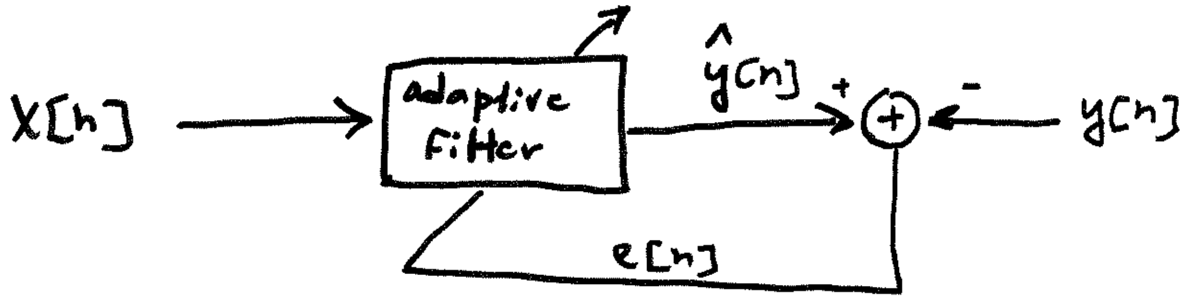
Basic Adaptive Filter Setup



$x[n]$: input

$y[n]$: desired output

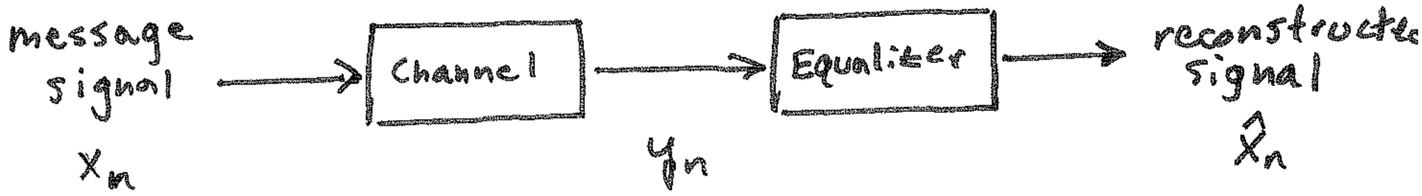
$\hat{y}[n]$: filtered input

$e[n]$: error signal

<u>Operation</u>: use error signal to adjust / adapt the filter in order to drive $\hat{y}[n] \rightarrow y[n]$.

An Adaptive Filtering Example

Channel Equalization



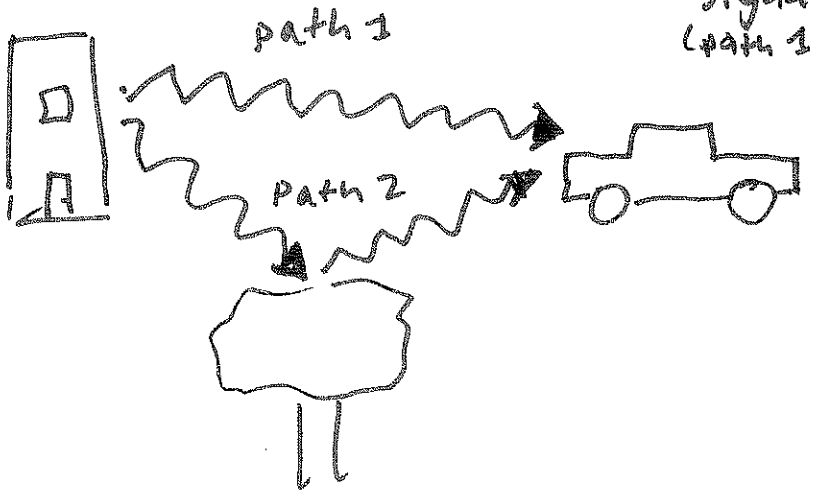
FIR Channel Model:

$$y_n = \sum_{k=1}^{M} h_k x_{n-k+1} + w_n$$

Ex. $(h_1, h_2, \dots, h_M) = (0, 0, 1, 0, 0, -0.5, 0, \dots, 0)$

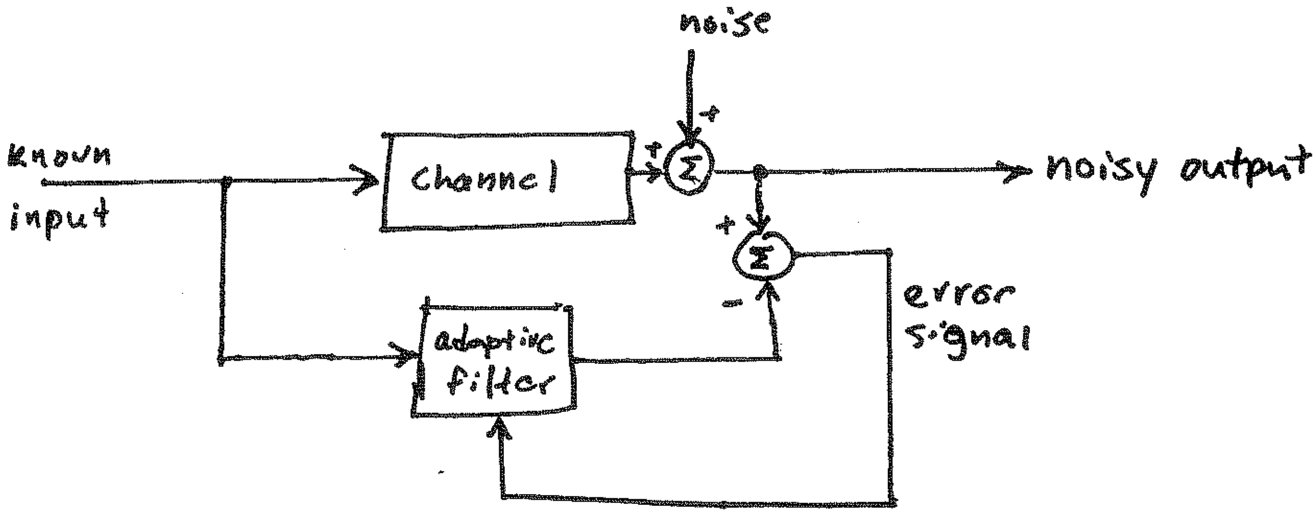
primary direct signal (path 1)

reflection (path 2)



Adaptive Filtering Applications

Channel/System Identification



Noise Cancellation – Suppression of maternal ECG component in fetal ECG

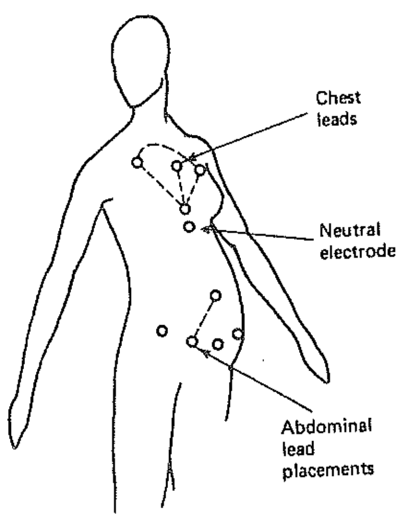
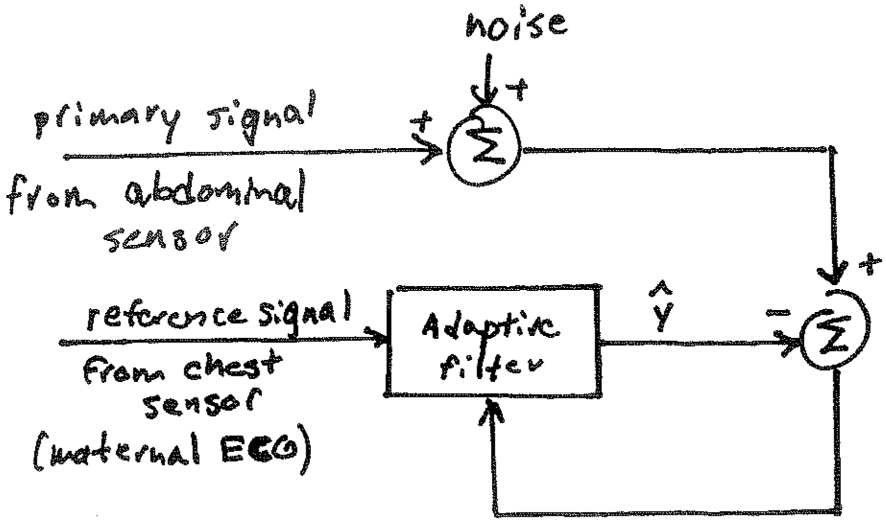


Figure 1.1 Cancelling maternal heartbeat in fetal electrocardiography (ECG): position of leads. From B. Widrow et al, *Adaptive Noise Cancelling: Principles and applications* [W3], © December 1975, IEEE.



$\hat{y}$ is a estimate of the maternal ECG signal present in abdominal signal

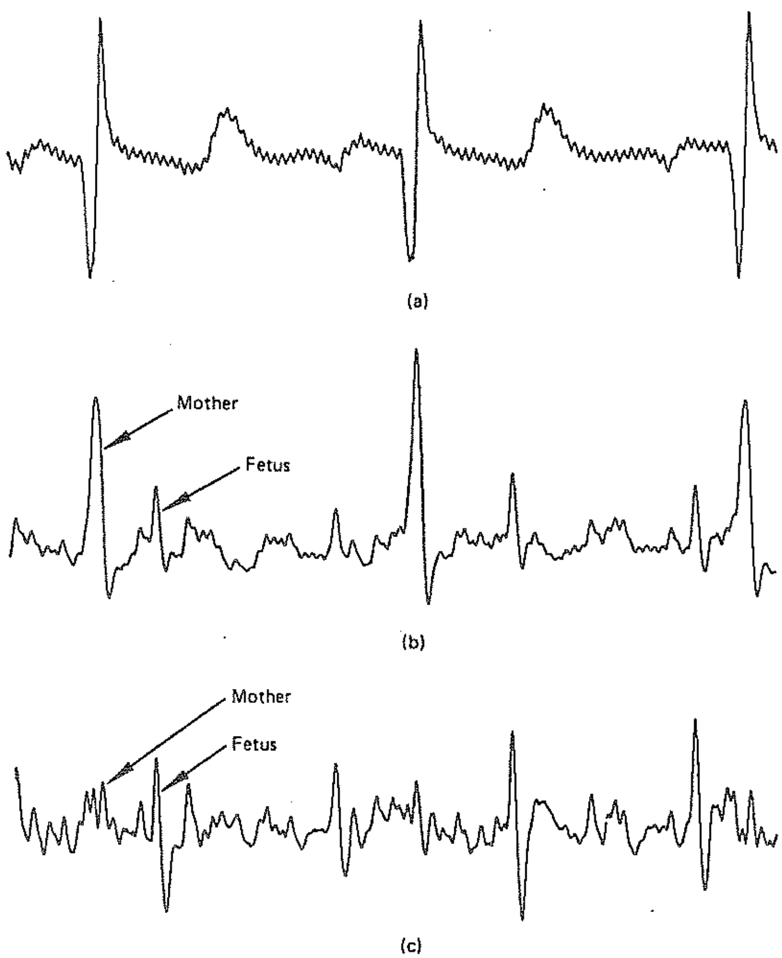
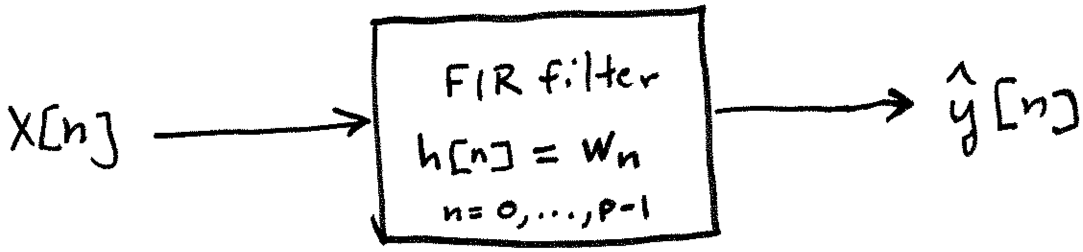


Figure 1.3 Results of fetal ECG experiment (bandwidth, 3–35 Hz; sampling rate, 256 Hz): (a) reference input (chest lead); (b) primary input (abdominal lead); (c) noise-canceller output. From B. Widrow et al, *Adaptive Noise Canceling: Principles and applications* [W3], © December 1975, IEEE.

FIR Adaptive Filters



$$\hat{y}[n] = \sum_{k=0}^{p-1} h[k]\, x[n-k]$$

$$= \sum_{k=0}^{p-1} w_p\, x[n-k]$$

"weights"

Vector notation:

$$\hat{y}[n] = \underline{x}_n^T \, \underline{w}$$

input vector $\quad \underline{x}_n = \left[x[n] \;\; x[n-1] \;\; x[n-2] \;\; \cdots \;\; x[n-p+1] \right]^T$

$$\underline{w} = \left[w_0 \;\; w_1 \;\; w_2 \;\; \cdots \;\; w_p \right]^T$$

"weight vector"

error signal:

$$e[n] = y[n] - \hat{y}[n]$$

$$= y[n] - \underline{x}_n^T \underline{w}$$

assumptions:

$x[n]$ and $y[n]$ are stationary random signals with zero mean

Objective:

Adapt filter so that $\hat{y}[n] \longrightarrow y[n]$

or equivalently

$$e[n] \longrightarrow 0$$

Mean Square Error Analysis

$$E\left[e^2[n]\right] = E\left[\left(y[n] - \sum_{k=0}^{p-1} w_k\, x[n-k]\right)^2\right]$$

$$= E\left[y^2[n]\right] - 2\sum_{k=0}^{p-1} w_k\, E\left[y[n]\,x[n-k]\right]$$

$$+ \sum_{k=0}^{p-1} \sum_{l=0}^{p-1} w_k w_l\, E\left[x[n-k]\,x[n-l]\right]$$

$$= \sigma_y^2 - 2\sum_{k=0}^{p-1} w_k\, R_{xy}[k]$$

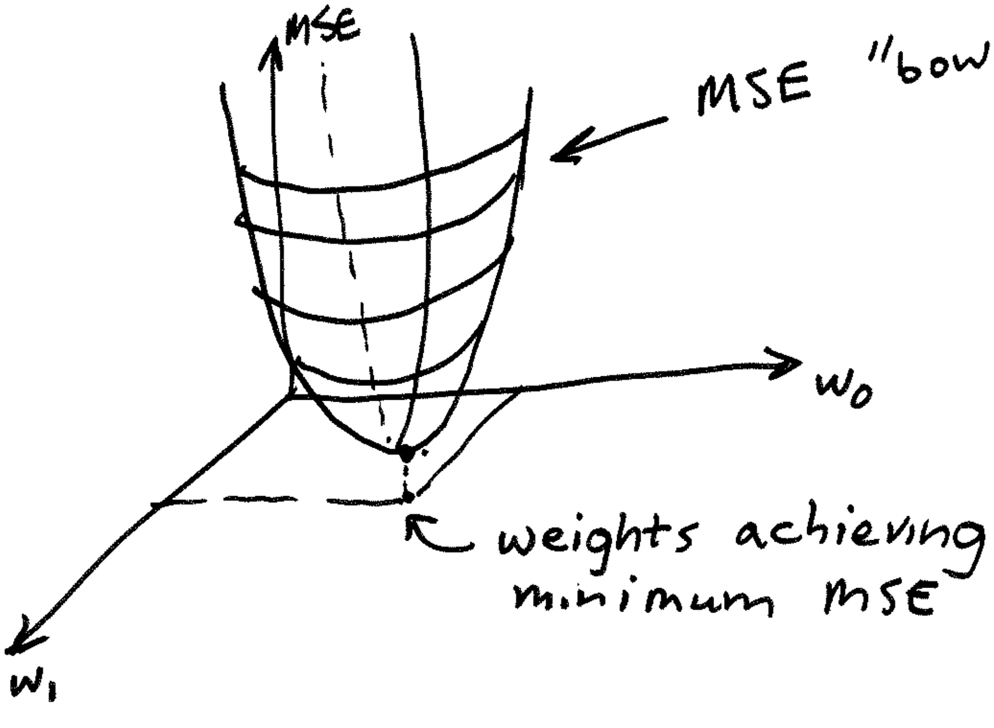
$$+ \sum_{k=0}^{p-1} \sum_{l=0}^{p-1} w_k w_l\, R_{xx}[k-l]$$

$\Rightarrow$ MSE depends on weights and cross-corr R_{xy} and auto-corr R_{xx}.

If we know R_{xy} and R_{xx}, then we can determine the weights that minimize the MSE.

Note: This is identical to FIR Wiener filter set-up!

The MSE is quadratic in $\underline{w}$



Since the MSE is a quadratic, bowl-shaped function of the weights, the minimum MSE is found by solving

$$\frac{\partial E[e^2[n]]}{\partial w_m} = 0 \qquad m = 0, \ldots, p-1$$

for the optimal weights.

$$\frac{\partial E[e^2[n]]}{\partial w_m} = \frac{\partial}{\partial w_m}\left\{ \sigma_y^2 - 2\sum_{k=0}^{p-1} w_k R_{xy}[k] + \sum_{k=0}^{p-1}\sum_{\ell=0}^{p-1} w_k w_\ell R_{xx}[k-\ell] \right\}$$

$$= -2R_{xy}[m] + \frac{\partial}{\partial w_m} E\left(\sum_{k=0}^{p-1} w_k x[n-k]\right)^2$$

$$= -2R_{xy}[m] + 2E\left[x[n-m] \cdot \sum_{k=0}^{p-1} w_k x[n-k]\right]$$

$$= -2R_{xy}[m] + 2\sum_{k=0}^{p-1} R_{xx}[m-k]\, w_k$$

$$\frac{\partial E\left[e^2[n]\right]}{\partial w_m} = -2R_{xy}[m] + 2\sum_{k=0}^{p-1} R_{xx}[m-k]\,w_k = 0$$

$$\Longrightarrow \quad \sum_{k=0}^{p-1} R_{xx}[m-k]\,w_k = R_{xy}[m]$$

$$m = 0, \ldots, p-1$$

or in matrix form:

$$\underbrace{\begin{bmatrix} R_{xx}[0] & R_{xx}[1] & \cdots & R_{xx}[p-1] \\ R_{xx}[1] & R_{xx}[0] & \ddots & \\ R_{xx}[2] & R_{xx}[1] & R_{xx}[0] & \\ \vdots & \ddots & \ddots & \ddots \\ R_{xx}[p-1] & & & R_{xx}[0] \end{bmatrix}}_{\substack{\underline{R_{xx}} \\ \text{covariance matrix} \\ \text{of } x[n]}} \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{p-1} \end{bmatrix}}_{\substack{\underline{w} \\ \text{weight} \\ \text{vector}}} = \underbrace{\begin{bmatrix} R_{xy}[0] \\ R_{xy}[1] \\ \vdots \\ R_{xy}[p-1] \end{bmatrix}}_{\substack{\underline{R_{xy}} \\ \text{cross-corr} \\ \text{vector}}}$$

So, the MSE is minimized by the weights satisfying

$$\underline{R_{xx}}\,\underline{w} = \underline{R_{xy}}$$

(Wiener-Hopf equation)

$$\boxed{\underline{w}^* = \underline{R_{xx}}^{-1}\,\underline{R_{xy}}}$$

matrix inv

Issues:

1. What if p is very large?
   (matrix inv requires $O(p^3)$ flops)

2. What if autocorrelation and cross-corr functions are unknown?

3. What if $x[n]$ and $y[n]$ are non-stationary?

Steepest Descent Optimization

If p is large, then

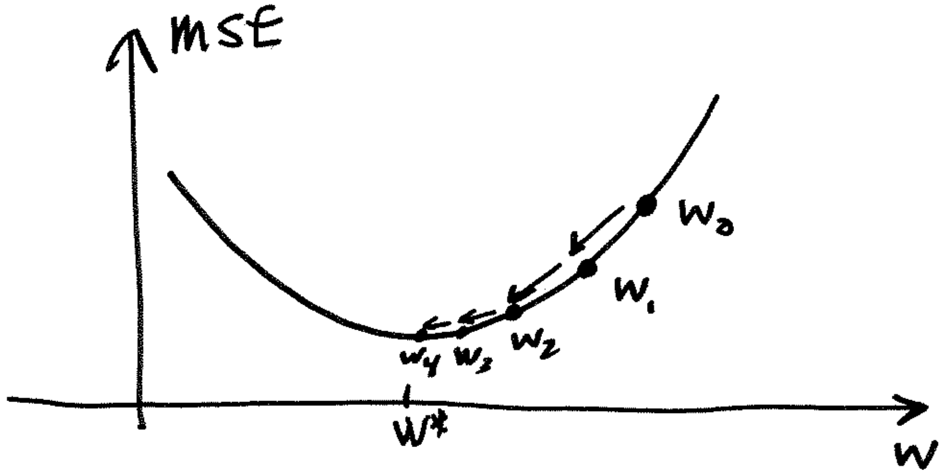
$$\underline{w}^* = \underline{R}_{xx}^{-1} \underline{R}_{xy}$$

cannot be easily computed.

Instead of matrix inversion, we want something less compute-intensive.

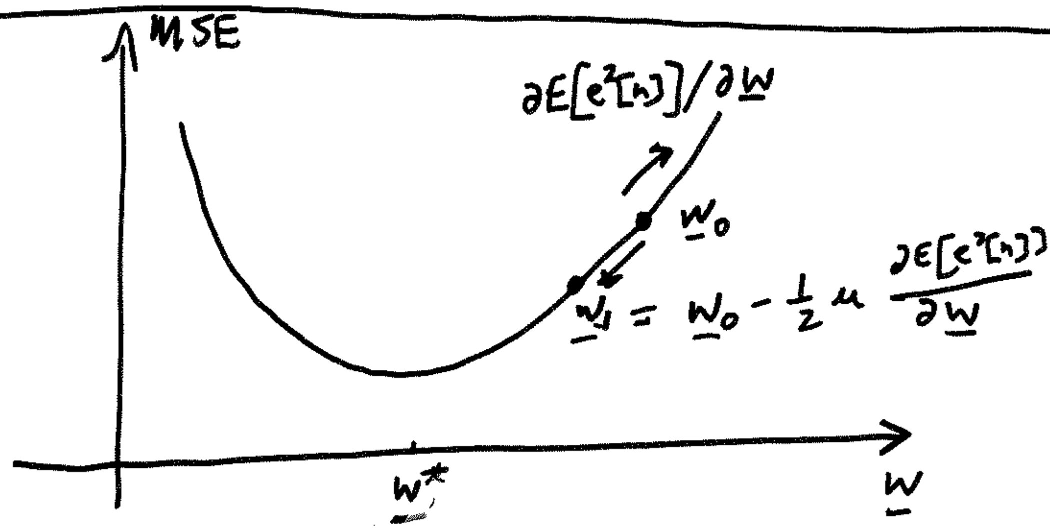
Suppose we think about an iterative approach in which we begin with an initial weight vector $\underline{w}_0$ and iteratively adjust the values to decrease the MSE.

In 1-D



We want to move <u>downhill</u> on the MSE bowl, so a natural and simple adjustment takes the form:

$$\boxed{\underline{w}_1 = \underline{w}_0 - \frac{1}{2}\mu \left.\frac{\partial E[e^2[n]]}{\partial \underline{w}}\right|_{\underline{w}=\underline{w}_0}}$$



Iteration:

$$\underline{w}_k = \underline{w}_{k-1} - \frac{1}{2} \mu \left. \frac{\partial E[e^2[n]]}{\partial \underline{w}} \right|_{\underline{w} = \underline{w}_{k-1}}$$

$\Rightarrow$ $\underline{w}_0, \underline{w}_1, \underline{w}_2, \ldots$

Hopefully, each subsequent $\underline{w}_k$ is a bit closer to $\underline{w}^*$

Does this procedure converge?

Can we adapt it to an "on-line" real-time, dynamic situation in which R_{xx} and R_{xy} are not known and/or $x[n]$ and $y[n]$ are not stationary?

Least Mean Square (LMS) Algorithm

Steepest Descent (Gradient Descent):

$$\underline{W}_n = \underline{W}_{n-1} - \frac{1}{2}\mu \left. \frac{\partial E\left[e^2[n]\right]}{\partial \underline{W}} \right|_{\underline{W} = \underline{W}_{n-1}}$$

where

$$e[n] = \underbrace{y[n]}_{\text{desired output}} - \underbrace{\hat{y}[n]}_{\text{output of adaptive filter}}$$

<u>Ex.</u> FIR filter

$$\hat{y}[n] = \sum_{k=0}^{p-1} W_n[k]\, x[n-k]$$

$W_n[k]$: impulse response of adaptive filter at time n

$x[n-k]$: input

Update for each filter weight:

$$W_n[m] = W_{n-1}[m] - \frac{1}{2}\mu \frac{\partial E\left[\left(y[n] - \sum_{k=0}^{p-1} W_{n-1}[k]\, x[n-k]\right)^2\right]}{\partial W_{n-1}[m]}$$

$$= W_{n-1}[m] - \frac{1}{2}\mu \cdot 2\, E\left[\left(y[n] - \sum_{k=0}^{p-1} W_{n-1}[k]\, x[n-k]\right)\left(-x[n-m]\right)\right]$$

$$= W_{n-1}[m] + \mu \cdot E\left[\left(y[n] - \sum_{k=0}^{p-1} W_{n-1}[k]\, x[n-k]\right) x[n-m]\right]$$

$$= W_{n-1}[m] + \mu \cdot \left(R_{xy}[m] - \sum_{k=0}^{p-1} W_{n-1}[k]\, R_{xx}[m-k]\right)$$

So if we know the autocorrelation function $R_{xx}[m]$ and cross-correlation function $R_{xy}[m]$, then we can easily compute these simple, steepest descent updates.

What if we don't know $R_{xx}[m]$ and $R_{xy}[m]$?

What if $x[n]$ and $y[n]$ are nonstationary?

LMS:

$$\underline{w}_n = \underline{w}_{n-1} - \frac{1}{2}\mu \underbrace{\frac{\partial e^2[n]}{\partial \underline{w}}}_{\substack{\text{"instantaneous"} \\ \text{gradient of error} \\ \text{at time } n}} \Bigg|_{\underline{w} = \underline{w}_{n-1}}$$

No need for correlation functions!

LMS Update per Weight:

$$W_n[m] = W_{n-1}[m] - \frac{1}{2}\mu \frac{\partial \left(y[n] - \sum_{k=0}^{p-1} W_{n-1}[k]\, x[n-k]\right)^2}{\partial W_{n-1}[m]}$$

$$= W_{n-1}[m] - \frac{1}{2}\mu \cdot 2\left(y[n] - \sum_{k=0}^{p-1} W_{n-1}[k]\, x[n-k]\right)\left(-x[n-m]\right)$$

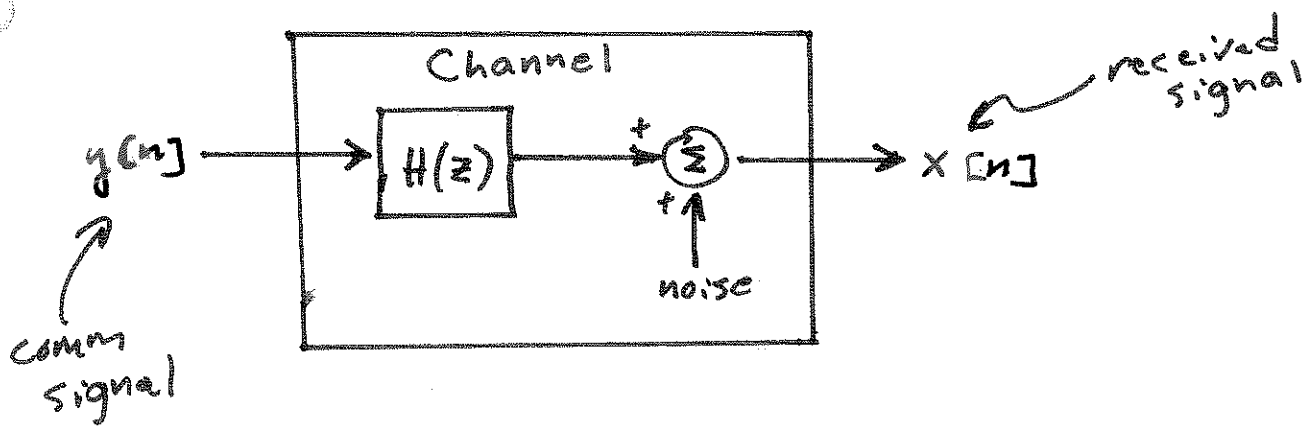
$$= W_{n-1}[m] + \mu \underbrace{\left(y[n] - \hat{y}[n]\right)}_{\text{error}} \cdot \underbrace{x[n-m]}_{\text{input weighted by } W[m]}$$

new weight = old weight − step × error × input

★ simple

★ easy to compute

Ex. Channel Equalization

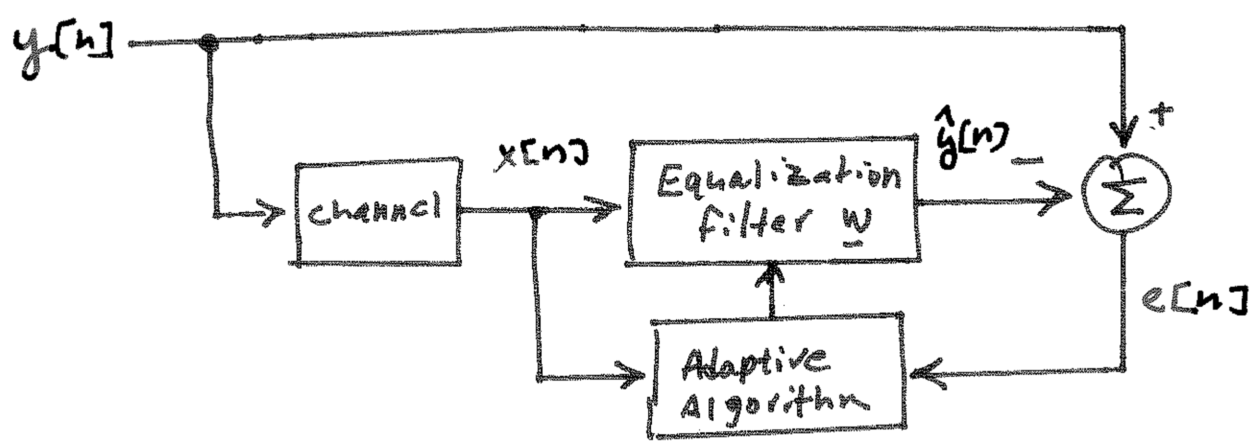


$$H(z) = \frac{1}{(1 + \alpha z^{-1})}$$ single-pole channel

$$\alpha = -0.681$$

By sending a known training signal y_k first, we can equalize the channel so that subsequent information will be

Equalizer in Training Mode



Since the single-pole channel $H(z)$ can be equalized with a single "zero" in the correct position, we will use a 2-tap (2 weights) FIR equalizing filter.

$$\underline{w} = [\, w[0] \; w[1] \,]^T$$

<u>LMS Algorithm:</u>

$$\underline{x}_n = \begin{bmatrix} x[n] \\ x[n-1] \\ \vdots \\ x[n-p+1] \end{bmatrix}$$

$$\underline{w}_n = \underline{w}_{n-1} - \mu \, \underline{x}_n \, e[n]$$

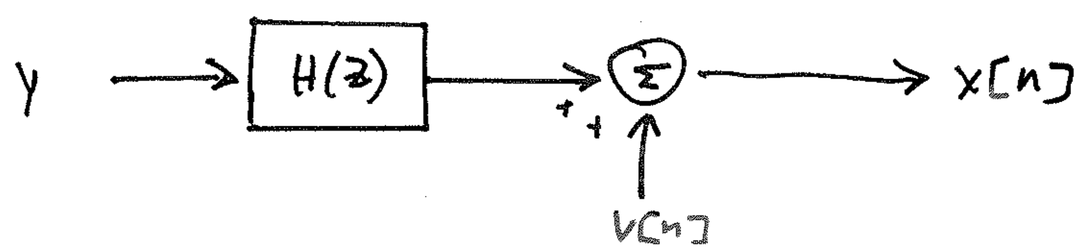
$$e[n] = y[n] - \hat{y}[n]$$

To simulate this experimentally, lets generate the training signal by synthesizing a realization from the first-order, singe pole Model:

$$\boxed{y[n] = 0.88\, y[n-1] + u[n]}$$

correlated random signal

$u[n]$
iid
white Gaussian noise
$\sigma^2 = 1/2$

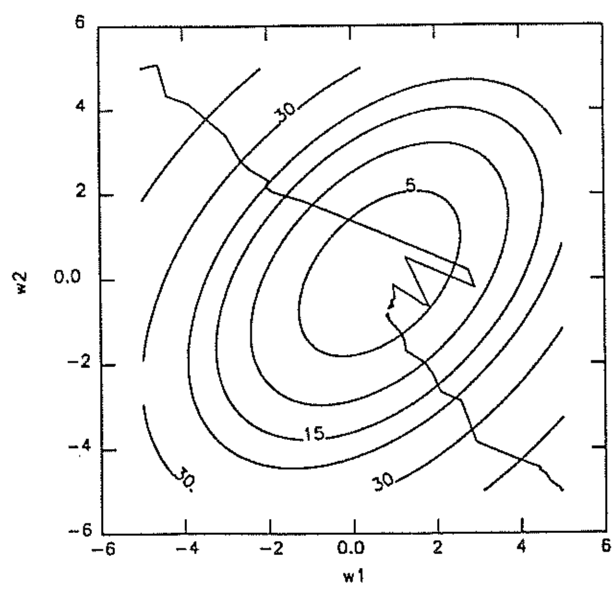


$$\boxed{x[n] = \alpha\, x[n-1] + v[n]}$$

The filter $\underline{w}$ that minimizes the MSE in this case turns out to be

$$\underline{w}_{opt} = \begin{bmatrix} 0.921 \\ -0.655 \end{bmatrix}$$

The trajectories of the LMS iteration, starting from two different initializations, are shown below.



Step Size and Convergence

Steepest Descent:

$$\underline{W}_n = \underline{W}_{n-1} - \frac{1}{2}\mu \left. \frac{\partial E[e^2[n]]}{\partial \underline{W}} \right|_{\underline{W} = \underline{W}_{n-1}}$$

$$= \underline{W}_{n-1} - \left(-\underline{R}_{xy} + \underline{R}_{xx}\underline{W}_{n-1}\right)$$

$$= \underline{W}_{n-1} - \mu \underline{R}_{xx}\left(-\underline{R}_{xx}^{-1}\underline{R}_{xy} + \underline{W}_{n-1}\right)$$

$$= \underline{W}_{n-1} - \mu \underline{R}_{xx}\left(\underline{W}_{n-1} - \underline{W}_{opt}\right)$$

$\Longrightarrow$

$$\underbrace{\underline{W}_n - \underline{W}_{opt}}_{\underline{V}_n} = \underbrace{\underline{W}_{n-1} - \underline{W}_{opt}}_{\underline{V}_{n-1}} - \mu \underline{R}_{xx}\left(\underline{W}_{n-1} - \underline{W}_{opt}\right)$$

$$\Longrightarrow \quad \underline{V}_n = \underline{V}_{n-1} - \mu \underline{R}_{xx}\underline{V}_{n-1}$$

$$\text{convergence} \Longleftrightarrow \underline{V}_n \to 0$$

$$\underline{V}_n = (\underline{I} - \mu \underline{R}_{xx}) \underline{V}_{n-1}$$

- or -

$$\underline{V}_n = (\underline{I} - \mu \underline{R}_{xx})^n \underline{V}_0$$

arbitrary intial

$\underline{V}_0 = \underline{W}_0 - \underline{W}_{opt}$

Clearly $\underline{V}_n \to 0$ if and only if

$$(\underline{I} - \mu \underline{R}_{xx})^n \to 0$$

as $n \to \infty$

This happens if all eigenvalues of $(\underline{I} - \mu \underline{R}_{xx})$ are less than 1 in magnitude.

The eigenvalues are less than 1 in magnitude if

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

largest eigenvalue of $\underline{R}_{xx}$

LMS Step Size:

$$\underline{w}_n = \underline{w}_{n-1} - \frac{1}{2}\mu \underbrace{\frac{\partial e^2[n]}{\partial \underline{w}}}_{\text{"instantaneous" gradient}} \Bigg|_{\underline{w}=\underline{w}_{n-1}}$$

$$\frac{\partial e^2[n]}{\partial \underline{w}} \approx \frac{\partial E[e^2[n]]}{\partial \underline{w}} + \text{noise}$$

LMS steps are "noisy"

Noisy steps $\Rightarrow$ more conservative choice / range for step size μ

Theoretical arguments suggest that

$$\mu < \frac{2/3}{\sum_{i=0}^{p-1} \lambda_i},$$

where λ_i is the i-th eigenvalue of $\underline{R}_{xx}$, is a reasonable upper bound on μ for the LMS algorithm.

Note:

$$\frac{2/3}{\sum_{i=0}^{p-1} \lambda_i} < \frac{2}{\lambda_{max}}$$

How do we know $\underline{R}_{xx}$?

We can estimate $\underline{R}_{xx}$ from the input signal.

Let $\quad \underline{x}_n = \begin{bmatrix} x[n] \\ x[n-1] \\ \vdots \\ x(n-p+1) \end{bmatrix} \qquad n = 0, 1, \ldots$

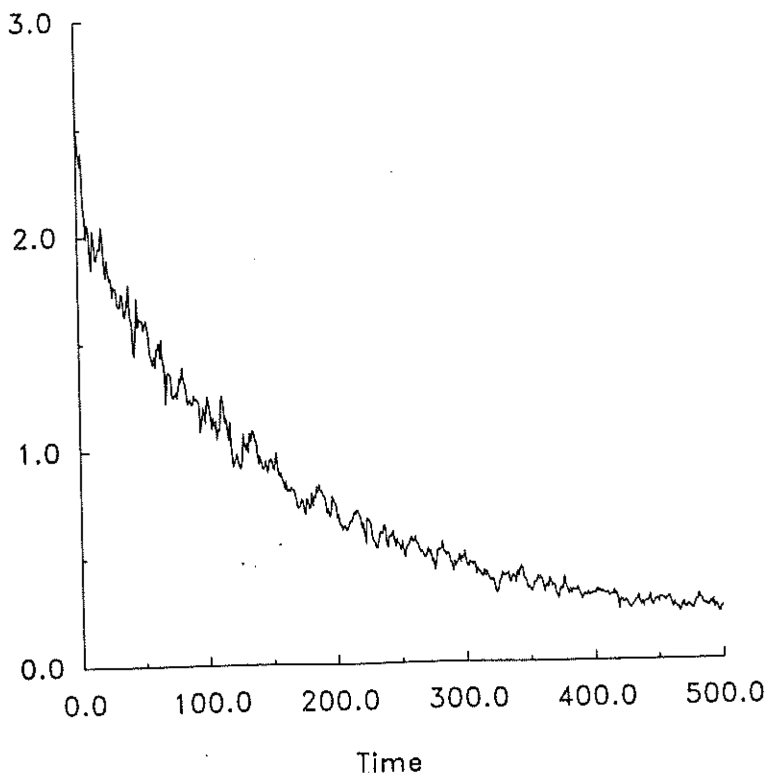
and form

$$\hat{\underline{R}}_{xx} = \frac{1}{N} \sum_{n=0}^{N-1} \underline{x}_n \underline{x}_n^T$$

<u>Note</u>: This is just replacing the ideal expectation with an average

$$\underline{R}_{xx} = E\left[\underline{x}_n \underline{x}_n^T\right] \approx \frac{1}{N} \sum_{n=0}^{N-1} \underline{x}_n \underline{x}_n^T$$

The performance of the LMS algorithm can be assessed experimentally by estimating the "learning curve" (MSE vs. iteration). Since LMS relies on the actual squared error e and not an expected value, the learning curves from different runs will depend heavily upon the noise terms in the observations. To estimate the learning curve we can perform several (e.g. 500) runs of the LMS algorithm, with independent noise realizations in each case.



Step Size and Tracking

In many cases, the filter characteristics must change over time (e.g., tracking a time varying comm channel).

To quickly adapt we should choose the step size as large as possible

$$\mu = \frac{2/3}{\sum_i \lambda_i}$$

On the other hand, in steady-state we don't want the filter to change at all. Noise, however will cause the filter weights to fluctuate

$$\underline{w}_n = \underline{w}_{n-1} - \frac{1}{2}\mu \frac{\partial e^2[n]}{\partial \underline{w}}$$ ← "noisy" error

and the amount of fluctuation is proportional to the step size μ.

small μ $\Rightarrow$ small fluctuation in steady-state.

Summary of LMS

Update:

$$W_n[m] = W_{n-1}[m] + \mu \cdot e[n] \cdot x[n-m]$$

where $e[n] = y[n] - \hat{y}[n]$

$$\hat{y}[n] = \sum_{k=0}^{p-1} W_{n-1}[k]\, x[n-k]$$

Step Size:

$$0 < \mu < \frac{2/3}{\sum_{i=0}^{p-1} \lambda_i}$$

$\{\lambda_i\} :=$ eigenvalues of $\underline{R}_{xx}$