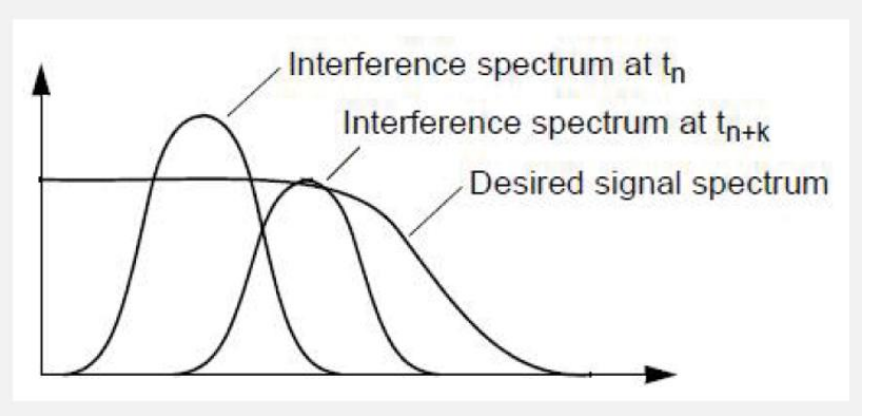
Adaptive algorithms

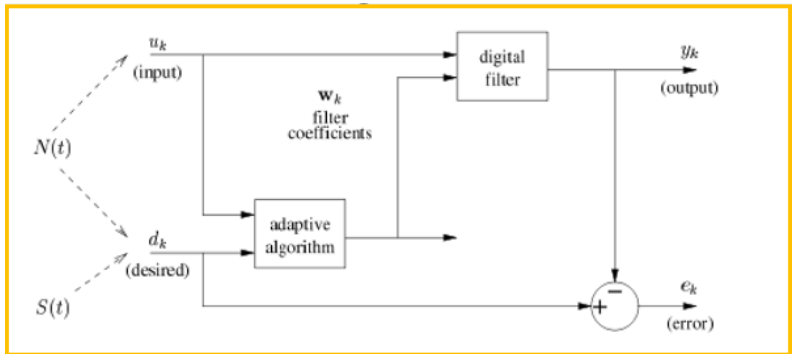
Signal Processing Systems Fall 2025

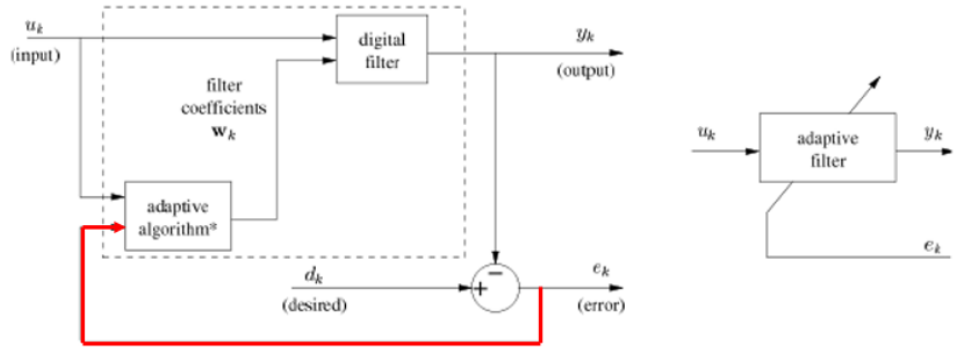
Lecture 13 (Monday 8.12.)

In previous lecture

- Why adaptive algorithms needed?
- General structure
- Error-based adaptation
- Configurations for applications
  - Noise cancellation
  - System identification
  - Inverse system modelling
  - Adaptive line enhancement





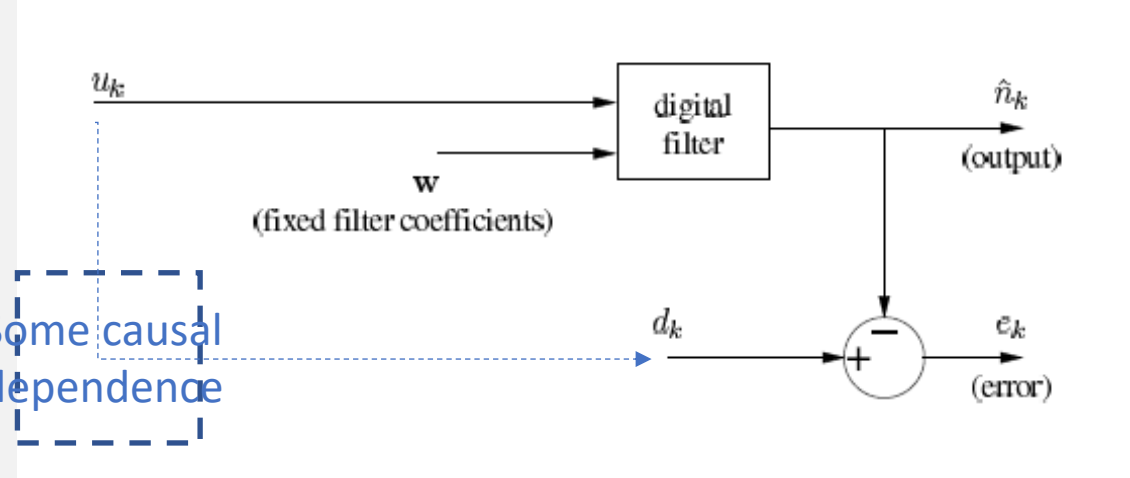


Today: common adaptive algorithms

- Wiener filter & steepest descent
- Least mean squares (LMS) algorithm
  - Derivation from steepest descent Wiener filter
  - Variants, LMS in frequency domain
- Recursive least squares (RLS) algorithm
  - Least squares regression model for filter coefficients
  - Recursive approach for adaptive processing

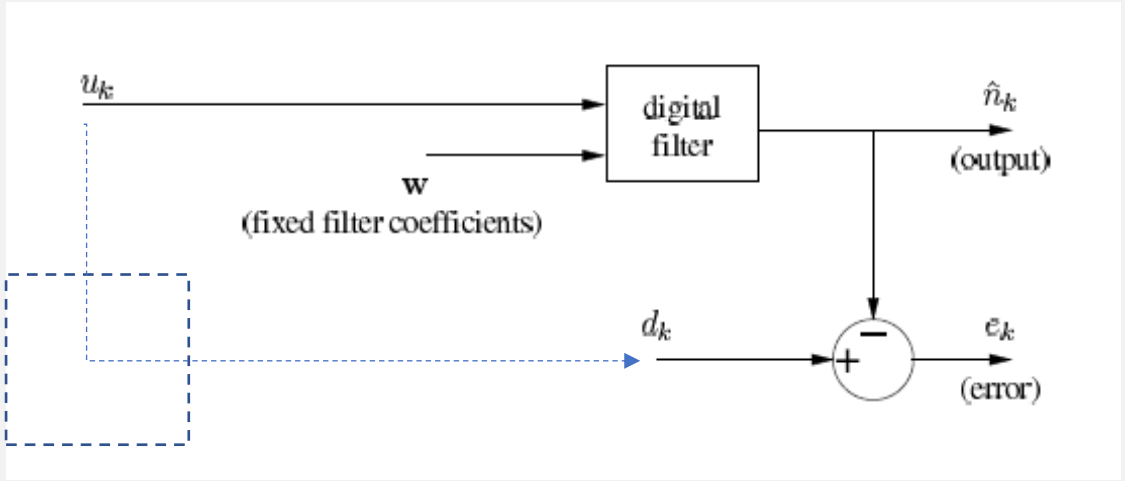
1. Basis: Wiener filter & steepest descent

- **Wiener filtering** is important to know as many adaptive algorithms can be considered as approximations of it
- Considers optimal choice of the filter parameters **w** to a filtering problem, whose structure is depicted on the right
- Optimality: Using u_k as the input, the filter provides an output n_k , that minimizes the mean value of the squared difference $(d_k - n_k)^2$



Wiener filter

- Optimal choice of the FIR filter coefficients **w** minimizes mean-squared error (MSE)
- Assuming that the input signals are wide-sense stationary (WSS), there is a closed-form solution for the coefficients, **Wiener-Hopf equation** (WHE)
- Note. This is not an adaptive filter as filter coefficients are precomputed using known fixed statistics of the inputs



WSS = mean and autocovariance do not vary with respect to time, 2nd moment finite for all times

Wiener-Hopf equation

$$\mathbf{w} = \mathbf{R}^{-1}\mathbf{p}$$

$$\mathbf{R} = \mathrm{E}\{\mathbf{u}_k \mathbf{u}_k^T\}$$ *L*x*L* autocorrelation matrix

$$\mathbf{p} = \mathrm{E}\{\mathbf{u}_k d_k\}$$ Length *L* cross-correlation vector

Steepest descent approach

- **Steepest descent** is a method for solving a system of equations **p** = **Rw** recursively
  - No inversion of matrix **R**
  - Can be used to determine Wiener filter coefficients (**w**)

1. Obtain auto- & cross-correlations

$$\begin{aligned} \mathbf{R} &= \mathrm{E}\{\mathbf{u}_k \mathbf{u}_k^T\} \\ \mathbf{p} &= \mathrm{E}\{\mathbf{u}_k d_k\} \end{aligned}$$

2. Apply

Steepest descent

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \upsilon \underbrace{2(\mathbf{p} - \mathbf{R}\mathbf{w}_i)}_{=\nabla J(\mathbf{w}_i)}$$

$\mathbf{w}_i \rightarrow \mathbf{w}$ of WHE as $i \rightarrow \infty$.

Gradient of the MSE cost function

Where ϑ is a coefficient, which controls the magnitude of updates.

Steepest descent by example

Example. Two filter parameters (L = 2)

Cost function minimized by Wiener filter solution is

$$J(\mathbf{w}) = \mathrm{E}\{(d_k - n_k)^2\} = \mathrm{E}\left\{\left(d_k - \mathbf{u}_k^T\mathbf{w}\right)^2\right\}$$

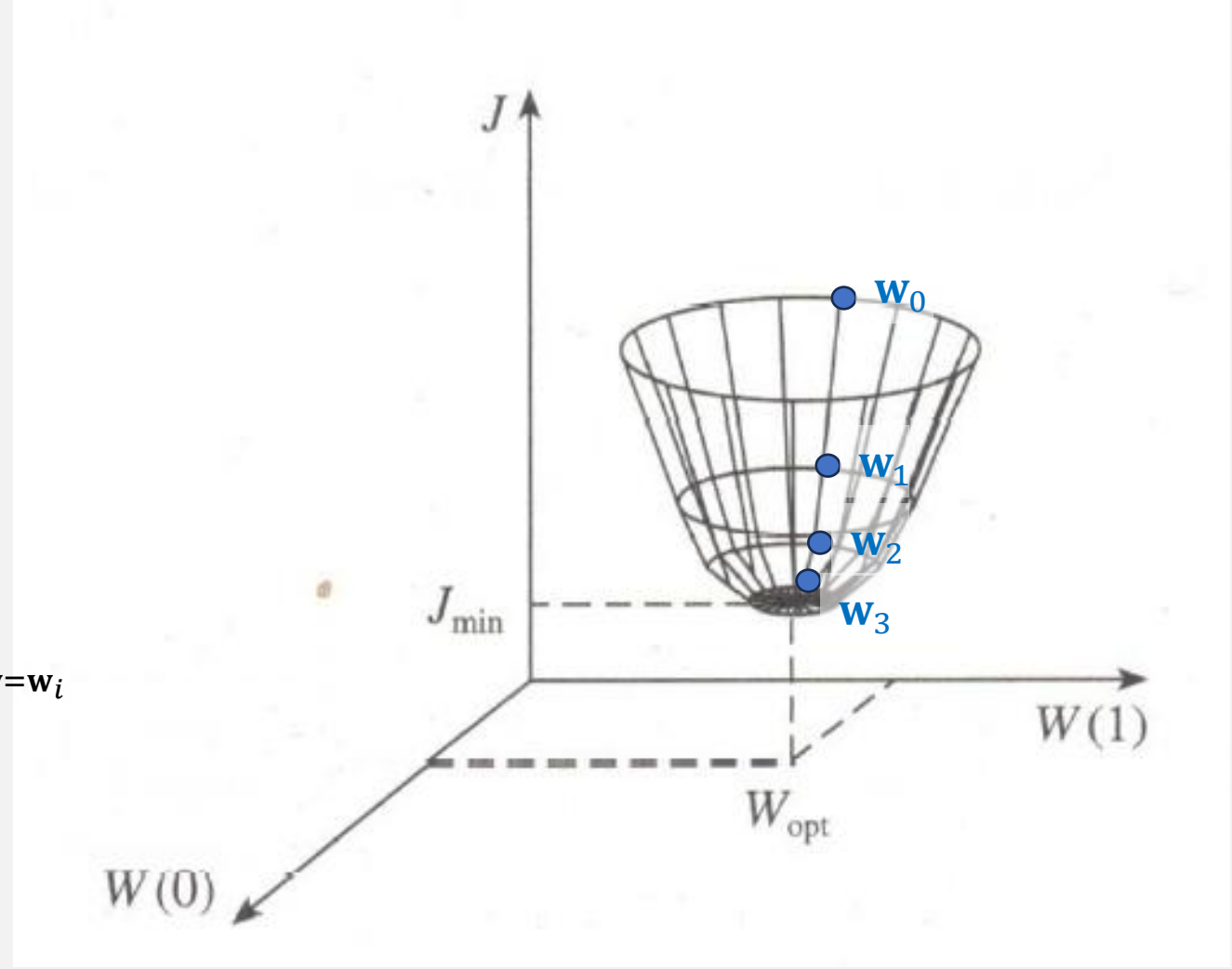
where $\mathrm{E}\{*\}$ denotes expected value.

The gradient of the cost function at $\mathbf{w}_i$, $\nabla J(\mathbf{w}_i)$, is

$$\left.\frac{\partial \mathrm{E}\left\{\left(d_k - \mathbf{u}_k^T\mathbf{w}\right)^2\right\}}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_i} = \left.\mathrm{E}\left\{2\mathbf{u}_k(d_k - \mathbf{u}_k^T\mathbf{w})\right\}\right|_{\mathbf{w}=\mathbf{w}_i}$$

$$= 2\left(\underbrace{\mathrm{E}\{\mathbf{u}_k d_k\}}_{\mathbf{p}} - \underbrace{\mathrm{E}\left\{\mathbf{u}_k\mathbf{u}_k^T\right\}}_{\mathbf{R}}\mathbf{w}_i\right)$$

Next step in iteration is $\upsilon\nabla J(\mathbf{w}_i)$



[Ifeachor&Jervis, Chapter 10.3]

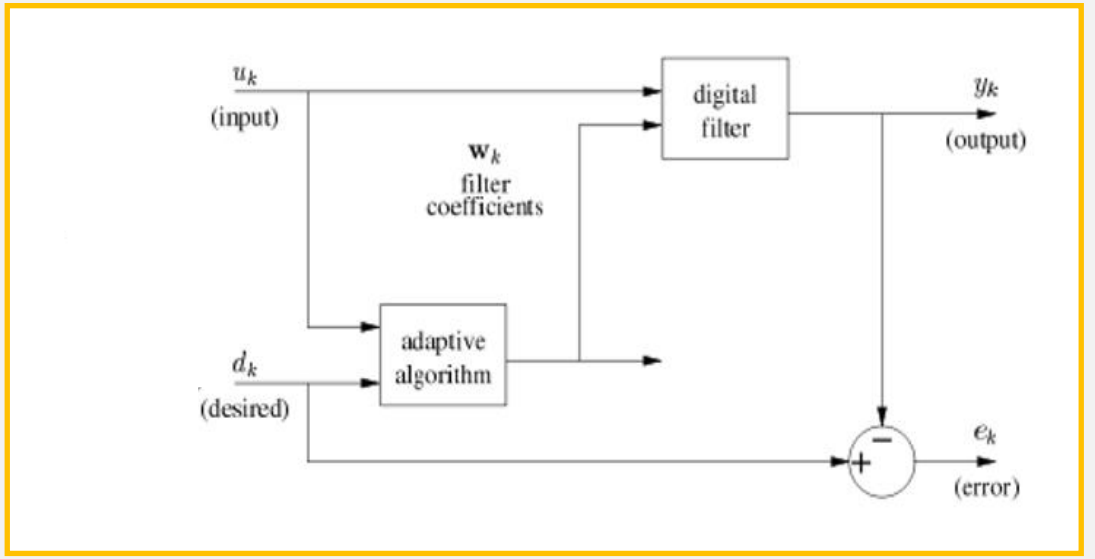
- Thus, we have two approaches for computing coefficients of the Wiener filter
  (1) Direct solution of the original equations, involving matrix inversion
  (2) Steepest descent based approach, where matrix inversion is avoided by recursive computation
- Both are based on having statistics
- Statistics assumed time-invariant

$$\mathbf{R} = \mathrm{E}\{\mathbf{u}_k \mathbf{u}_k^T\}$$
$$\mathbf{p} = \mathrm{E}\{\mathbf{u}_k d_k\}$$

- Next, considering situation when statistics are time-varying …

Adaptive variant of WF (1)

- Adaptive algorithm estimates the autocorrelation $\mathbf{R}$ and cross-correlation $\mathbf{p}$ at run time and then solves WHE
  - Estimates are time-varying
  - Denoted by $\mathbf{R}_k$ and $\mathbf{p}_k$
  - Time-varying filter coefficients $\mathbf{w}_k$ obtained
  - Run-time variation of signal statistics should be sufficiently slow
- The approach requires
  - Buffering of signal samples
  - Matrix inversion
- Note: this is **not error-based** adaptation
  - Adaptation is based on estimation of correlation statistics



$$\mathbf{w}_k = \mathbf{R}_k^{-1}\mathbf{p}_k$$

$$\boldsymbol{R_k} = \frac{1}{N}\sum_{j=0}^{N-1}\mathbf{u}_{k-j}\mathbf{u}_{k-j}^T$$

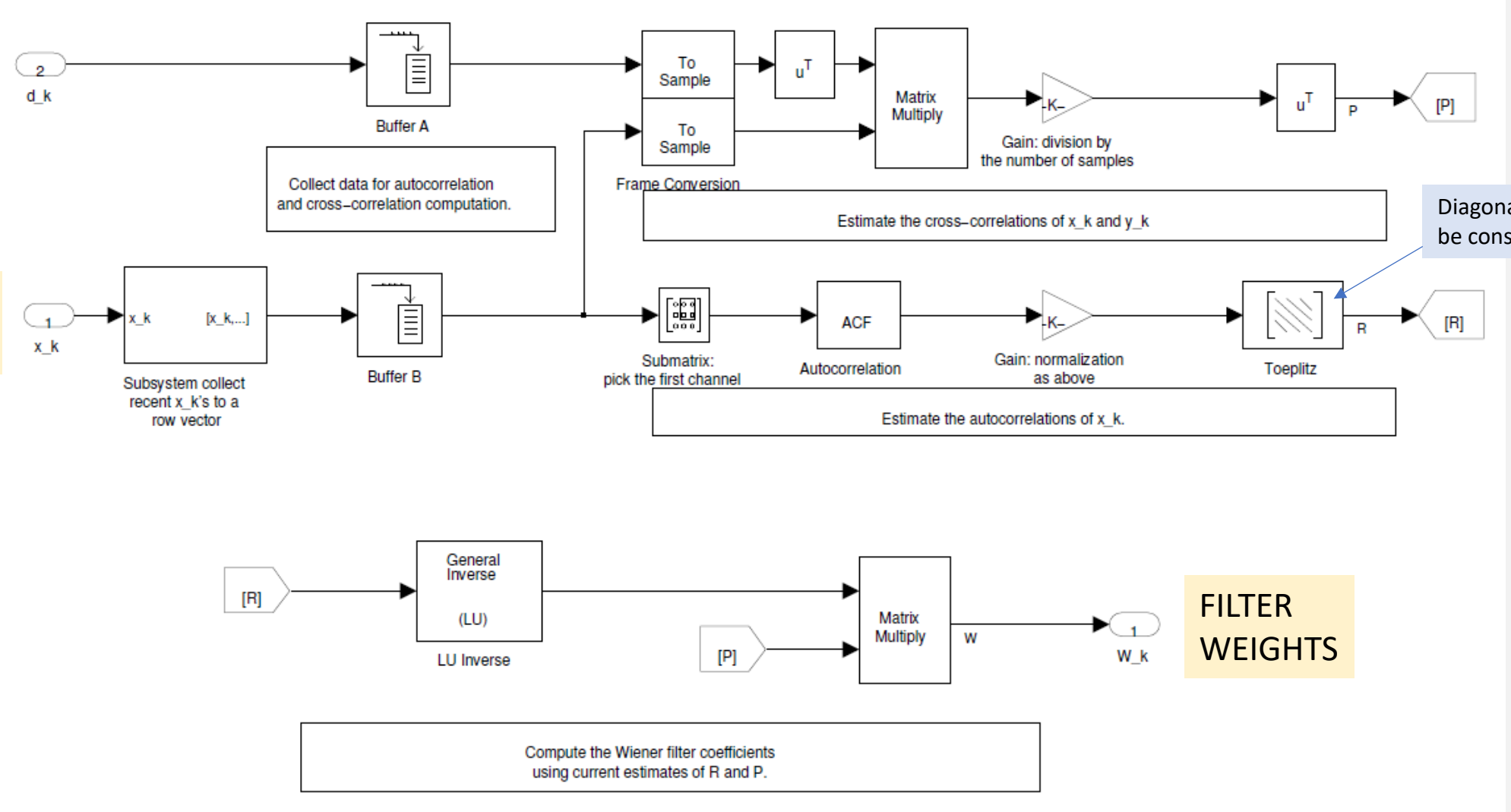
$$\boldsymbol{p_k} = \frac{1}{N}\sum_{j=0}^{N-1}\mathbf{u}_{k-j}d_{k-j}$$

Adaptive Wiener filter: a Simulink model

wienerfilt.slx in Moodle

PRIMARY INPUT

REFERENCE INPUT



Adaptive variant of WF (2)

- Based on steepest descent method
- Application as an adaptive algorithm
  - Statistics replaced by time varying estimates: $\mathbf{p} \leftarrow \mathbf{p}_k, \mathbf{R} \leftarrow \mathbf{R}_k$
  - If one iteration per time instant, iteration index i is also substituted by time index k.
  - However, there can be several steepest descent iterations at each time step
- Algorithm:

For each time instant *k*,
Update estimates of statistics **p** and **R**
Perform one or more SD iterations to update filter weights **w**

Steepest descent

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \upsilon \underbrace{2(\mathbf{p} - \mathbf{R}\mathbf{w}_i)}_{=\nabla J(\mathbf{w}_i)}$$

$\mathbf{w}_i \rightarrow \mathbf{w}$ of WHE as $i \rightarrow \infty$.

Gradient of the MSE cost function

$$\mathbf{R} = \mathrm{E}\{\mathbf{u}_k \mathbf{u}_k^T\}$$
$$\mathbf{p} = \mathrm{E}\{\mathbf{u}_k d_k\}$$

2. Least mean squares (LMS) algorithm

i is SD iteration index
k is time index

Is based on steepest descent solution of Wiener filter

Derivation from steepest descent:

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \mu 2(\mathbf{p} - \mathbf{R}\mathbf{w}_i) \quad \textbf{SD}$$

(1) Substitutions: $i \leftarrow k, \mathbf{p} \leftarrow \mathbf{p}_k, \mathbf{R} \leftarrow \mathbf{R}_k$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mu 2(\mathbf{p_k} - \mathbf{R}_k\mathbf{w}_k)$$

(2) Set N = 1 in $\boldsymbol{R_k} = \frac{1}{N}\sum_{j=0}^{N-1} \mathbf{u}_{k-j}\mathbf{u}_{k-j}^T$, $\boldsymbol{p_k} = \frac{1}{N}\sum_{j=0}^{N-1} \mathbf{u}_{k-j}d_{k-j}$

$\Rightarrow \mathbf{p}_k = \mathbf{u}_k d_k$ and $\mathbf{R}_k = \mathbf{u}_k\mathbf{u}_k^T$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mu 2(\mathbf{u_k}d_{\boldsymbol{k}} - \mathbf{u}_k\mathbf{u}_k^T\mathbf{w}_k)$$

(3) $\mathbf{u}_k^T\mathbf{w}_k$ is equal to the filter output y_k.

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mu 2(\mathbf{u}_k d_k - \mathbf{u}_k y_k)$$

(4) Take $\mathbf{u}_k$ out from parentheses.

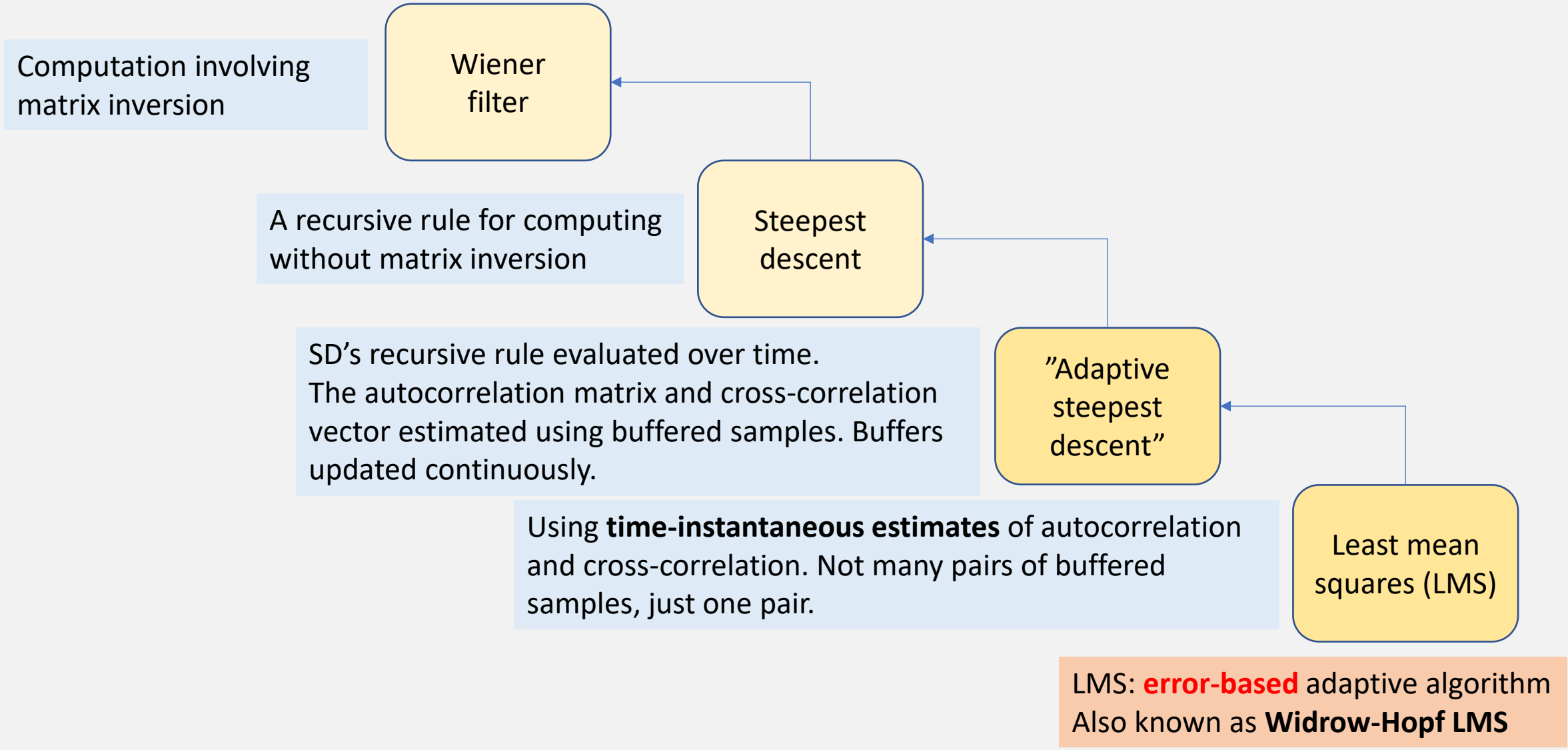
$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mu 2\mathbf{u}_k(d_k - y_k)$$

(5) $(d_k - y_k)$ is equal to the error output

$$\mathbf{w}_{k+1} = \mathbf{w}_k + 2\mu\mathbf{u_k}e_k \quad \textbf{LMS}$$

Result: an error-based adaptive algorithm, LMS

From Wiener filter to LMS

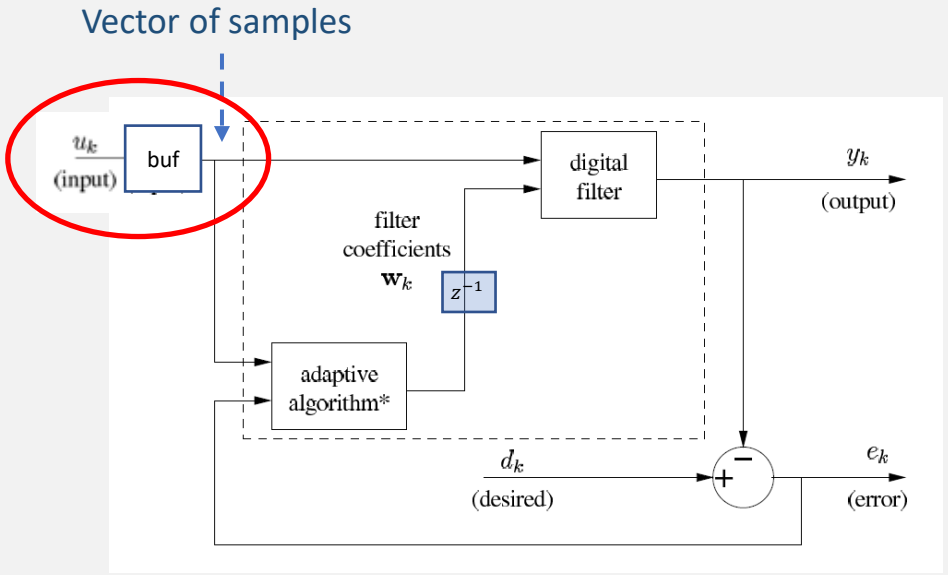


LMS operation

For each time step *k*:

1. Update input buffer with a sample u_k (discard oldest sample)

$$(\mathbf{u}_{k-1}, u_k) \rightarrow \mathbf{u}_k$$



Error-based adaptation

LMS operation

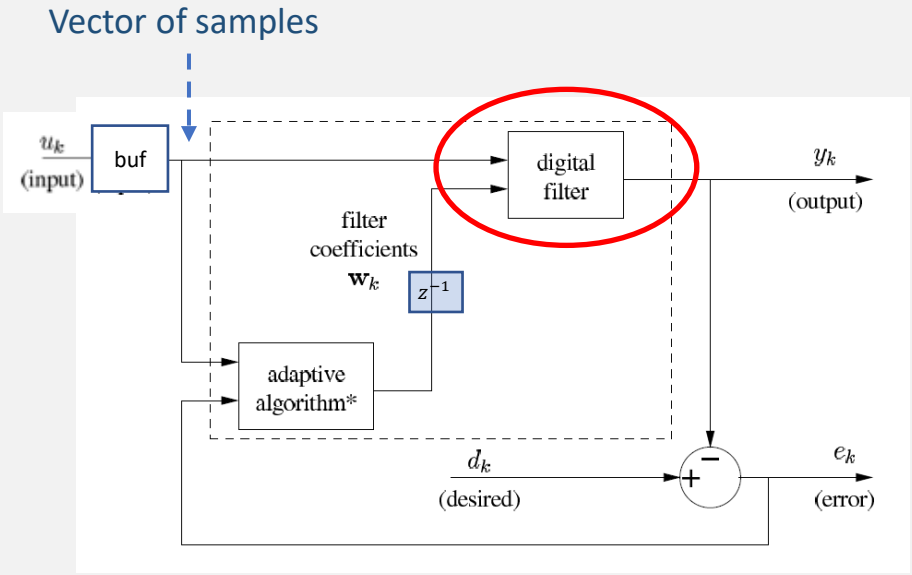
For each time step *k*:

1. Update input buffer with a sample u_k (discard oldest sample)

$$(\mathbf{u}_{k-1}, u_k) \rightarrow \mathbf{u}_k$$

2. Execute the digital filter (output y_k)

$$y_k = \mathbf{w}_k^T \mathbf{u}_k$$



Error-based adaptation

LMS operation

For each time step *k*:

1. Update input buffer with a sample u_k (discard oldest sample)

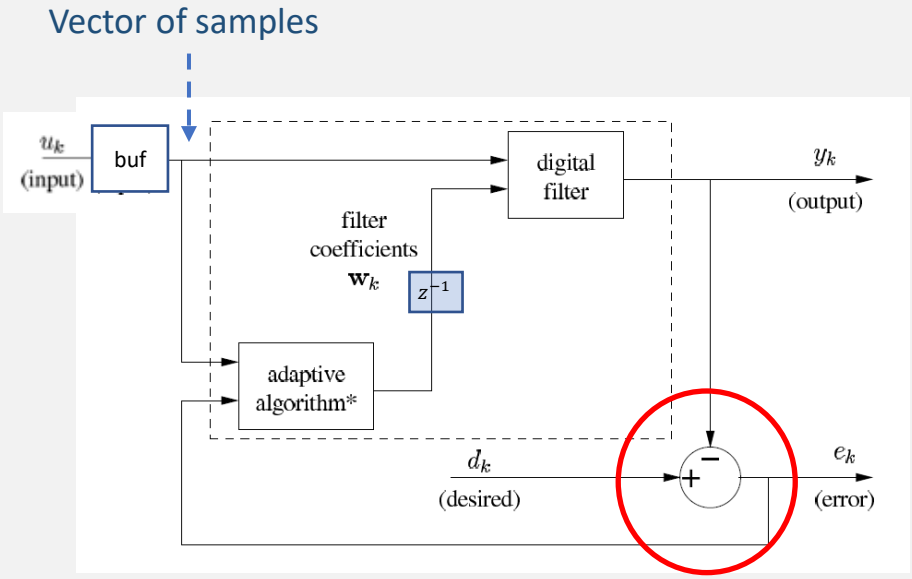
$$(\mathbf{u}_{k-1}, u_k) \rightarrow \mathbf{u}_k$$

2. Execute the digital filter (output y_k)

$$y_k = \mathbf{w}_k^T \mathbf{u}_k$$

3. Using the current value of d_k, compute the error e_k

$$e_k = d_k - y_k$$



Error-based adaptation

LMS operation

For each time step *k*:

1. Update input buffer with a sample u_k (discard oldest sample)

$$(\mathbf{u}_{k-1}, u_k) \rightarrow \mathbf{u}_k$$

2. Execute the digital filter (output y_k)

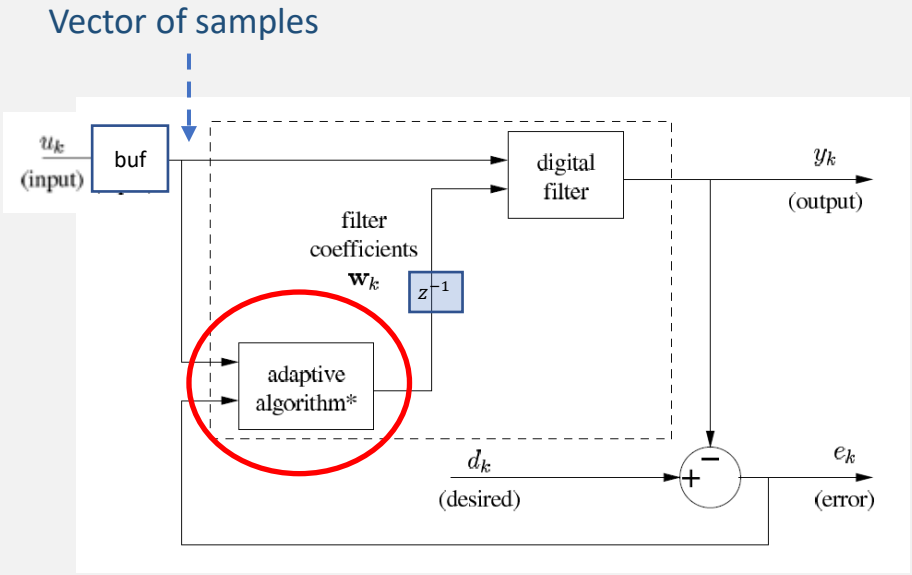
$$y_k = \mathbf{w}_k^T \mathbf{u}_k$$

3. Using the current value of d_k, compute the error e_k

$$e_k = d_k - y_k$$

4. Update the filter coefficients ($\mathbf{w}_{k+1}$)

$$\mathbf{w}_{k+1} = \mathbf{w}_k + 2\mu e_k \mathbf{u}_k$$



Error-based adaptation

LMS operation

For each time step *k*:

1. Update input buffer with a sample u_k (discard oldest sample)

$$(\mathbf{u}_{k-1}, u_k) \rightarrow \mathbf{u}_k$$

2. Execute the digital filter (output y_k)

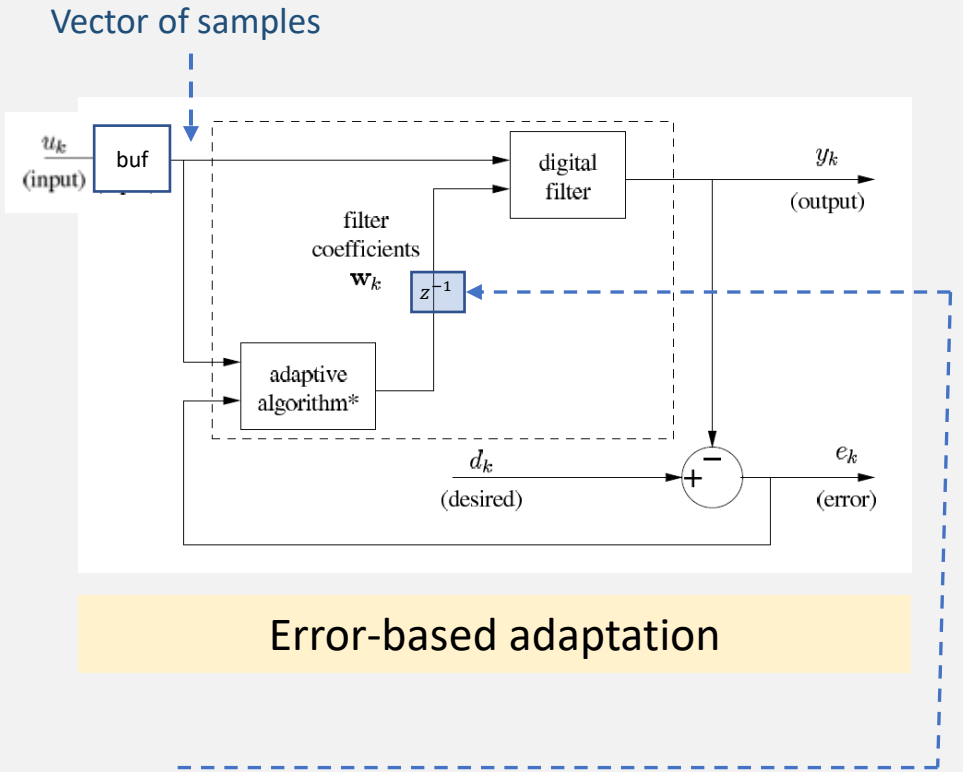
$$y_k = \mathbf{w}_k^T \mathbf{u}_k$$

3. Using the current value of d_k, compute the error e_k

$$e_k = d_k - y_k$$

4. Update the filter coefficients ($\mathbf{w}_{k+1}$)

$$\mathbf{w}_{k+1} = \mathbf{w}_k + 2\mu e_k \mathbf{u}_k$$



A unit delay between the adaptive algorithm and filter emphasizes that the filter coefficients for the current iteration are computed in the previous iteration.

LMS stability and performance

- In the rule $w_{k+1} = w_k + 2\mu\mathbf{u}_k e_k$, μ is a **parameter*** that defines the adaptation rate of the algorithm
  - The parameter cannot be arbitrarily large as then the algorithm will not converge properly
  - There are some guidelines for selecting this value
- **Excess MSE**: noise at the filter output due to use of estimated MSE cost gradient $-2\mathbf{u}_k e_k$
  - Suggests using small values for μ and filter length
- In general, experimental simulation work needed. Use of LMS variants can also help

Guidelines for setting μ:

(1) Should be below $1/\lambda_{max}$ where λ_{max} is the maximum eigenvalue of the autocorrelation matrix **R** (Widrow&al 1975)

(2) Should be below $1/LP_u$, where L is the filter length and P_u is the power of the reference input (Kuo&Gan 2005).

$$\text{Excess MSE} = \mu L P_u \xi_{\min}$$

Where $\xi_{\min}$ is the minimum MSE at the optimum solution in steady state

*Note that **in the Simulink's LMS block, the step-size parameter corresponds to 2μ**, not μ.

LMS variants

- **Normalized LMS**: technique for optimizing the speed of convergence in applications where the power of the input signal u_k varies.
  - Update equation is the same as in basic LMS, the difference is in the choice of the step-size parameter
  - Estimated reference signal power $\hat{P}_k$ affects the choice of the step size

- **Leaky LMS**: a leakage factor ν (< 1) is added to the update equation in order to avoid potential divergence and overflow of filter coefficients
  - Another solution is reduction of the amplitude of the primary input d_k, which leads to reduced gain demands for filter coefficients

$$y_k = \mathbf{w}_k^T \mathbf{u}_k \,, e_k = d_k - y_k$$

lower d_k => lower y_k => lower $\mathbf{w}_k$ elements

- **LMS algorithms with sign function**: the idea is to reduce the number of multiplications (e.g. in ASIC implementations)

Step size:

$$\mu_k = \frac{\alpha}{L\hat{P}_k}$$

Power estimators:

$$\hat{P}_k = \mathbf{u}_k^T \mathbf{u}_k / L$$

$$\hat{P}_k = (1-\beta)\hat{P}_{k-1} + \beta u_k^2$$

Combined formula, with numerical instability prevention:

$$\mu_k = \frac{\alpha}{\mathbf{u}_k^T \mathbf{u}_k + \varepsilon}$$

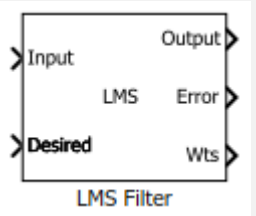
$$\mathbf{w}_{k+1} = \nu \mathbf{w}_k + 2\mu e_k \mathbf{u}_k$$

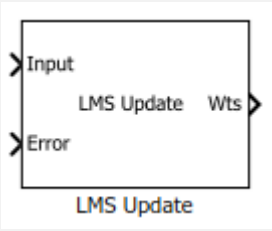
$$\mathbf{w}_{k+1} = \mathbf{w}_k + 2\mu \ \mathrm{sgn}(e_k)\ \mathbf{u}_k$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + 2\mu e_k \ \mathrm{sgn}(\mathbf{u}_k)$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + 2\mu \ \mathrm{sgn}(e_k)\ \mathrm{sgn}(\mathbf{u}_k)$$

LMS related blocks in Simulink





Four blocks can found from DSP System Toolbox.

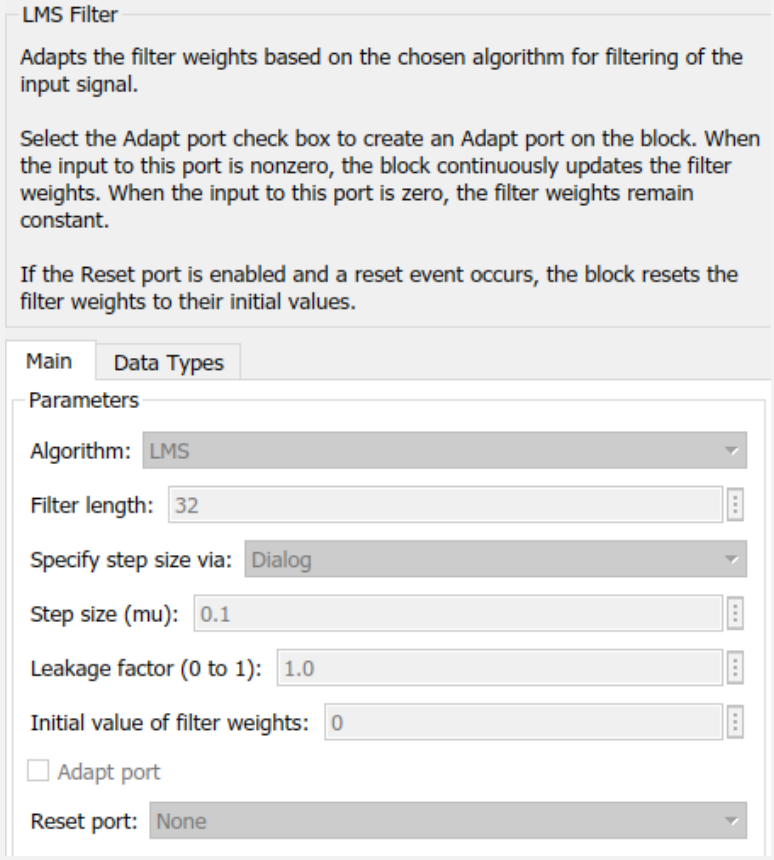
The basic blocks are **LMS** and **LMS Update**. The latter one just computes weights and can be used in configurations with reference and error signals (the block provides input to a FIR filter block).

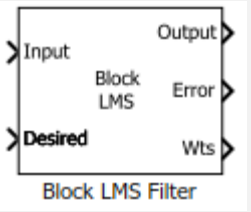
Available algorithms here are LMS, Normalized LMS, and LMS variants using sign function.

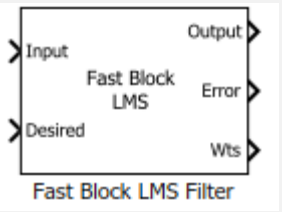
Leakage factor can be specified in order to simulate Leaky LMS.

Block LMS processes data in blocks. Filter parameters are updated once for each block.

Fast Block LMS uses FFT for fast convolution and is useful in the case of long filter lengths.







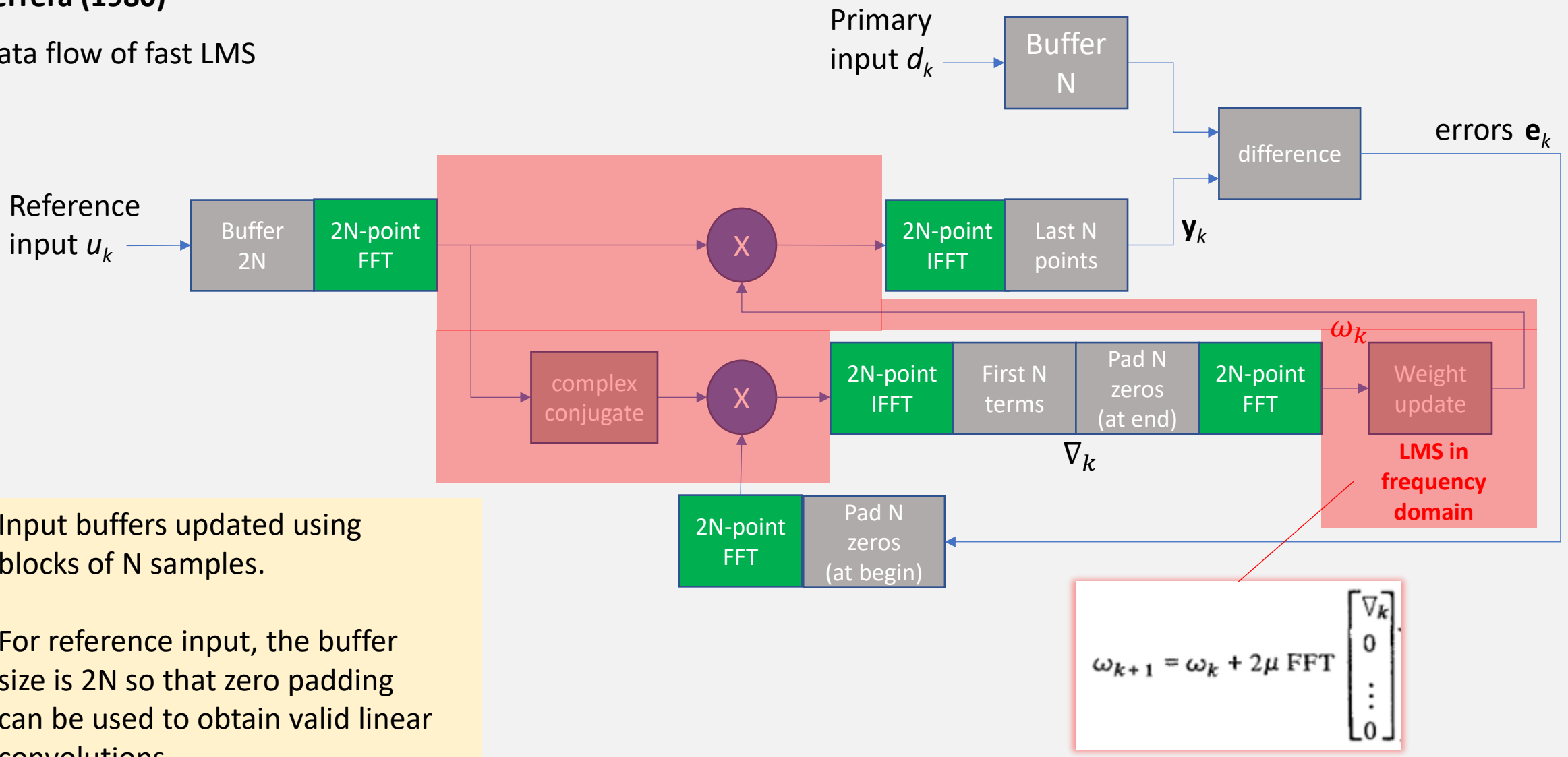
Fast LMS algorithms

- The idea is to process data in blocks and utilize FFT/IFFT in computations
- Useful for long filter lengths. There may be gain already at the block size of 64 points
- Many algorithms, a basic solution is provided in the paper by Ferrera (1980)

Ferrera (1980) *Fast implementation of LMS adaptive filters*. IEEE Trans. ASSP 28: 474-475.

Ferrera (1980)

Data flow of fast LMS



$$\omega_{k+1} = \omega_k + 2\mu \, \text{FFT} \begin{bmatrix} \nabla_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Input buffers updated using blocks of N samples.

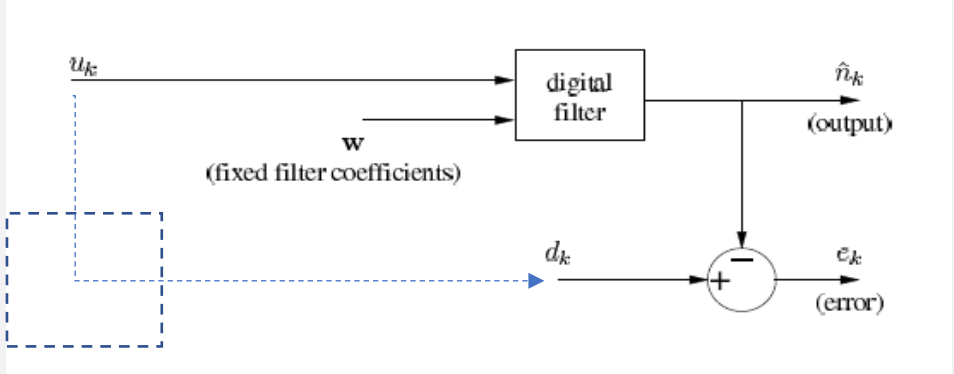
For reference input, the buffer size is 2N so that zero padding can be used to obtain valid linear convolutions

LMS operates in frequency domain.

3. Recursive Least Squares (RLS) algorithm

- **Basis**: least-squares (LS) method for estimating optimal filter coefficients
- Regression model for the primary input

$$d_k = \mathbf{u}_k^T \mathbf{w} + e_k$$



- Least squares estimator based on the samples from time instants $k \in \{0, \ldots, l-1\}$

LS estimator:

$$\hat{\mathbf{w}}_l = [\mathbf{U}_l^T \mathbf{U}_l]^{-1} \mathbf{U}_l^T \mathbf{d}_l$$

where $\mathbf{U}_l = [\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{l-1}]^T$

and $\mathbf{d}_l = [d_0, d_1, \ldots, d_{l-1}]^T$.

LS estimator

- Assume that we have N samples of d_k
  - $N \geq L$
  - $k = 0,1, \ldots, N-1$
- Assume that we have $N + L - 1$ samples of u_k so that we can express the regression model for each d_k.
- Assume that filter coefficients are fixed ($\mathbf{w}_k = \mathbf{w}$). We have

L	The number of filter coefficients
d_k	The *primary* input signal (also called the desired signal)
e_k	The error signal, the difference between d_k and y_k
u_k	The *reference* input signal (input of the digital filter)
$\mathbf{u}_k$	Vector of input signal values, $[\, u_k,\ u_{k-1},\ \ldots,\ u_{k-L+1}\,]^T$
$w_k(i)$	Filter coefficient i (weight i)
$\mathbf{w}_k$	The vector of filter coefficients, $[\, w_k(0),\ w_k(1),\ \ldots,\ w_k(L-1)\,]^T$
y_k	The filter output signal, equal to $\mathbf{w}_k^T \mathbf{u}_k$

$$d_0 = [u_0 \quad u_{-1} \quad \cdots \quad u_{0-(L-1)}]\boldsymbol{w} + e_0 = \mathbf{u}_0^T\boldsymbol{w} + e_0$$

$$d_1 = [u_1 \quad u_0 \quad \cdots \quad u_{1-(L-2)}]\boldsymbol{w} + e_1 = \mathbf{u}_1^T\boldsymbol{w} + e_1$$

$$d_{N-1} = [u_{N-1} \quad u_{N-2} \quad \cdots \quad u_{N-(L-1)}]\boldsymbol{w} + e_{N-1} = \mathbf{u}_{N-1}^T\boldsymbol{w} + e_{N-1}$$

LS estimator

$$
\begin{aligned}
d_0 &= [u_0 \quad u_{-1} \quad \cdots \quad u_{0-(L-1)}]\boldsymbol{w} + e_0 &&= \mathbf{u}_0^T\mathbf{w} + e_0 \\
d_1 &= [u_1 \quad u_0 \quad \cdots \quad u_{1-(L-2)}]\boldsymbol{w} + e_1 &&= \mathbf{u}_1^T\mathbf{w} + e_1 \\
&\vdots &&\vdots \\
d_{N-1} &= [u_{N-1} \quad u_{N-2} \quad \cdots \quad u_{N-(L-1)}]\boldsymbol{w} + e_{N-1} &&= \mathbf{u}_{N-1}^T\mathbf{w} + e_{N-1}
\end{aligned}
$$

Can be written in vector/matrix form

$$
\underbrace{\begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{N-1} \end{bmatrix}}_{\mathbf{d}} = \underbrace{\begin{bmatrix} \mathbf{u}_0^T \\ \mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_{N-1}^T \end{bmatrix}}_{\mathbf{U}} \mathbf{w} + \underbrace{\begin{bmatrix} e_0 \\ e_1 \\ \vdots \\ e_{N-1} \end{bmatrix}}_{\mathbf{e}}
$$

The sum of squared errors, $\sum_{n=0}^{N-1} e_n^2 = \mathbf{e}^T\mathbf{e}$, is minimized by coefficients $\mathbf{w}$ estimated using

$$\widehat{\mathbf{w}} = (\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{d}$$

$$\boldsymbol{d} = \mathbf{U}\mathbf{w}$$

$$\mathbf{U}^T\mathbf{d} = \mathbf{U}^T\mathbf{U}\mathbf{w}$$

$$(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{d} = (\mathbf{U}^T\mathbf{U})^{-1}(\mathbf{U}^T\mathbf{U})\mathbf{w}$$

$$(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{d} = \mathbf{w}$$

Recursive LS

- We might consider using LS estimator so that always, when we get new sample of primary and reference inputs, we setup equations and solve them
  - **Growing** matrices and vectors: it is obvious that such an approach cannot be used
  - Also one should be **adaptive** to current conditions and neglect old samples in estimation

- But if we use just *N* past samples of the primary input and reference input vector, there is still need for **matrix inversion** in the LS estimator

- Due to these reasons, recursive LS solution is introduced
  - For derivation, see e.g. ”Recursive least squares filter” in Wikipedia
  - The rule shown on the right updates weights according to the new observation
  - Parameter $\gamma \leq 1$ controls how much weight is given for the old solution

LS estimator:

$$\hat{\mathbf{w}}_l = [\mathbf{U}_l^T \mathbf{U}_l]^{-1} \mathbf{U}_l^T \mathbf{d}_l$$

where $\mathbf{U}_l = [\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{l-1}]^T$

and $\mathbf{d}_l = [d_0, d_1, ..., d_{l-1}]^T$.

Weight update rule in RLS:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{G}_k e_k$$

where $\mathbf{G}_k = \frac{\mathbf{P}_{k-1}\mathbf{u}_k}{\alpha_k}$

$$\mathbf{P}_k = \frac{1}{\gamma}(\mathbf{P}_{k-1} - \mathbf{G}_k \mathbf{u}_k^T \mathbf{P}_{k-1})$$

$$\alpha_k = \gamma + \mathbf{u}_k^T \mathbf{P}_{k-1} \mathbf{u}_k$$

RLS: Note on fixed-point implementation

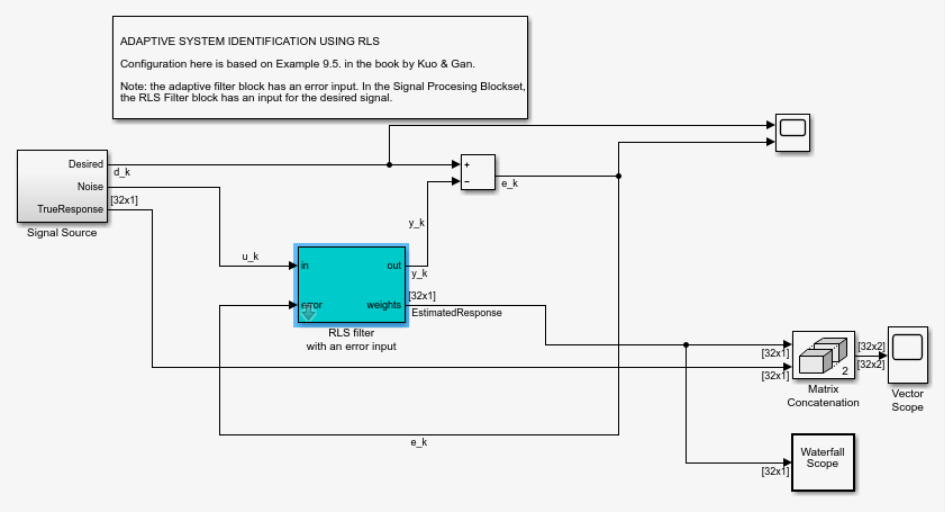
- Basic implementation can involve numerical instability
- Problematic in the algorithm is differencing
  - The result should be positive definite

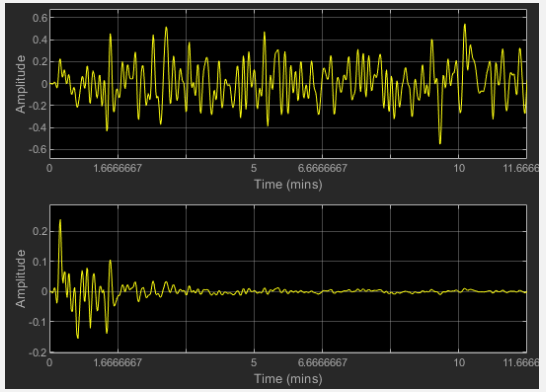
$$\mathbf{P}_k = \frac{1}{\gamma}(\mathbf{P}_{k-1} - \mathbf{G}_k \mathbf{u}_k^T \mathbf{P}_{k-1})$$

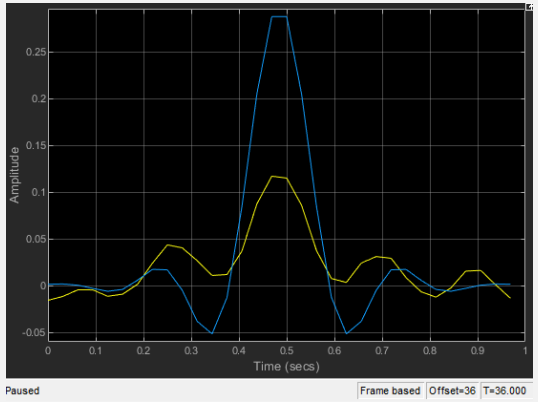
- **Solution**: Matrix factorizations applied to $\mathbf{P}_k$
  - For example, $\mathbf{P}_k = \mathbf{S}_k \mathbf{S}_k^T$, where $\mathbf{S}_k$ is an upper triangular matrix
  - $\mathbf{S}_k$ is the target of updates instead of $\mathbf{P}_k$
  - More in: Ifeachor&Jervis: DSP – Practical approach, 2nd Ed, Ch. 10.5

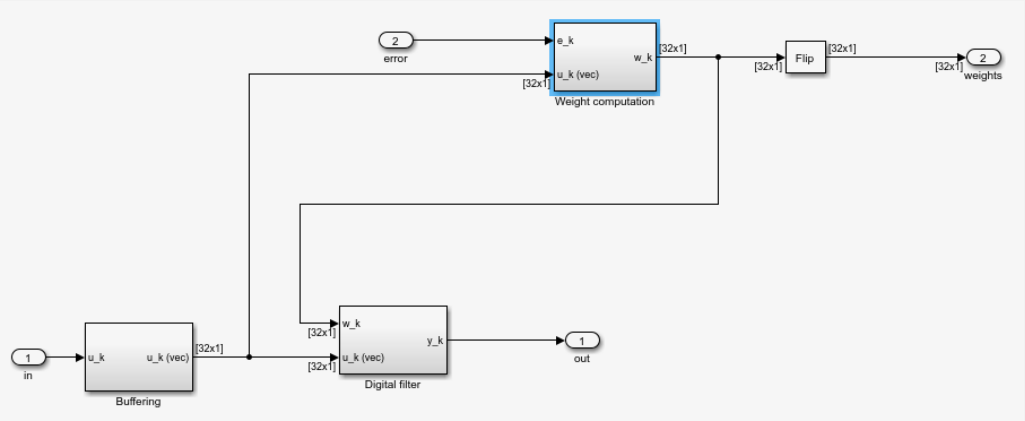
Example. RLS

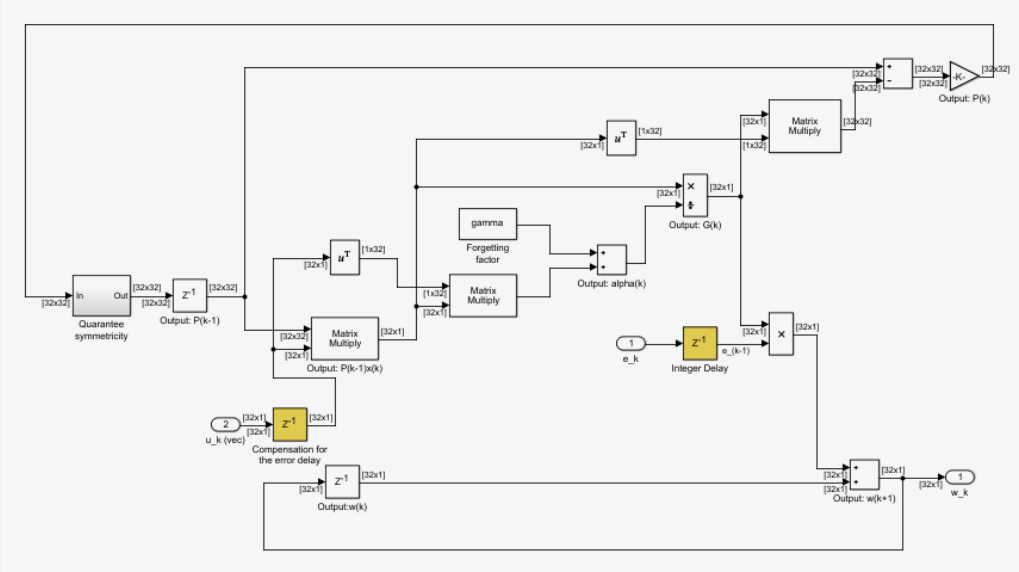
rls_demo.slx in Moodle











Summary

- LMS filter
  - Perhaps the most used filter
  - Roots in Wiener filter & its recursive solution, steepest descent
  - Performs error-based adaptation $\mathbf{w}_{k+1} = \mathbf{w}_k + 2\mu \mathrm{e}_k \mathbf{u}_k$
- Fast block LMS
  - Long filters require computation in the frequency domain
- RLS filter
  - Roots in regression modelling and least squares error estimator
  - Performs error-based adaptation, updating current estimates of the involved matrices, with stability consideration ($\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{G}_k e_k$, $\mathbf{G}_k \leftarrow \mathbf{P}_k \leftarrow \mathbf{S}_k$)