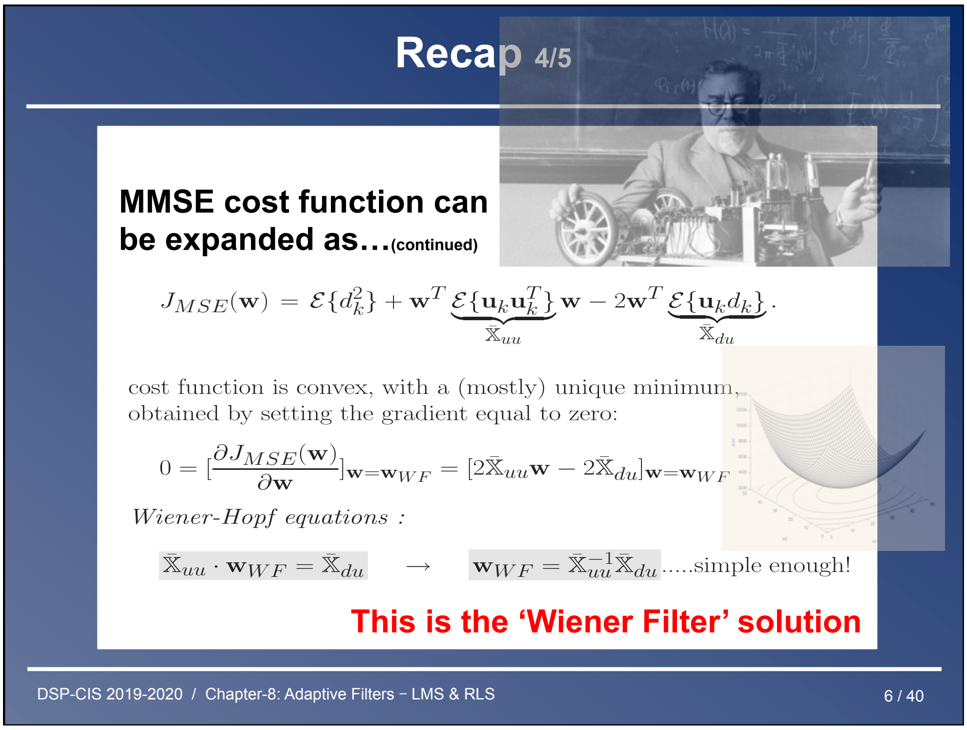
Quadratic cost function :

minimum mean-square error (MMSE) criterion

= minimize

$$J_{MSE}(\mathbf{w}) = \mathrm{E}\left\{e_k^2\right\} = \mathrm{E}\left\{\left(d_k - y_k\right)^2\right\} = \mathrm{E}\left\{\left(d_k - \mathbf{u}_k^T\mathbf{w}\right)^2\right\}$$

$\mathcal{E}\{x\}$ is 'expected value' (mean) of x



Recap 4/5

MMSE cost function can be expanded as…(continued)

$$J_{MSE}(\mathbf{w}) \ = \mathcal{E}\{d_k^2\} + \mathbf{w}^T \underbrace{\mathcal{E}\{\mathbf{u}_k\mathbf{u}_k^T\}}_{\bar{\mathbb{X}}_{uu}} \mathbf{w} - 2\mathbf{w}^T \underbrace{\mathcal{E}\{\mathbf{u}_k d_k\}}_{\bar{\mathbb{X}}_{du}}.$$

cost function is convex, with a (mostly) unique minimum, obtained by setting the gradient equal to zero:

$$0 = [\frac{\partial J_{MSE}(\mathbf{w})}{\partial \mathbf{w}}]_{\mathbf{w}=\mathbf{w}_{WF}} = [2\bar{\mathbb{X}}_{uu}\mathbf{w} - 2\bar{\mathbb{X}}_{du}]_{\mathbf{w}=\mathbf{w}_{WF}}$$

Wiener-Hopf equations :

$\bar{\mathbb{X}}_{uu} \cdot \mathbf{w}_{WF} = \bar{\mathbb{X}}_{du} \quad \rightarrow \quad \mathbf{w}_{WF} = \bar{\mathbb{X}}_{uu}^{-1}\bar{\mathbb{X}}_{du}$ .....simple enough!

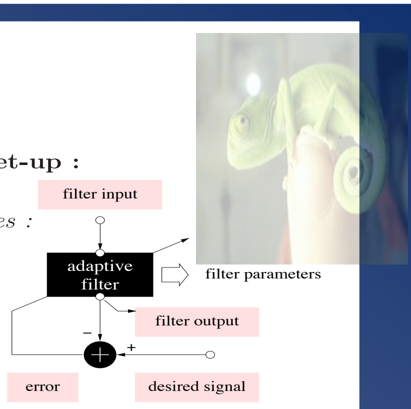
This is the 'Wiener Filter' solution

Recap 5/5

Prototype adaptive filtering set-up :

Basic operation involves 2 processes :

1. *filtering process*
2. *adaptation process*
   - adjusting filter parameters to (time-varying) environment
   - adaptation is steered by error signal



filter input

adaptive filter

filter parameters

filter output

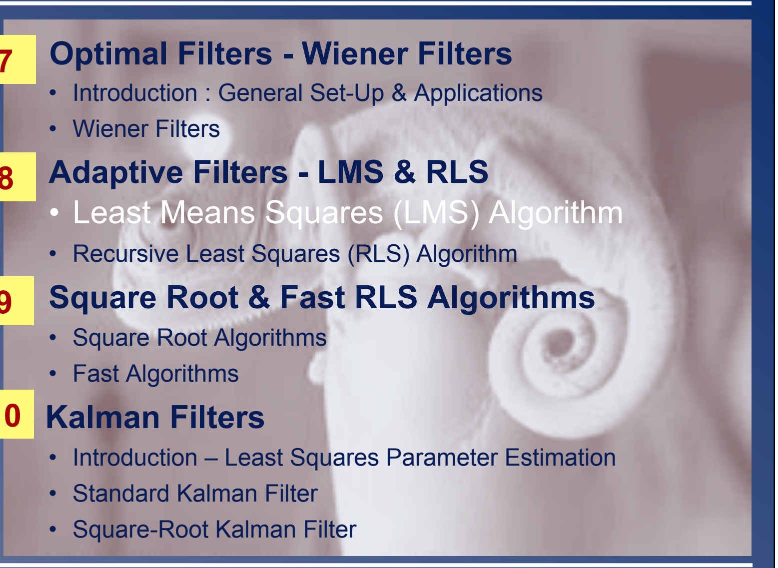
error

desired signal

- Depending on the application, either the filter parameters, the filter output or the error signal is of interest

---

Part-III : Optimal & Adaptive Filters



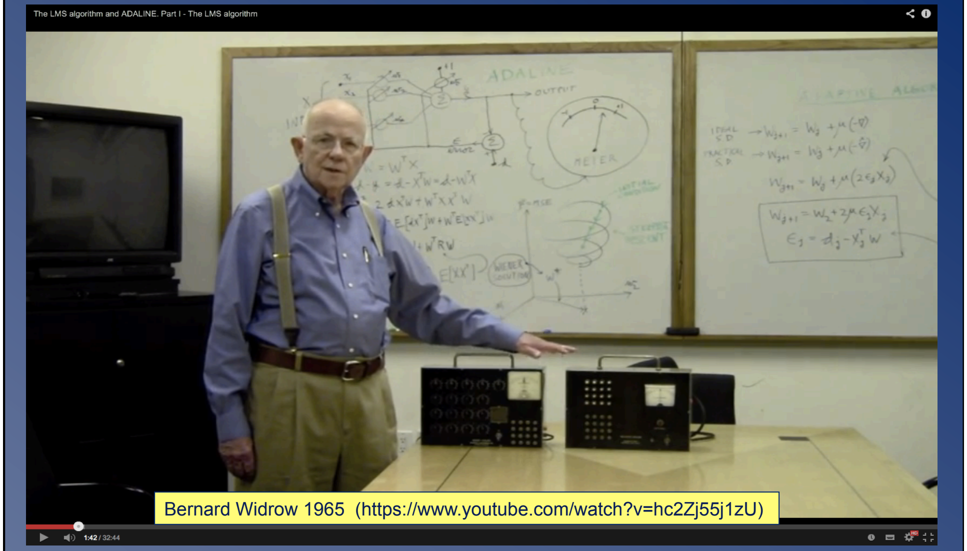
Chapter-7 **Optimal Filters - Wiener Filters**
- Introduction : General Set-Up & Applications
- Wiener Filters

Chapter-8 **Adaptive Filters - LMS & RLS**
- Least Means Squares (LMS) Algorithm
- Recursive Least Squares (RLS) Algorithm

Chapter-9 **Square Root & Fast RLS Algorithms**
- Square Root Algorithms
- Fast Algorithms

Chapter-10 **Kalman Filters**
- Introduction – Least Squares Parameter Estimation
- Standard Kalman Filter
- Square-Root Kalman Filter

Adaptive Filters: LMS



Bernard Widrow 1965 (https://www.youtube.com/watch?v=hc2Zj55j1zU)

Adaptive Filters: LMS

How do we compute the Wiener filter?

1) Cfr supra: By solving Wiener-Hopf equations (L+1 equations in L+1 unknowns)

$$\bar{\mathbb{X}}_{uu} \cdot \mathbf{w}_{WF} = \bar{\mathbb{X}}_{du}$$

2) Can also apply iterative procedure to minimize MMSE criterion, e.g.

Steepest-descent iterations :

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) + \frac{\mu}{2} \cdot \left[\frac{-\partial J_{MSE}(\mathbf{w})}{\partial \mathbf{w}} \right]_{\mathbf{w}=\mathbf{w}(n)} \\ &= \mathbf{w}(n) + \mu \cdot (\bar{\mathbb{X}}_{du} - \bar{\mathbb{X}}_{uu}\mathbf{w}(n)) \end{aligned}$$

here *n* is iteration index

μ is ‘stepsize’ (to be tuned..)

Adaptive Filters: LMS

Bound on stepsize ?

Steepest-descent iterations :

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \cdot (\bar{\mathbb{X}}_{du} - \bar{\mathbb{X}}_{uu}\mathbf{w}(n))$$

$\bar{\mathbb{X}}_{uu} \cdot \mathbf{w}_{WF} = \bar{\mathbb{X}}_{du}$

Stability ?

$$\begin{aligned}[\mathbf{w}(n+1) - \mathbf{w}_{WF}] &= (I - \mu\bar{\mathbb{X}}_{uu}) \cdot [\mathbf{w}(n) - \mathbf{w}_{WF}] \\ &= (I - \mu\bar{\mathbb{X}}_{uu})^{n+1} \cdot [\mathbf{w}(0) - \mathbf{w}_{WF}]\end{aligned}$$

stable iff (λ_i = eigenvalues of $\bar{\mathbb{X}}_{uu}$)

$$-1 < 1 - \mu\lambda_i < 1 \quad \forall i$$

$$0 < \mu < \frac{2}{\lambda_{\max}}$$

➔ large λ_max implies a small stepsize

Adaptive Filters: LMS

Convergence speed?

Transient behavior ?

$$[\mathbf{w}(n+1) - \mathbf{w}_{WF}] = (I - \mu\bar{\mathbb{X}}_{uu})^{n+1} \cdot [\mathbf{w}(0) - \mathbf{w}_{WF}]$$

with (symmetric eigenvalue decomposition)

$$\bar{\mathbb{X}}_{uu} = Q_{uu}\Lambda_{uu}Q_{uu}^T \qquad Q_{uu}^T Q_{uu} = I$$

$$[\mathbf{w}(n+1) - \mathbf{w}_{WF}] = Q_{uu}(I - \mu\Lambda_{uu})^{n+1}Q_{uu}^T \cdot [\mathbf{w}(0) - \mathbf{w}_{WF}]$$

$$\underbrace{Q_{uu}^T[\mathbf{w}(n+1) - \mathbf{w}_{WF}]}_{\text{error vector projected onto eigenvectors}} = \text{diag}\{1 - \mu\lambda_i\}^{n+1}\underbrace{Q_{uu}^T \cdot [\mathbf{w}(0) - \mathbf{w}_{WF}]}_{\text{initial error vector projected onto eigenvectors}}$$

i.e. $(1 - \mu\lambda_i)^n$ for 'mode' i (=projection on i-th eigenvector)

➔ small λ_i implies slow convergence ($1-\mu\lambda_i$ close to 1) for mode i

Adaptive Filters: LMS

Convergence speed?

Hence slowest convergence for $\lambda_i = \lambda_{min}$

With upper bound for μ (see p11) :

$$1-2(\lambda_{min}/\lambda_{max}) \leq 1-\mu\lambda_i \leq 1$$

Hence $\lambda_{min} << \lambda_{max}$ (i.e. large 'eigenvalue spread')
implies **very** slow convergence

$\lambda_{min} << \lambda_{max}$ whenever input signal u[k] is very '**colored**'

($\lambda_{min} = \lambda_{max}$ for 'white' input signal (i.e. autocorrelation matrix = I))

Adaptive Filters: LMS

LMS is derived from WF steepest-descent iterations as follows

Replace *n+1* by *n* for convenience…

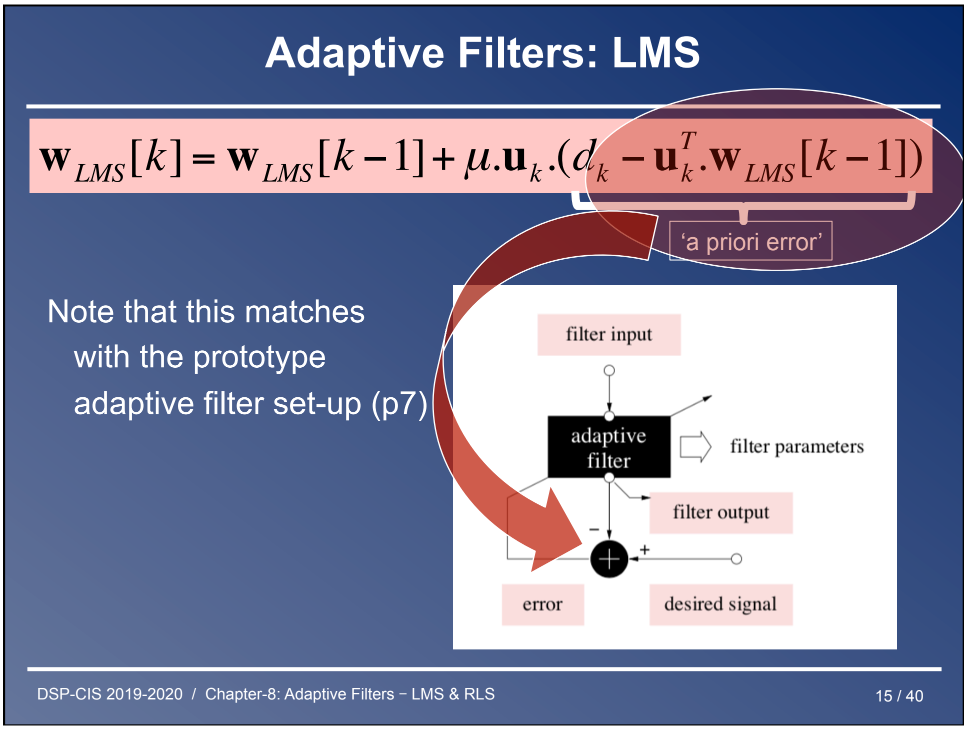
$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mu.(\mathrm{E}\{\mathbf{u}_k.d_k\} - \mathrm{E}\{\mathbf{u}_k.\mathbf{u}_k^T\}.\mathbf{w}(n-1))$$

Then replace iteration index n by time index k
(i.e. perform 1 iteration per sampling interval)

$$\mathbf{w}[k] = \mathbf{w}[k-1] + \mu.(\mathrm{E}\{\mathbf{u}_k.d_k\} - \mathrm{E}\{\mathbf{u}_k.\mathbf{u}_k^T\}.\mathbf{w}[k-1])$$

Then leave out expectation operators
(i.e. replace expected values by instantaneous estimates)

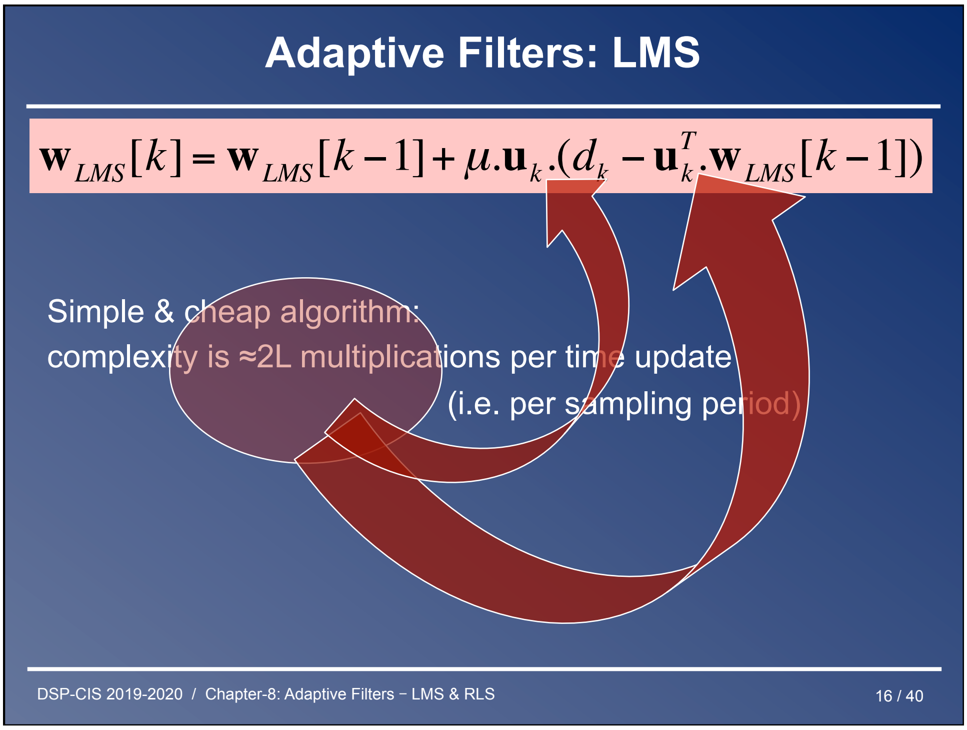
$$\mathbf{w}_{LMS}[k] = \mathbf{w}_{LMS}[k-1] + \mu.\mathbf{u}_k.\underbrace{(d_k - \mathbf{u}_k^T.\mathbf{w}_{LMS}[k-1])}_{\text{'a priori error'}}$$



Adaptive Filters: LMS

$$\mathbf{w}_{LMS}[k]=\mathbf{w}_{LMS}[k-1]+\mu.\mathbf{u}_k.(\underbrace{d_k-\mathbf{u}_k^T.\mathbf{w}_{LMS}[k-1]}_{\text{'a priori error'}})$$

Note that this matches
with the prototype
adaptive filter set-up (p7)



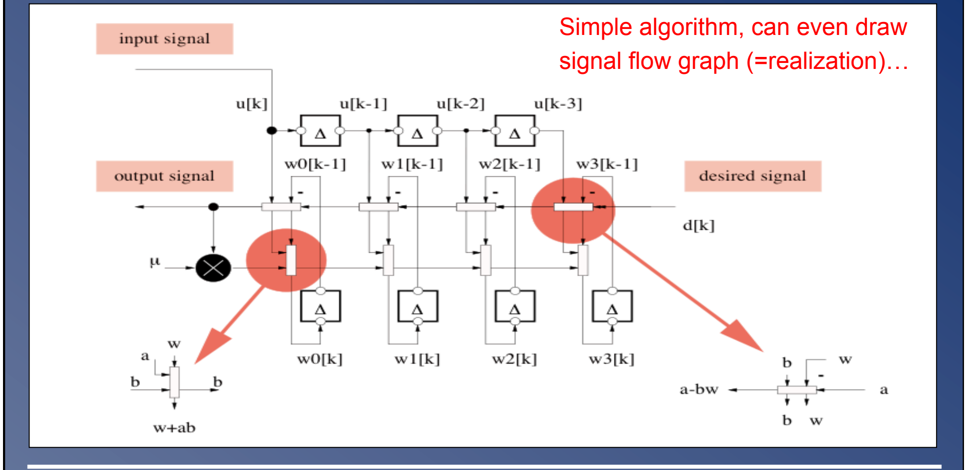
Adaptive Filters: LMS

$$\mathbf{w}_{LMS}[k]=\mathbf{w}_{LMS}[k-1]+\mu.\mathbf{u}_k.(d_k-\mathbf{u}_k^T.\mathbf{w}_{LMS}[k-1])$$

Simple & cheap algorithm:
complexity is ≈2L multiplications per time update
(i.e. per sampling period)

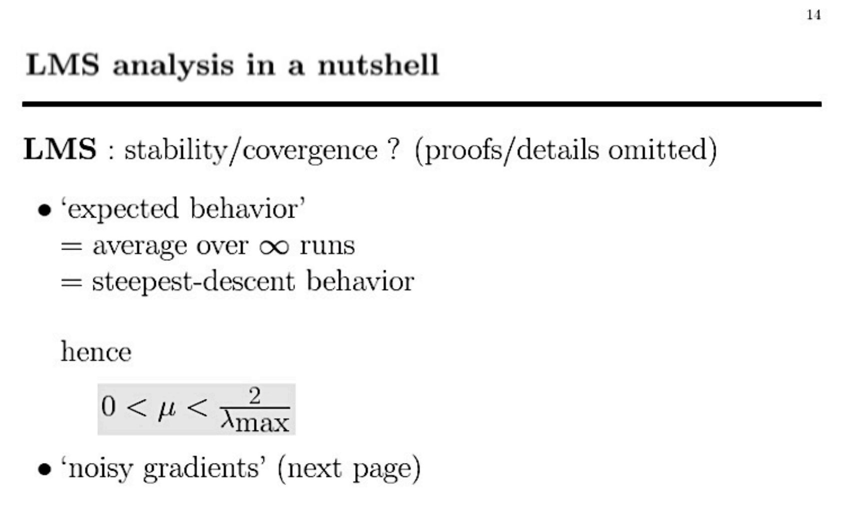
Adaptive Filters: LMS

$$\mathbf{w}_{LMS}[k]=\mathbf{w}_{LMS}[k-1]+\mu.\mathbf{u}_k.(d_k-\mathbf{u}_k^T.\mathbf{w}_{LMS}[k-1])$$



Simple algorithm, can even draw signal flow graph (=realization)…

Adaptive Filters: LMS



LMS analysis in a nutshell

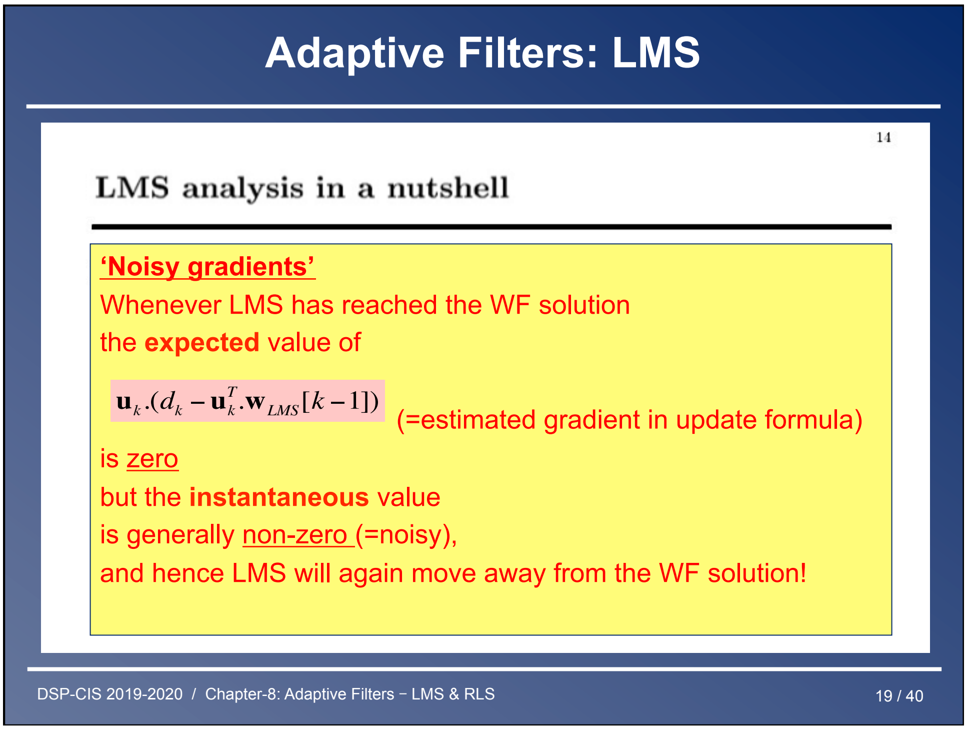
LMS : stability/covergence ? (proofs/details omitted)

- 'expected behavior'
  = average over ∞ runs
  = steepest-descent behavior

  hence

  $$0 < \mu < \frac{2}{\lambda_{\max}}$$

- 'noisy gradients' (next page)



Adaptive Filters: LMS

LMS analysis in a nutshell

<u>'Noisy gradients'</u>

Whenever LMS has reached the WF solution

the **expected** value of

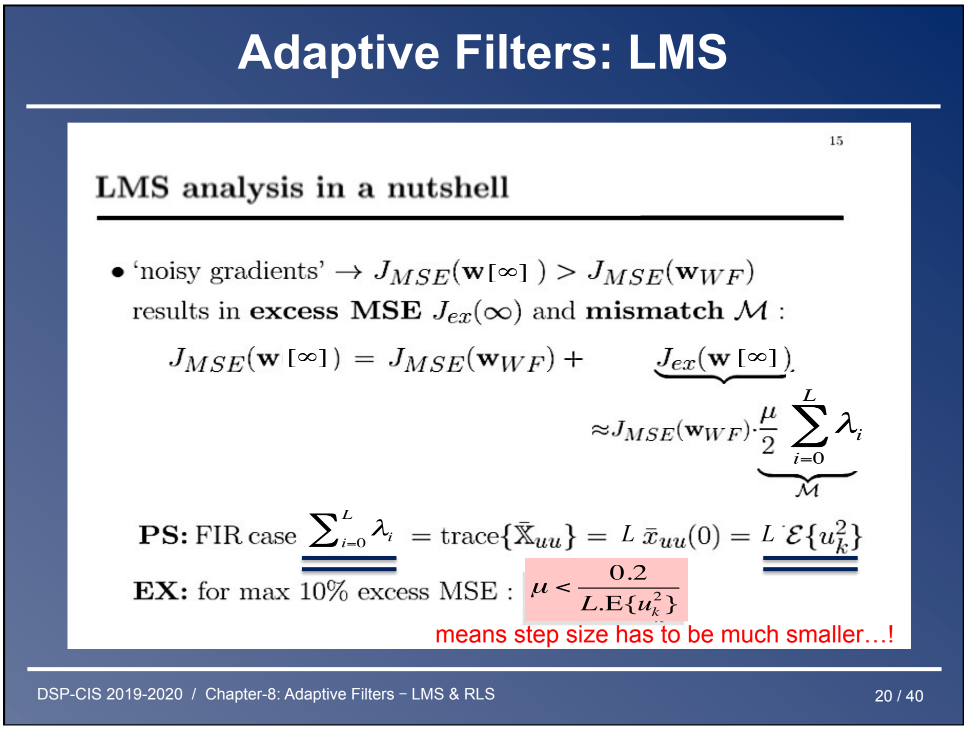
$\mathbf{u}_k.(d_k - \mathbf{u}_k^T.\mathbf{w}_{LMS}[k-1])$ (=estimated gradient in update formula)

is <u>zero</u>

but the **instantaneous** value

is generally <u>non-zero</u> (=noisy),

and hence LMS will again move away from the WF solution!



Adaptive Filters: LMS

LMS analysis in a nutshell

- 'noisy gradients' $\rightarrow J_{MSE}(\mathbf{w}[\infty]) > J_{MSE}(\mathbf{w}_{WF})$
  results in **excess MSE** $J_{ex}(\infty)$ and **mismatch** $\mathcal{M}$:

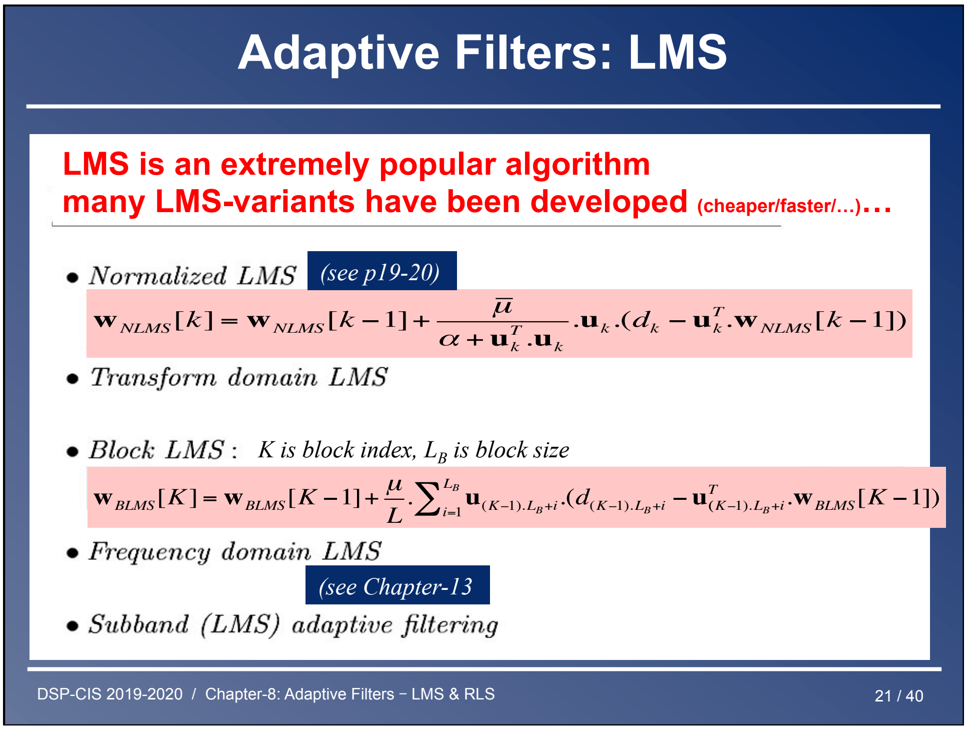
$$J_{MSE}(\mathbf{w}[\infty]) = J_{MSE}(\mathbf{w}_{WF}) + \underbrace{J_{ex}(\mathbf{w}[\infty])}$$

$$\approx J_{MSE}(\mathbf{w}_{WF}) \cdot \underbrace{\frac{\mu}{2} \sum_{i=0}^{L} \lambda_i}_{\mathcal{M}}$$

PS: FIR case $\underline{\underline{\sum_{i=0}^{L} \lambda_i}} = \text{trace}\{\bar{\mathbb{X}}_{uu}\} = L\ \bar{x}_{uu}(0) = \underline{\underline{L\ \mathcal{E}\{u_k^2\}}}$

EX: for max 10% excess MSE : $\mu < \frac{0.2}{L.\mathrm{E}\{u_k^2\}}$

means step size has to be much smaller…!



Adaptive Filters: LMS

LMS is an extremely popular algorithm
many LMS-variants have been developed (cheaper/faster/…)…

- *Normalized LMS* (see p19-20)

$$\mathbf{w}_{NLMS}[k]=\mathbf{w}_{NLMS}[k-1]+\frac{\bar{\mu}}{\alpha+\mathbf{u}_k^T.\mathbf{u}_k}.\mathbf{u}_k.(d_k-\mathbf{u}_k^T.\mathbf{w}_{NLMS}[k-1])$$

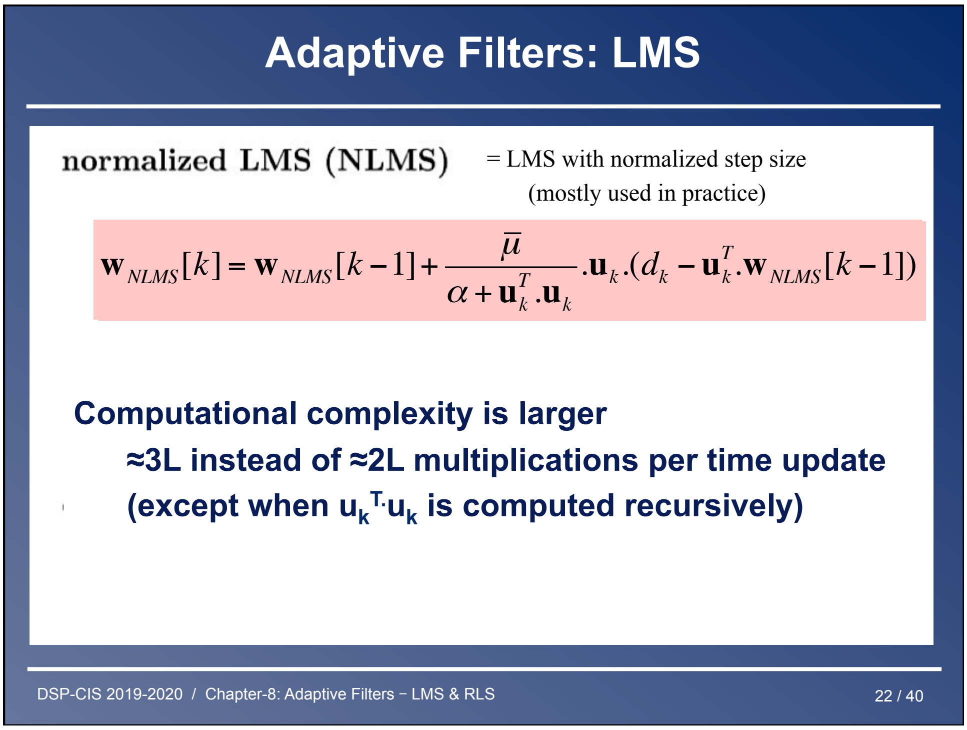
- *Transform domain LMS*
- *Block LMS* : *K is block index, L_B is block size*

$$\mathbf{w}_{BLMS}[K]=\mathbf{w}_{BLMS}[K-1]+\frac{\mu}{L}.\sum_{i=1}^{L_B}\mathbf{u}_{(K-1).L_B+i}.(d_{(K-1).L_B+i}-\mathbf{u}^T_{(K-1).L_B+i}.\mathbf{w}_{BLMS}[K-1])$$

- *Frequency domain LMS*

(see Chapter-13

- *Subband (LMS) adaptive filtering*



Adaptive Filters: LMS

normalized LMS (NLMS) = LMS with normalized step size
(mostly used in practice)

$$\mathbf{w}_{NLMS}[k]=\mathbf{w}_{NLMS}[k-1]+\frac{\bar{\mu}}{\alpha+\mathbf{u}_k^T.\mathbf{u}_k}.\mathbf{u}_k.(d_k-\mathbf{u}_k^T.\mathbf{w}_{NLMS}[k-1])$$

Computational complexity is larger
≈3L instead of ≈2L multiplications per time update
(except when $u_k^T \cdot u_k$ is computed recursively)

Adaptive Filters: LMS

normalized LMS (NLMS) = LMS with normalized step size
(mostly used in practice)

$$\mathbf{w}_{NLMS}[k] = \mathbf{w}_{NLMS}[k-1] + \frac{\bar{\mu}}{\alpha + \mathbf{u}_k^T.\mathbf{u}_k}.\mathbf{u}_k.(d_k - \mathbf{u}_k^T.\mathbf{w}_{NLMS}[k-1])$$

Step size tuning for NLMS is much easier…

- stability/convergence ? :
  convergence if $0 < \bar{\mu} < 2$
  max. 10% excess MSE obtained with $\bar{\mu} < 0.2$

Adaptive Filters: LMS

normalized LMS (NLMS) = LMS with normalized step size
(mostly used in practice)

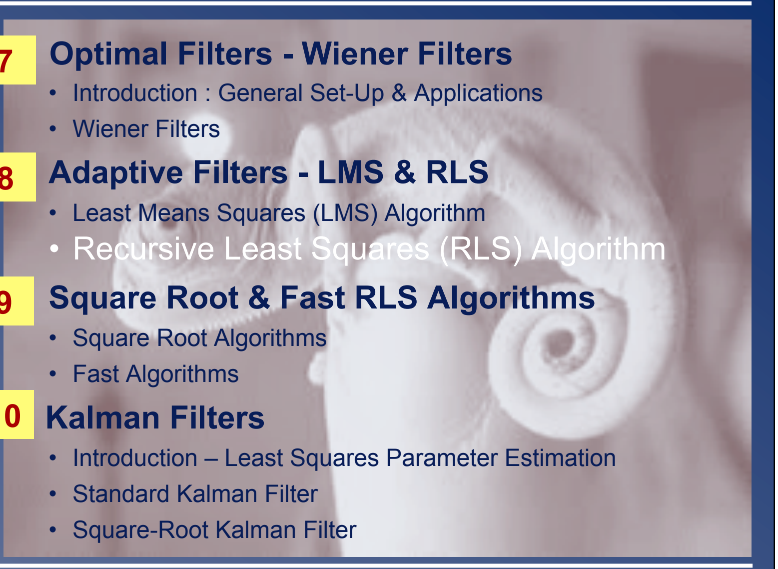
$$\mathbf{w}_{NLMS}[k] = \mathbf{w}_{NLMS}[k-1] + \frac{\bar{\mu}}{\alpha + \mathbf{u}_k^T.\mathbf{u}_k}.\mathbf{u}_k.(d_k - \mathbf{u}_k^T.\mathbf{w}_{NLMS}[k-1])$$

- NLMS (for $\bar{\mu} = 1$) also solves a specific *optimization problem*:

$$\min_{\mathbf{w}[k]} \tilde{J}(\mathbf{w}[k]) = \alpha.\left\|\mathbf{w}[k] - \mathbf{w}_{NLMS}[k-1]\right\|_2^2 + (\underbrace{d_k - \mathbf{u}_k^T.\mathbf{w}[k]}_{\text{'a posteriori error'}})^2$$

For instance with (normalized step size=1 and) α➔0, the NLMS solution at time k sets the a posteriori error to zero, with minimal change with respect to previous NLMS solution at time k-1

Part-III : Optimal & Adaptive Filters



Chapter-7 Optimal Filters - Wiener Filters
- Introduction : General Set-Up & Applications
- Wiener Filters

Chapter-8 Adaptive Filters - LMS & RLS
- Least Means Squares (LMS) Algorithm
- Recursive Least Squares (RLS) Algorithm

Chapter-9 Square Root & Fast RLS Algorithms
- Square Root Algorithms
- Fast Algorithms

Chapter-10 Kalman Filters
- Introduction – Least Squares Parameter Estimation
- Standard Kalman Filter
- Square-Root Kalman Filter

---

1. Least Squares (LS) Estimation

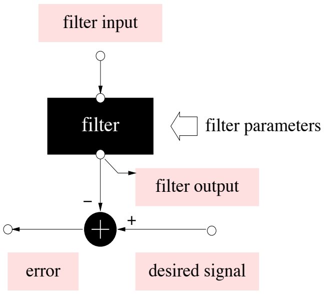
Prototype optimal/adaptive filter revisited

filter structure ?

→ FIR filters
(=pragmatic choice)

cost function ?

→ quadratic cost function
(=pragmatic choice)



1. Least Squares (LS) Estimation

Quadratic cost function

MMSE :

$$J_{MSE}(\mathbf{w}) = \mathrm{E}\left\{e_k^2\right\} = \mathrm{E}\left\{\left(d_k - y_k\right)^2\right\} = \mathrm{E}\left\{\left(d_k - \mathbf{u}_k^T\mathbf{w}\right)^2\right\}$$

Least-squares(LS) criterion :

if statistical info is not available, may use an alternative 'data-based' criterion...

$$J_{LS}(\mathbf{w}) = \sum_{l=1}^{k} e_l^2 = \sum_{l=1}^{k}\left(d_l - y_l\right)^2 = \sum_{l=1}^{k}\left(d_l - \mathbf{u}_l^T\mathbf{w}\right)^2$$

Interpretation? : see below

---

1. Least Squares (LS) Estimation

filter input sequence : $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \ldots \quad \mathbf{u}_k$

corresponding desired response sequence is : $d_1, d_2, d_3, \ldots, \; d_k$

$$\underbrace{\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_k \end{bmatrix}}_{\text{error signal } \mathbf{e}} = \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \end{bmatrix}}_{\mathbf{d}} - \underbrace{\begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \vdots \\ \mathbf{u}_k^T \end{bmatrix}}_{U} . \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_L \end{bmatrix}}_{\mathbf{w}}$$

cost function $J_{LS}(\mathbf{w}) = \sum_{l=1}^{k} e_l^2 = \|\mathbf{e}\|_2^2 = \|\mathbf{d} - U\mathbf{w}\|_2^2$

→ *linear least squares problem* : $\min_{\mathbf{w}} \|\mathbf{d} - U\mathbf{w}\|_2^2$

1. Least Squares (LS) Estimation

$$J_{LS}(\mathbf{w}) = \sum_{l=1}^{k} e_l^2 = \|\mathbf{e}\|_2^2 = \mathbf{e}^T.\mathbf{e} = \|\mathbf{d} - U\mathbf{w}\|_2^2$$

minimum obtained by setting gradient $= 0$:

$$0 = [\frac{\partial J_{LS}(\mathbf{w})}{\partial \mathbf{w}}]_{\mathbf{w}=\mathbf{w}_{LS}} = [\frac{\partial}{\partial \mathbf{w}}(\mathbf{d}^T\mathbf{d} + \mathbf{w}^T U^T U \mathbf{w} - 2\mathbf{w}^T U^T \mathbf{d})]_{\mathbf{w}=\mathbf{w}_{LS}}$$

$$= [2\underbrace{U^T U}_{\mathbb{X}_{uu}}\mathbf{w} - 2\underbrace{U^T \mathbf{d}}_{\mathbb{X}_{du}}]_{\mathbf{w}=\mathbf{w}_{LS}}$$

$\mathbb{X}_{uu} \cdot \mathbf{w}_{LS} = \mathbb{X}_{du}$ → $\mathbf{w}_{LS} = \mathbb{X}_{uu}^{-1}\mathbb{X}_{du}$

'Normal equations'
(L+1 equations in L+1 unknowns)

This is the
'Least Squares Solution'

---

1. Least Squares (LS) Estimation

Note : correspondences with Wiener filter theory ?

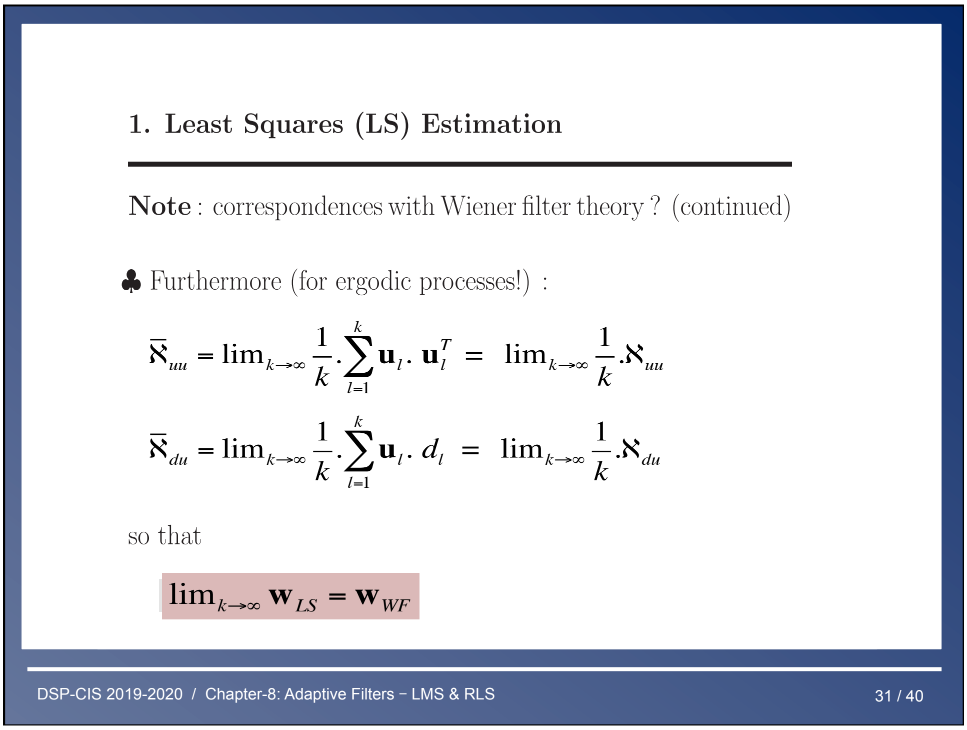
♣ estimate $\bar{\mathbb{X}}_{uu}$ and $\bar{\mathbb{X}}_{du}$ by time-averaging (ergodicity!)

$$\text{estimate}\{\bar{\aleph}_{uu}\} = \frac{1}{k}.\sum_{l=1}^{k}\mathbf{u}_l.\,\mathbf{u}_l^T = \frac{1}{k}.\;U^T U = \frac{1}{k}.\aleph_{uu}$$

$$\text{estimate}\{\bar{\aleph}_{du}\} = \frac{1}{k}.\sum_{l=1}^{k}\mathbf{u}_l.\,d_l = \frac{1}{k}.\;U^T\mathbf{d} = \frac{1}{k}.\aleph_{du}$$

leads to same optimal filter :

$$\text{estimate}\{\mathbf{w}_{WF}\} = (\tfrac{1}{k}\mathbb{X}_{uu})^{-1} \cdot (\tfrac{1}{k}\mathbb{X}_{du}) = \mathbb{X}_{uu}^{-1} \cdot \mathbb{X}_{du} = \mathbf{w}_{LS}$$



1. Least Squares (LS) Estimation

Note : correspondences with Wiener filter theory ? (continued)

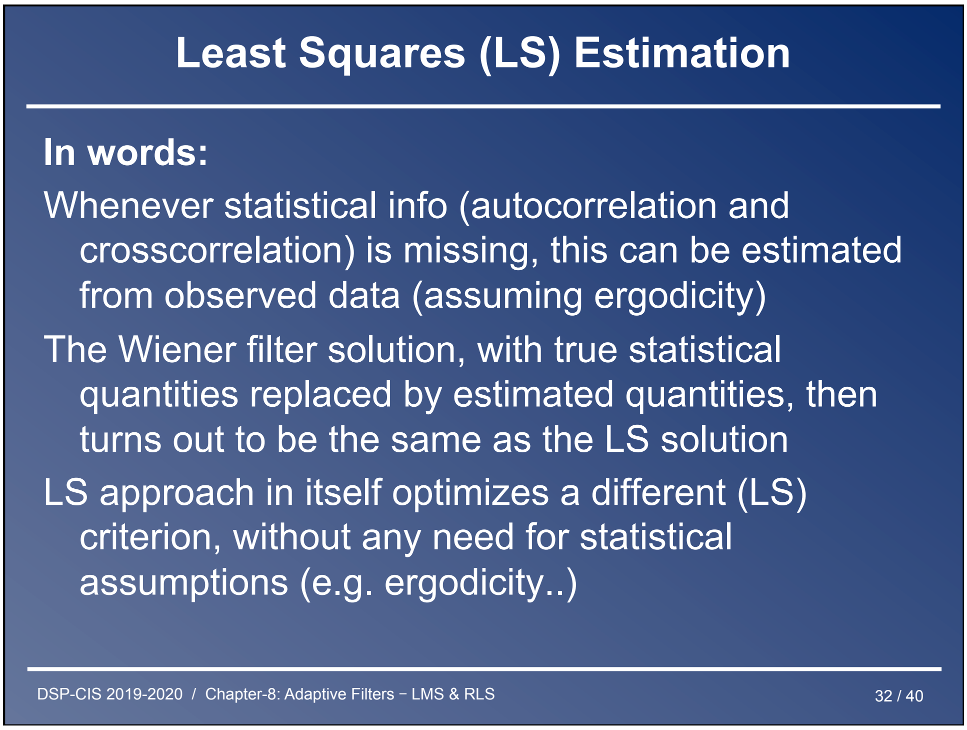
♣ Furthermore (for ergodic processes!) :

$$\overline{\aleph}_{uu} = \lim_{k\to\infty} \frac{1}{k}.\sum_{l=1}^{k} \mathbf{u}_l .\ \mathbf{u}_l^T \ = \ \lim_{k\to\infty} \frac{1}{k}.\aleph_{uu}$$

$$\overline{\aleph}_{du} = \lim_{k\to\infty} \frac{1}{k}.\sum_{l=1}^{k} \mathbf{u}_l .\ d_l \ = \ \lim_{k\to\infty} \frac{1}{k}.\aleph_{du}$$

so that

$$\lim_{k\to\infty} \mathbf{w}_{LS} = \mathbf{w}_{WF}$$



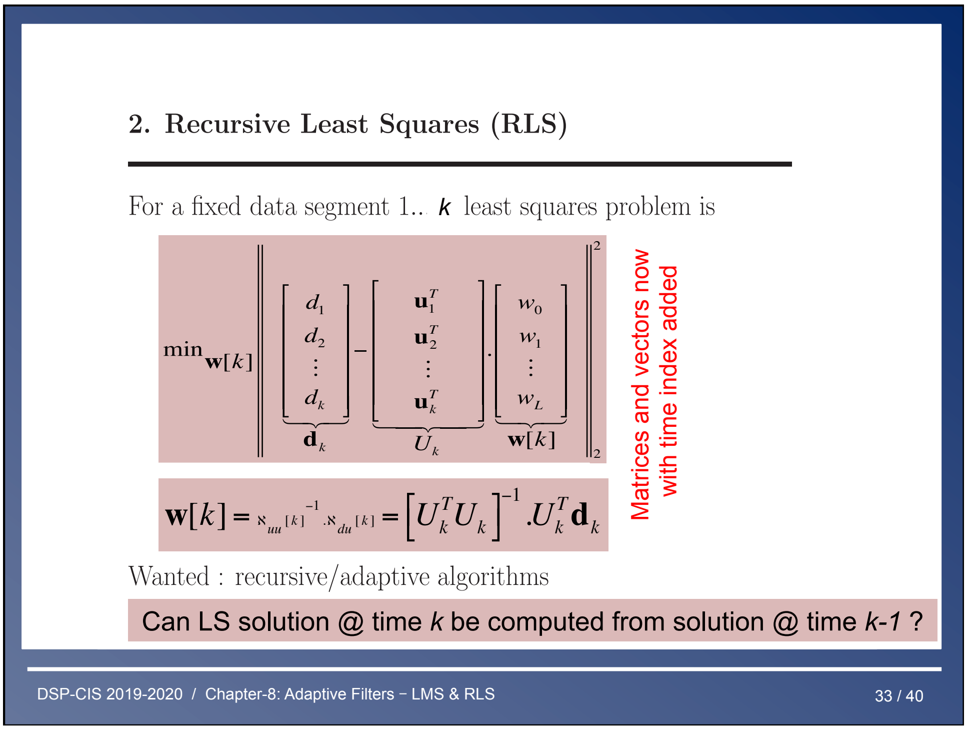
Least Squares (LS) Estimation

In words:

Whenever statistical info (autocorrelation and crosscorrelation) is missing, this can be estimated from observed data (assuming ergodicity)

The Wiener filter solution, with true statistical quantities replaced by estimated quantities, then turns out to be the same as the LS solution

LS approach in itself optimizes a different (LS) criterion, without any need for statistical assumptions (e.g. ergodicity..)



2. Recursive Least Squares (RLS)

For a fixed data segment 1.. *k* least squares problem is

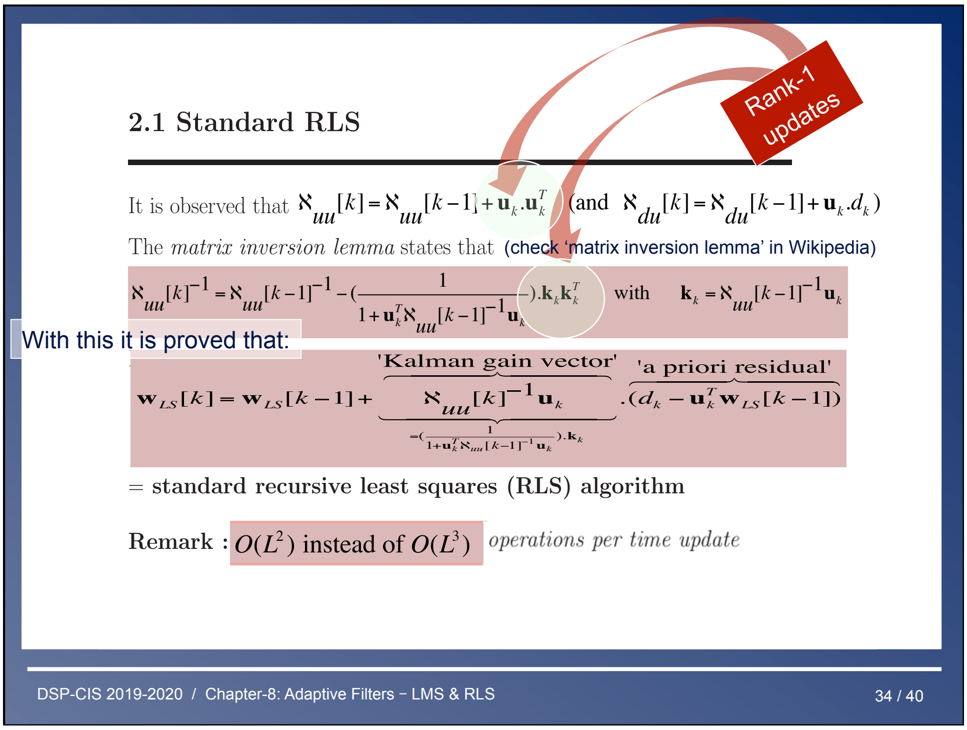
$$\min_{\mathbf{w}[k]} \left\| \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \end{bmatrix}}_{\mathbf{d}_k} - \underbrace{\begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \vdots \\ \mathbf{u}_k^T \end{bmatrix}}_{U_k} . \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_L \end{bmatrix}}_{\mathbf{w}[k]} \right\|_2^2$$

Matrices and vectors now with time index added

$$\mathbf{w}[k] = \aleph_{uu}[k]^{-1}.\aleph_{du}[k] = \left[U_k^T U_k\right]^{-1}.U_k^T \mathbf{d}_k$$

Wanted : recursive/adaptive algorithms

Can LS solution @ time *k* be computed from solution @ time *k-1* ?



2.1 Standard RLS

Rank-1 updates

It is observed that $\aleph_{uu}[k] = \aleph_{uu}[k-1] + \mathbf{u}_k.\mathbf{u}_k^T$ (and $\aleph_{du}[k] = \aleph_{du}[k-1] + \mathbf{u}_k.d_k$)

The *matrix inversion lemma* states that (check 'matrix inversion lemma' in Wikipedia)

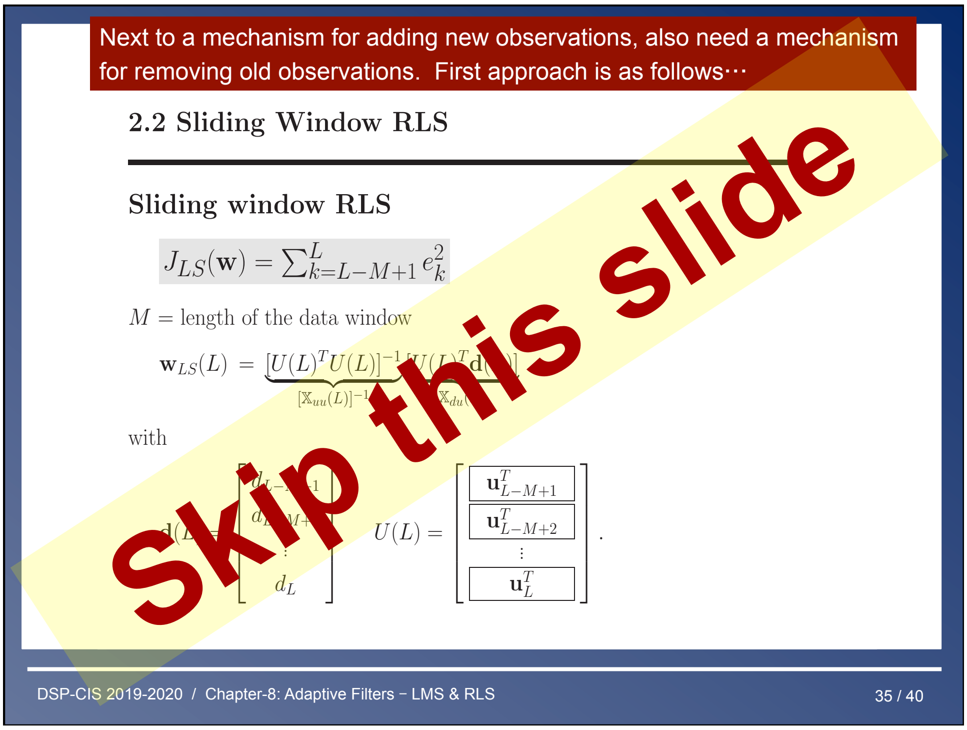
$$\aleph_{uu}[k]^{-1} = \aleph_{uu}[k-1]^{-1} - \left(\frac{1}{1+\mathbf{u}_k^T \aleph_{uu}[k-1]^{-1}\mathbf{u}_k}\right).\mathbf{k}_k\mathbf{k}_k^T \quad \text{with} \quad \mathbf{k}_k = \aleph_{uu}[k-1]^{-1}\mathbf{u}_k$$

With this it is proved that:

$$\mathbf{w}_{LS}[k] = \mathbf{w}_{LS}[k-1] + \overbrace{\underbrace{\aleph_{uu}[k]^{-1}\mathbf{u}_k}_{=\left(\frac{1}{1+\mathbf{u}_k^T\aleph_{uu}[k-1]^{-1}\mathbf{u}_k}\right).\mathbf{k}_k}}^{\text{'Kalman gain vector'}} . \overbrace{(d_k - \mathbf{u}_k^T \mathbf{w}_{LS}[k-1])}^{\text{'a priori residual'}}$$

= **standard recursive least squares (RLS) algorithm**

Remark : $O(L^2)$ instead of $O(L^3)$ *operations per time update*



Next to a mechanism for adding new observations, also need a mechanism for removing old observations. First approach is as follows⋯

2.2 Sliding Window RLS

Sliding window RLS

$$J_{LS}(\mathbf{w}) = \sum_{k=L-M+1}^{L} e_k^2$$

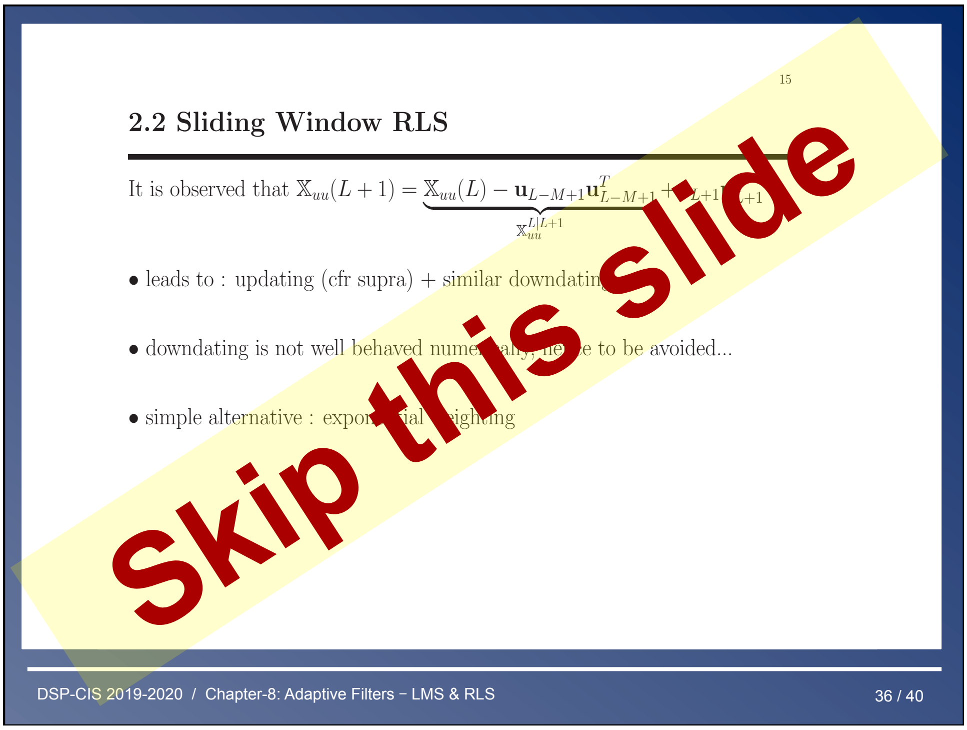
M = length of the data window

$$\mathbf{w}_{LS}(L) = \underbrace{[U(L)^T U(L)]^{-1}}_{[\mathbb{X}_{uu}(L)]^{-1}} \underbrace{[U(L)^T\mathbf{d}(\text{[illegible]})]}_{\mathbb{X}_{du}(\text{[illegible]}}$$

with

$$\mathbf{d}(L\text{[illegible]} \begin{bmatrix} d_{L-\text{[illegible]}1} \\ d_{L\text{[illegible]}M+\text{[illegible]}} \\ \vdots \\ d_L \end{bmatrix} \qquad U(L) = \begin{bmatrix} \boxed{\mathbf{u}_{L-M+1}^T} \\ \boxed{\mathbf{u}_{L-M+2}^T} \\ \vdots \\ \boxed{\mathbf{u}_L^T} \end{bmatrix}.$$

Skip this slide



15

2.2 Sliding Window RLS

It is observed that $\mathbb{X}_{uu}(L+1) = \underbrace{\mathbb{X}_{uu}(L) - \mathbf{u}_{L-M+1}\mathbf{u}_{L-M+1}^T}_{\mathbb{X}_{uu}^{L|L+1}} + \text{[illegible]}_{L+1}\text{[illegible]}_{L+1}$

- leads to : updating (cfr supra) + similar downdating
- downdating is not well behaved numerically, hence to be avoided...
- simple alternative : exponential weighting

Skip this slide

Next to a mechanism for adding new observations, also need a mechanism for removing old observations. Simpler approach is as follows…

2.3 Exponentially Weighted RLS

Exponentially weighted RLS : Goal is to give a smaller weight to 'older' data, i.e.

$$J_{LS}(\mathbf{w}) = \sum_{l=1}^{k} \lambda^{2(k-l)} e_l^2$$

$0 < \lambda < 1$ is *weighting factor* or *forget factor*

$\frac{1}{1-\lambda}$ is a 'measure of the memory of the algorithm'

Which leads to…

$$\min_{\mathbf{w}[k]} \left\| \underbrace{\begin{bmatrix} \lambda^{k-1} d_1 \\ \lambda^{k-2} d_2 \\ \vdots \\ \lambda^0 d_k \end{bmatrix}}_{\mathbf{d}_k} - \underbrace{\begin{bmatrix} \lambda^{k-1} \mathbf{u}_1^T \\ \lambda^{k-2} \mathbf{u}_2^T \\ \vdots \\ \lambda^0 \mathbf{u}_k^T \end{bmatrix}}_{U_k} . \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_L \end{bmatrix}}_{\mathbf{w}[k]} \right\|_2^2$$

$$\mathbf{w}[k] = \aleph_{uu}[k]^{-1} . \aleph_{du}[k] = \left[U_k^T U_k \right]^{-1} . U_k^T \mathbf{d}_k$$

2.3 Exponentially Weighted RLS

It is observed that $\aleph_{uu}[k] = \lambda^2 . \aleph_{uu}[k-1] + \mathbf{u}_k . \mathbf{u}_k^T$ (and $\aleph_{du}[k] = \lambda^2 . \aleph_{du}[k-1] + \mathbf{u}_k . d_k$)

hence

$$\aleph_{uu}[k]^{-1} = \frac{1}{\lambda^2} \aleph_{uu}[k-1]^{-1} - \left(\frac{1}{1 + \frac{1}{\lambda^2} \mathbf{u}_k^T \aleph_{uu}[k-1]^{-1} \mathbf{u}_k} \right) . \mathbf{k}_k \mathbf{k}_k^T \quad \text{with} \quad \mathbf{k}_k = \frac{1}{\lambda^2} \aleph_{uu}[k-1]^{-1} \mathbf{u}_k$$

$$\mathbf{w}_{LS}[k] = \mathbf{w}_{LS}[k-1] + \aleph_{uu}[k]^{-1} \mathbf{u}_k . (d_k - \mathbf{u}_k^T \mathbf{w}_{LS}[k-1])$$

i.e. exponential weighting hardly changes RLS formulas.. (easy!)

Recursive Least Squares (RLS) Algorithm

Computational Complexity:

Standard RLS algorithm (with exponential weighting) has **O(L²)** computational complexity per time update

Compare to **O(L)** for LMS (=cheaper, but slow convergence)

In Chapter-9, will present ‘Fast RLS’ algorithms with **O(L)** computational complexity (and without compromising convergence properties)

Recursive Least Squares (RLS) Algorithm

Numerical Analysis/Stability:

Standard RLS algorithm (even with exponential weighting) has been shown to have unstable quantization error propagation (in low-precision implementation)

In Chapter-9, will present ‘Square Root RLS’ algorithms which are shown to be perfectly stable numerically (without compromising complexity & convergence properties)