Statistical Signal Processing & Inference

Adaptive Estimation and Inference

Danilo Mandic

room 813, ext: 46271

Department of Electrical and Electronic Engineering

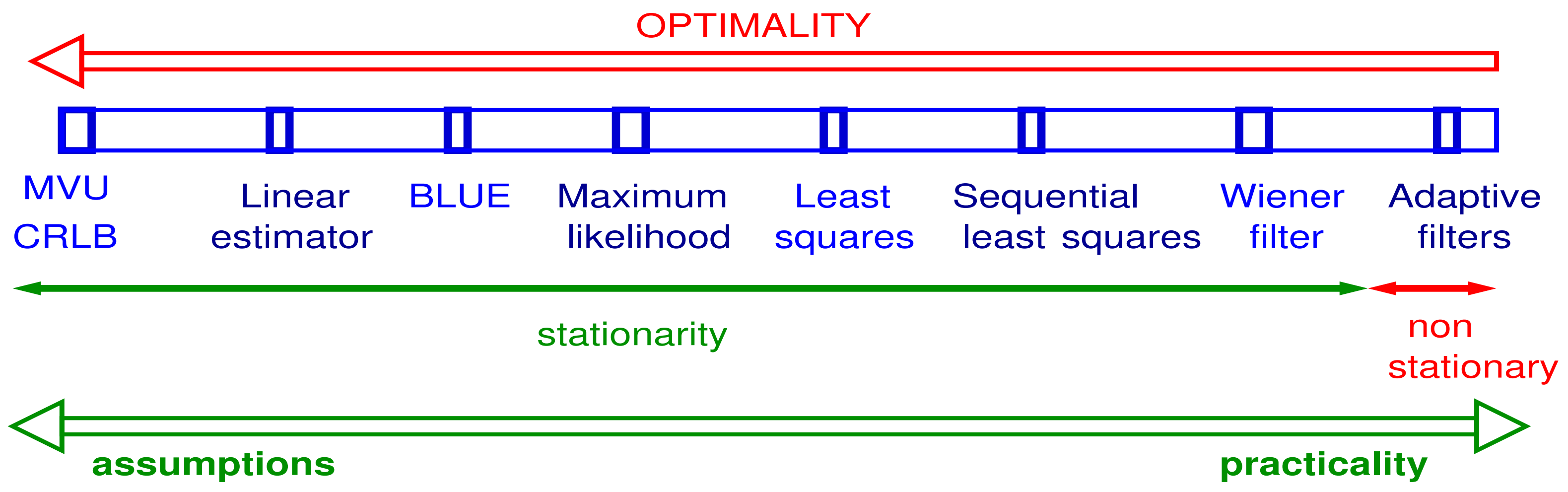
Imperial College London, UK

d.mandic@imperial.ac.uk, URL: www.commsp.ee.ic.ac.uk/∼mandic

Aims

- To introduce real-time adaptive estimation for streaming data
- Adaptive filters $\looparrowright$ "ARMA models with adaptive coefficients"
- Wiener filter and the method of steepest descent
- Stochastic gradient and the Least Mean Square (LMS) algorithm
- Role of learning rate (stepsize), bias and variance in estimation
- Adaptive filtering configurations (prediction, SYS ID, denoising, ...)
- Simple nonlinear structures (model of an artificial neuron)
- Stability and convergence of adaptive estimators, link with CRLB
- Applications (also a link with your Coursework)

A big picture of estimators so far



- Minimum variance unbiased estimator (MVU), Cramer Rao Lower Bound (CRLB) $\looparrowright$ known pdf, linearity assumption, stationarity
- Linear model $\looparrowright$ known pdf, stationarity, and linearity
- Best linear unbiased estimator (BLUE) $\looparrowright$ linear in the unknown parameter, stationarity, pdf not needed
- Maximum likelihood estimation (MLE) $\looparrowright$ stationarity, known pdf
- Least squares estimation (LS) $\looparrowright$ stationarity, deterministic data model
- Wiener filter $\looparrowright$ stationarity, no other assumptions
- **Adaptive filters $\looparrowright$ no assumptions**

Number guessing game
principle of adaptive estimation

Let us play a guessing game: One person will pick an integer between -100 and 100 and remember it, and the rest of us will try to discover that number in the following ways:

- Random guess with no feedback;
- Random guess followed by feedback $\rightsquigarrow$ the only information given is whether the guess was high or low;
- But we can make it a bit more complicated $\rightsquigarrow$ the guessed number may change along the iterations (nonstationarity).

Let us formalise this: If the current guess is denoted by $g_i(n)$, we can build a **recursive update** in the form

$$\begin{aligned} g_i(n+1) \quad &= \quad g_i(n) + \mathrm{sign}\big(e(n)\big)\,\mathrm{rand}\big[g_i(n), g_i(n-1)\big] \\ \text{new guess} \quad &= \quad \text{old guess} + \text{correction} \end{aligned}$$

Welcome to the wonderful world of adaptive filters!

Adaptive filters
basis for machine intelligence

The last equation was actually an adaptive filter in the form:

$$\left(\begin{array}{c}\text{New}\\\text{Estimate}\end{array}\right)=\left(\begin{array}{c}\text{Old}\\\text{Estimate}\end{array}\right)+\left(\begin{array}{c}\text{Correction}\\\text{Term}\end{array}\right)$$

Usually

$$\left(\begin{array}{c}\text{Correction}\\\text{Term}\end{array}\right)=\left(\begin{array}{c}\text{Learning}\\\text{Rate}\end{array}\right)\times\left(\begin{array}{c}\text{Function of}\\\text{Input Data}\end{array}\right)\times\left(\begin{array}{c}\text{Function of}\\\text{Output Error}\end{array}\right)$$

This is the very basis of learning in any adaptive machine!

The most famous example is the **Least Mean Square (LMS)** algorithm, for which the parameter (weights) update equation is given by (more later)

$$\mathbf{w}(n+1)=\mathbf{w}(n)+\mu e(n)\mathbf{x}(n)$$

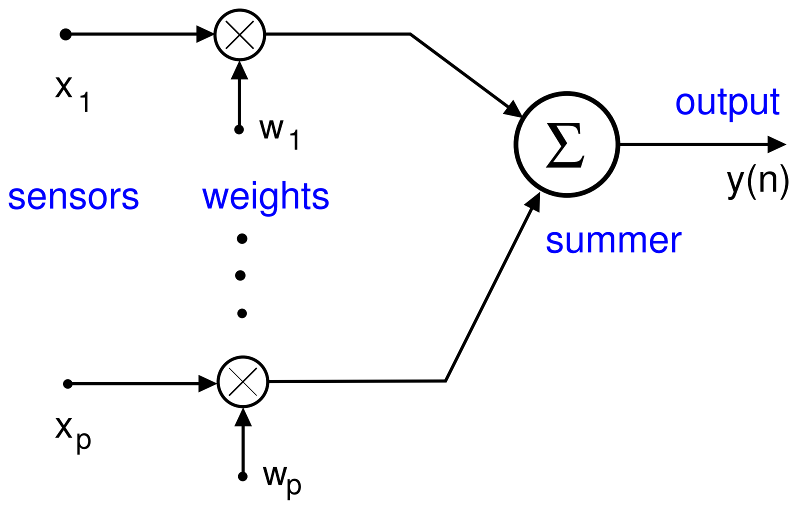
where $\mathbf{w}(n)\in\mathbb{R}^{p\times 1}$ are (time-varying) filter coefficients, commonly called **filter weights**, $\mathbf{x}(n)\in\mathbb{R}^{p\times 1}$ are input data in filter memory, $e(n)$ is the output error at time instant n, and $\mu>0$ is the learning rate (step size).

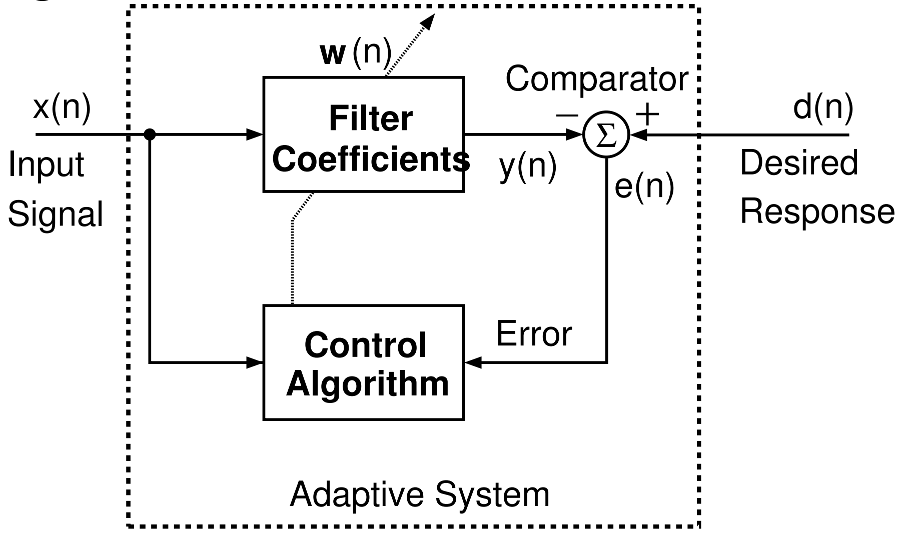
Problem formulation

from a fixed h in digital filters to a time-varying $\mathrm{w}(n)$ in adaptive filters

Consider a set of p sensors at different points in space (filter order p)

Let $x_1, x_2, \ldots, x_p$ be the individual signals from the sensors





- The sensor signals are weighted by the corresponding set of **time–varying** filter parameters $\mathbf{w}(n) = [w_1(n), \ldots, w_p(n)]^T$ (weights)
- The weighted signals are then summed to produce the output

$$y(n) = \sum_{i=1}^{p} w_i(n)x_i(n) = \mathbf{x}^T(n)\mathbf{w}(n) = \mathbf{w}^T(n)\mathbf{x}(n), \qquad n = 0, 1, 2, \ldots$$

where $\quad \mathbf{x}^T(n) = [x_1(n), \ldots, x_p(n)], \quad \mathbf{w}^T(n) = [w_1(n), \ldots, w_p(n)]$

Wiener–Hopf solution: The setting

Objective: To determine the **optimum** set of **fixed** weights $\mathbf{w}_o = [w_{o1}, \ldots, w_{op}]^T$ so as to minimize the difference between the system output and some desired response d in the mean square sense.

- The input-output relation of the filter is given by

$$y(n) = \sum_{k=1}^{p} w_k(n)x_k(n) = \mathbf{w}^T(n)\mathbf{x}(n)$$

- Let $\{d(n)\}$ denote the **desired response** or (*target output or teaching signal*) for the filter. Then, the **error signal** is

$$e(n) = d(n) - y(n)$$

- A natural **objective function** or **cost function** we wish to optimise is the **mean square error**, defined as (note the expectation $E\{\cdot\}$)

$$J = \frac{1}{2}E\{e^2(n)\} = \frac{1}{2}E\Big\{\Big(d(n) - \sum_{k=1}^{p} w_k(n)x_k(n)\Big)^2\Big\} \qquad (= J(\mathbf{w}))$$

In other words, we wish to **minimise** the expected value of error power.

Wiener filter and the optimal filtering problem

for convenience we use $\frac{1}{2}E\{e^2(n)\}$ instead of $E\{e^2(n)\}$ $\rightsquigarrow$ both have min. at $\mathbf{w}_o$

The optimal filtering problem for a given signal $\{x(n)\}$:

Determine the **optimal set of weights** $\mathbf{w}_o = [w_{o1}, \ldots, w_{op}]^T$ (fixed) for which the mean square error $J = \frac{1}{2}E\{e^2(n)\}$ is minimum.

The solution to this problem is known as the Wiener filter.

The cost function is quadratic in the error (and thus in the weights), it is convex and has exactly one minimum $J_{min} = J(\mathbf{w}_o)$, corresponding to $\mathbf{w}_o$

$$J = \frac{1}{2}E\{e^2\} = \frac{1}{2}E\{d^2\} - E\left\{\sum_{k=1}^{p} w_k x_k d\right\} + \frac{1}{2}E\left\{\sum_{j=1}^{p}\sum_{k=1}^{p} w_j w_k x_j x_k\right\}$$

where the "double summation" calculates the square of a sum, that is $\left(\sum_k\right)^2 = \sum_k \sum_j$, and the time index "n" is omitted for brevity.

$$\text{Then} \qquad J = \frac{1}{2}E\{d^2\} - \sum_{k=1}^{p} w_k E\{x_k d\} + \frac{1}{2}\sum_{j=1}^{p}\sum_{k=1}^{p} w_j w_k E\{x_j x_k\}$$

Wiener filter: Error surface

Introduce the notation:

$$\begin{aligned}
\sigma_d^2 &= E\{d^2\} && \rightarrow && \text{power of the teaching (desired) signal} \\
r_{dx}(k) &= E\{dx_k\}, \quad k = 1, 2, \ldots, p && \rightarrow && \text{crosscorrelation between } d \ \& \ x_k \\
r_x(j,k) &= E\{x_j x_k\}, \quad j,k = 1, 2, \ldots, p && \rightarrow && \text{autocorrelation at lag } (j-k)
\end{aligned}$$

Plug back into J to yield

$$J = \frac{1}{2}\sigma_d^2 - \sum_{k=1}^{p} w_k r_{dx}(k) + \frac{1}{2}\sum_{j=1}^{p}\sum_{k=1}^{p} w_j w_k r_x(j,k)$$

Definition: A multidimensional plot of the cost function J versus the weights (free parameters) $w_1, \ldots, w_p$ constitutes the **error performance surface** or simply the **error surface** of the filter.

The error surface is bowl–shaped with a well–defined bottom (global minimum point). It is precisely at this point where the spatial filter from Slide 6 is optimal in the sense that the mean squared error attains its minimum value $J_{min} = J(\mathbf{w}_o)$.

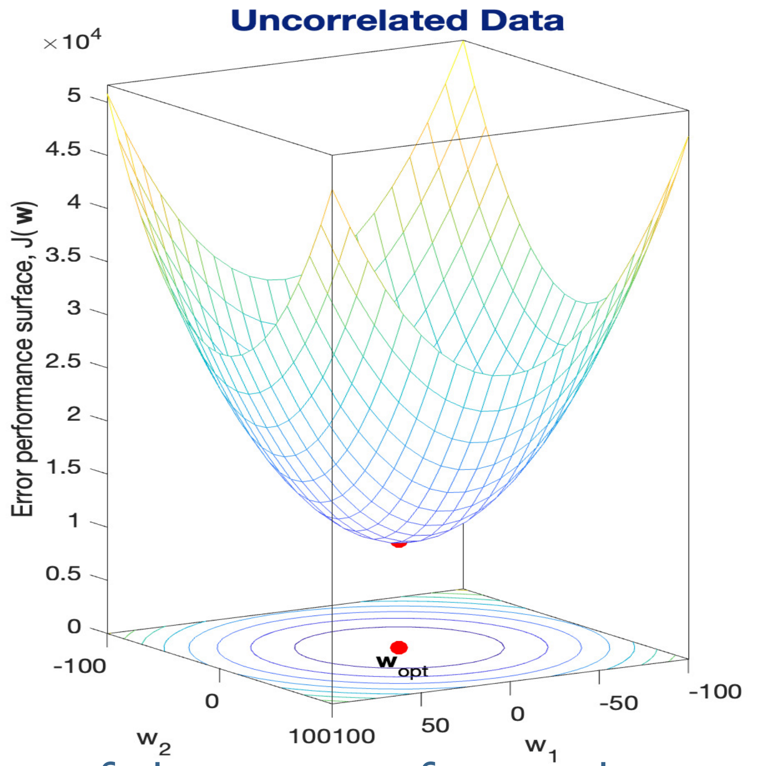
Recall that $J = J(e) = J(\mathbf{w})$, as **the unknown parameter is the weight vector.**

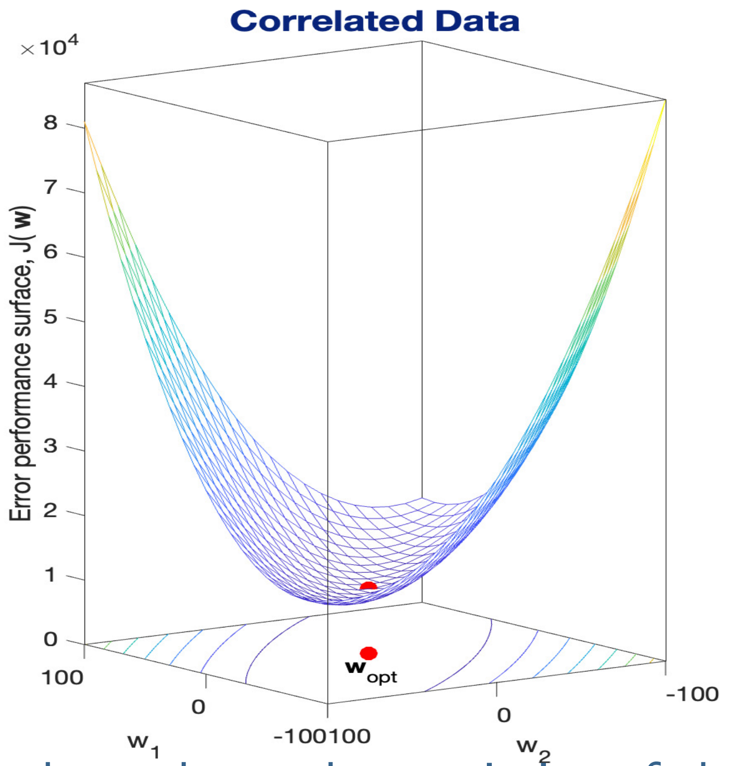
Error Performance Surface and Wiener solution

Error Performance Surface (EPS) and Contour Plots Contour_Plot.m

Error Performance Surface (EPS): A plot of the cost function $J = J(e) = J(\mathbf{w})$ against the whole range of possible weights, $\mathbf{w}$

Contour plot: A 2D plot of the projections (horizontal slices) of the 3D EPS on the weights plane; in our case, projections on the $[w_1, w_2]$ plane





Shape of the error surface and contours depends on the statistics of the input (concentric circles for white data, ellipses for correlated data)

Finally, the Wiener solution
(fixed set of optimum weight $\looparrowright$ a static solution)

To determine the optimum weights, follow the least squares approach:

$$\nabla_{w_k} J = \frac{\partial J}{\partial w_k} = \frac{\partial}{\partial w_k}\Big[\frac{1}{2}\sigma_d^2 - \sum_{k=1}^{p} w_k r_{dx}(k) + \frac{1}{2}\sum_{j=1}^{p}\sum_{k=1}^{p} w_j w_k r_x(j,k)\Big] \qquad k = 1,\dots,p$$

Differentiate wrt to w_k and set to zero to give

$$\nabla_{w_k} J = -r_{dx}(k) + \sum_{j=1}^{p} w_j r_x(j,k) = 0$$

Let w_{ok} denote the optimum value of weight w_k. Then, the optimum weights are determined by the following set of simultaneous equations

$$\sum_{j=1}^{p} w_{oj} r_x(j,k) = r_{dx}(k), \qquad k = 1,2,\dots,p \quad \Leftrightarrow \quad \mathbf{R}_{xx}\mathbf{w}_o = \mathbf{r}_{d\mathbf{x}}$$

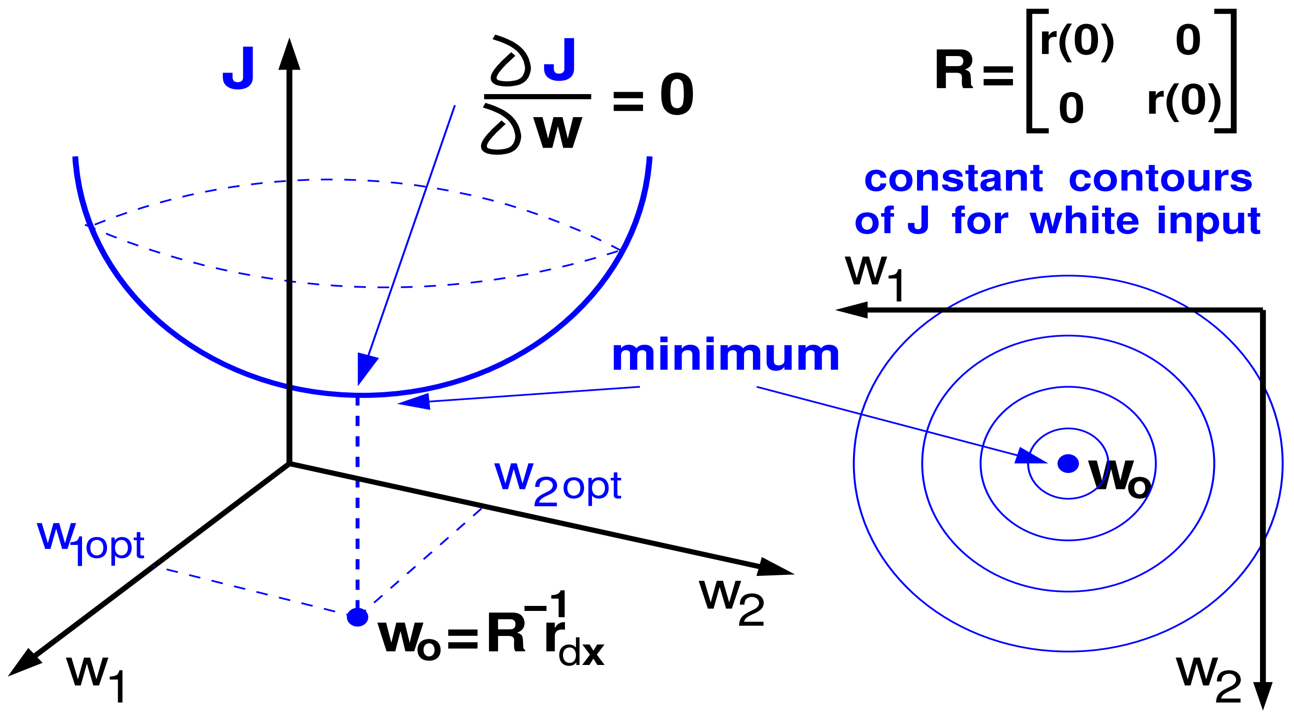
or in a compact form $\qquad \mathbf{w}_o = \mathbf{R}_{\mathbf{xx}}^{-1}\mathbf{r}_{d\mathbf{x}}$

This system of equations is termed the **Wiener-Hopf** equations. The filter whose weights satisfy the Wiener-Hopf equations is called **Wiener filter**. ($\mathbf{R}_{\mathbf{xx}}$ is the input autocorrelation matrix and $\mathbf{r}_{d\mathbf{x}}$ the vector of $\{r_{dx}\}$)

Notice, this is a block filter, operating on the whole set of data (non-sequential)

Wiener solution and error performance surface

- The Wiener solution is now illustrated for the two–dimensional case, by plotting the cost function $J(\mathbf{w})$ against the weights w_1 and w_2, which are elements of the two–dimensional weight vector $\mathbf{w}(n) = [w_1, w_2]^T$.
- The distinguishing feature is that a linear system can find a unique global minimum of the cost function, whereas in nonlinear adaptive systems (neural networks) we can have both global and local minima.


J
$\frac{\partial J}{\partial \mathbf{w}} = 0$
minimum
w_{1opt}
w_{2opt}
w_1
w_2
$\mathbf{w_o} = \mathbf{R}^{-1}\mathbf{r}_{dx}$
$\mathbf{R} = \begin{bmatrix} r(0) & 0 \\ 0 & r(0) \end{bmatrix}$
constant contours of J for white input
w_1
w_2
w_o

Vector-matrix formulation of the Wiener filter

The cost (error, objective) function, $J = \frac{1}{2}e^2(n)$ can be expanded as

$$\begin{aligned} J &= \frac{1}{2}\,E\{e\;e^T\} = \frac{1}{2}E\Big\{\big(d-\mathbf{w}^T\mathbf{x}\big)\big(d-\mathbf{w}^T\mathbf{x}\big)^T\Big\} \\ &= \frac{1}{2}\,E\{d^2 - d\mathbf{x}^T\mathbf{w} - d\mathbf{w}^T\mathbf{x} + \mathbf{w}^T\mathbf{x}\mathbf{x}^T\mathbf{w}\} \\ &= \frac{1}{2}\,E\{d^2 - 2d\mathbf{x}^T\mathbf{w} + \mathbf{w}^T\mathbf{x}\mathbf{x}^T\mathbf{w}\} \\ &= \frac{1}{2}\,E\{d^2\} - \frac{1}{2}\,2\mathbf{w}^T E\{\mathbf{x}d\} + \frac{1}{2}\,\mathbf{w}^T E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w} \end{aligned}$$

where the **cross-correlation** vector $\mathbf{r}_{d\mathbf{x}} \equiv E[\mathbf{x}d]^T$ and **autocorr.** matrix $\mathbf{R} \equiv E[\mathbf{x}\mathbf{x}^T]$

Thus, ($\mathbf{w}$ is still a fixed vector for the time being) the cost function

$$J = \frac{1}{2}\sigma_d^2 - \mathbf{w}^T\mathbf{r}_{d\mathbf{x}} + \frac{1}{2}\mathbf{w}^T\mathbf{R}\mathbf{w}$$

is quadratic in $\mathbf{w}$ and for a full rank $\mathbf{R}$, it has **a unique minimum**, $J(\mathbf{w}_o)$.

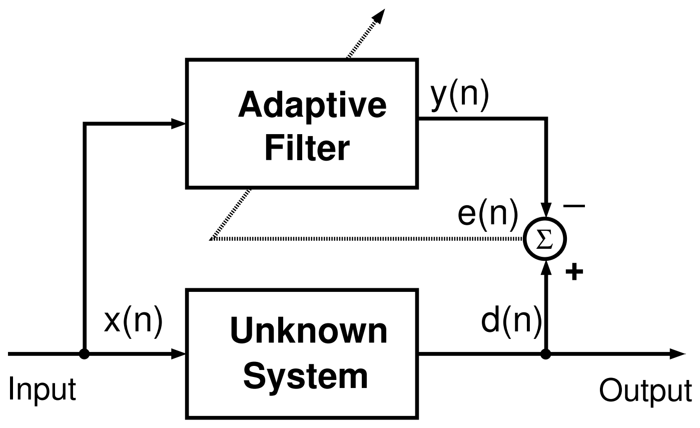
Now: For $J_{min} = J(\mathbf{w}_o) \quad \looparrowright \quad \partial J\,/\,\partial\mathbf{w} = -\mathbf{r}_{d\mathbf{x}} + \mathbf{R}\cdot\mathbf{w} = \mathbf{0} \quad \Rightarrow \quad -\mathbf{r}_{d\mathbf{x}} + \mathbf{R}\cdot\mathbf{w}_o = \mathbf{0}$

Finally $\qquad \mathbf{w}_o = \mathbf{R}^{-1}\mathbf{r}_{d\mathbf{x}}$ $\qquad$ the co-called **Wiener–Hopf equations**

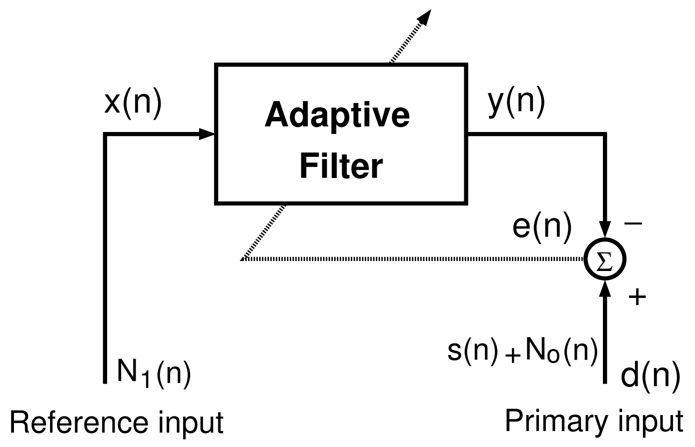
Applications: Adaptive filtering configurations

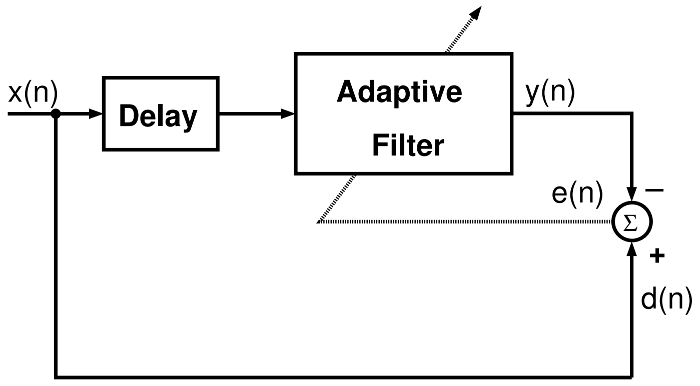
(valid for all adaptive filtering algorithms, more later in the context of LMS)

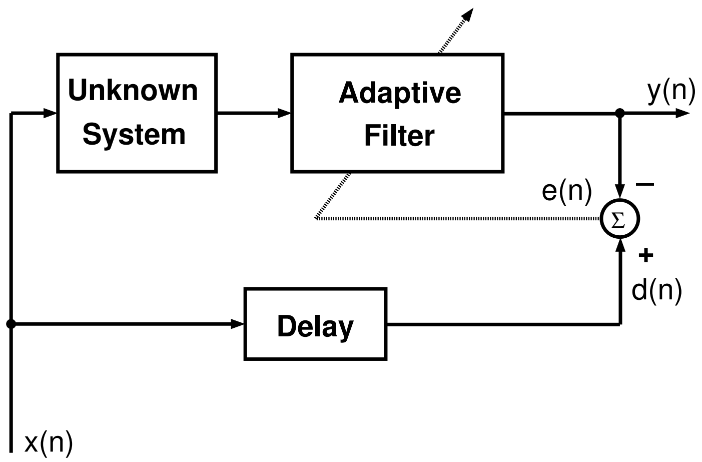
System identification



Noise cancellation





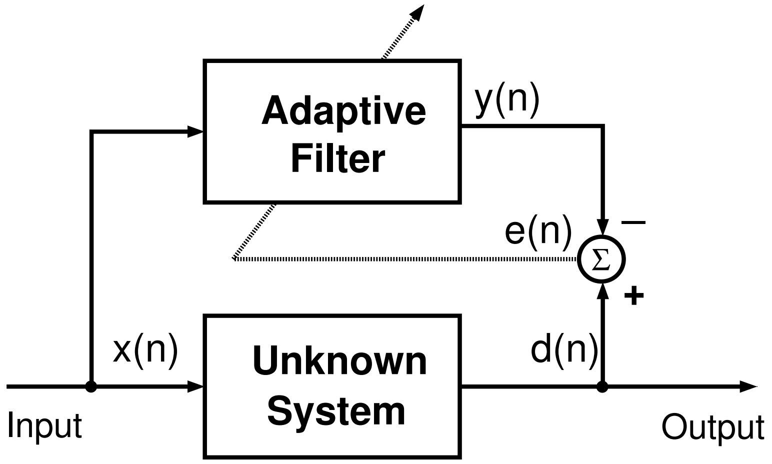


Adaptive prediction

Inverse system modelling

A zoom-in into adaptive system identification

Consider a System Identification (SYS-ID) configuration of adaptive filters



◦ Our goal is to identify the parameters, $\mathbf{h} = [h_1, \ldots, h_p]^T$, of an unknown FIR system (plant) in an adaptive, on–line manner

◦ To this end, we connect the adaptive filter "in parallel" with the unknown system, with the aim to achieve $\mathbf{w}(\infty) \approx \mathbf{h}$ after convergence

◦ The unknown parameters, $\mathbf{h}$, are found by minimising a suitable **convex error function**, for example, the error power, $J(e) = J(\mathbf{w}) = E\{e^2(n)\}$

◦ Then, the **optimum set of adaptive filter weights**, $\mathbf{w}_o = \mathbf{w}(\infty) \approx \mathbf{h}$

Example 1: Performance of the Wiener filter (in SYS ID)

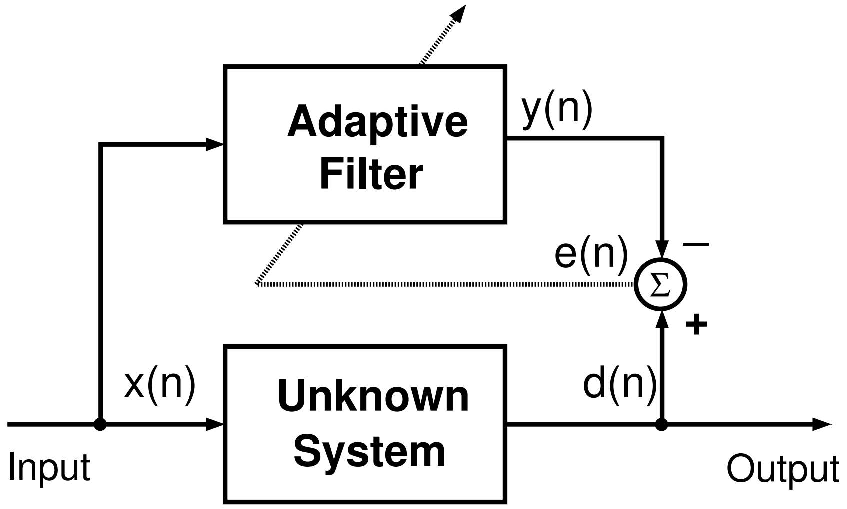
Wiener_Chirp_Denoising

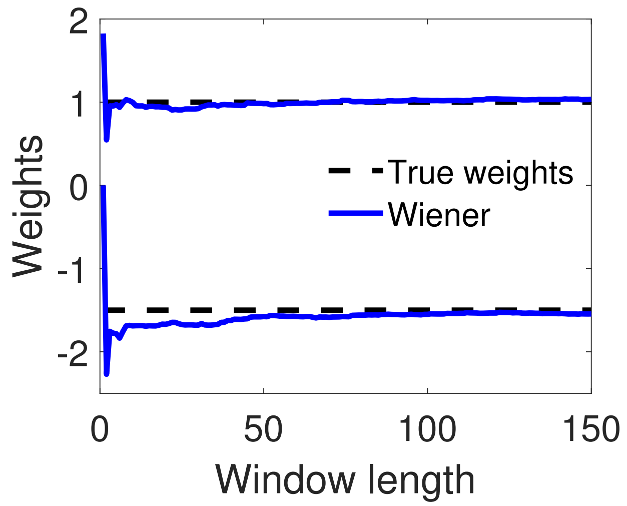
Set up: Consider the system model $d(n) = x(n) - 1.5x(n-1) + q(n)$ where $x \sim \mathcal{N}(0,1)$ and noise $q \sim \mathcal{N}(0, 0.16)$

Task: Find the "system coefficients", $\mathbf{w}_o = [1, -1.5]^T$, from the available noisy observations, $d(n)$, without any knowledge about $\mathbf{w}_o$ or q

Solution: We can write $d(n) = \mathbf{w}_o^T \mathbf{x}(n) + q(n)$

and use the Wiener filter in the System Identification (SYS-ID) setting to find the unknown system coefficients as $\mathbf{w}_o = \mathbf{R}^{-1}\mathbf{r}_{d\mathbf{x}}$.





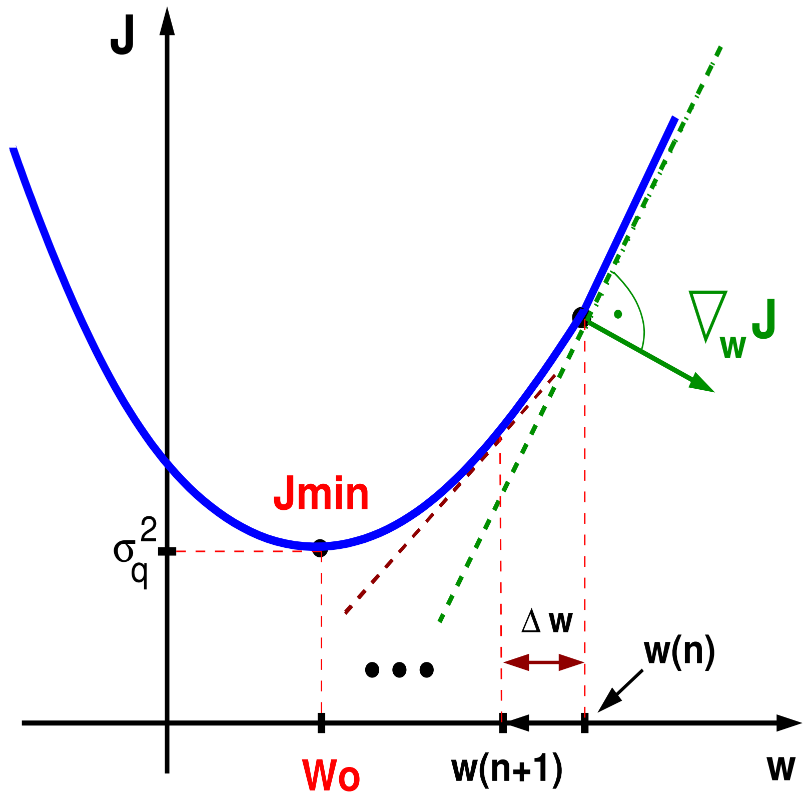
☞ Observe that the performance of the Wiener filter depends on filter length

Method of steepest descent: Iterative Wiener solution

we reach w_o through iterations $\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta\mathbf{w}(n) = \mathbf{w}(n) - \mu\nabla_{\mathbf{w}}J(n)$

Problem with the Wiener filter: it is computationally demanding to calculate the inverse of a possibly large correlation matrix $\mathbf{R}_{xx}$.

Solution: Allow the weights to have a **time–varying** form, so that they can be adjusted in an **iterative** fashion along the error surface.



This is achieved in the direction of steepest descent of error surface, that is, in a **direction opposite to the gradient vector** whose elements are defined by $\nabla_{w_k} J, \quad k = 1, 2, \ldots, p$.

For a teaching signal, assume

$$d(n) = \mathbf{x}^T(n)\mathbf{w}_o + q(n),$$

where $q \in \mathcal{N}(0, \sigma_q^2)$, so that we have $J_{min} = \sigma_q^2$

Method of steepest descent $\looparrowright$ continued

The gradient of the error surface of the filter wrt the weights now takes a **time varying** form

$$\nabla_{w_k} J(n) = -r_{dx}(k) + \sum_{j=1}^{p} w_j(n) r_x(j,k) \qquad (*)$$

where the indices j, k refer e.g. to locations of different sensors in space, while the index n refers to iteration number.

According to the method of steepest descent, the adjustment applied to the weight $w_k(n)$ at iteration n, **called the weight update**, $\Delta w_k(n)$, is defined **along the direction of the negative of the gradient**, as

$$\Delta w_k(n) = -\mu \nabla_{w_k} J(n), \qquad k = 1, 2, \ldots, p$$

where μ is a small positive constant, $\mu \in \mathbb{R}^+$, called the **learning rate** parameter (also called step size, usually denoted by μ or η).

Method of steepest descent: Final form

Recall $\nabla_{w_k} J(n) = -r_{dx}(k) + \sum_{j=1}^{p} w_j(n) r_x(j,k)$ **or** $\nabla_{\mathbf{w}} J(n) = -\mathbf{r}_{dx} + \mathbf{R}\,\mathbf{w}$

Given the **current** value of the kth weight $w_k(n)$ at iteration n, the **updated** value of this weight at the next iteration $(n+1)$ is computed as

$$w_k(n+1) = w_k(n) + \Delta w_k(n) = w_k(n) - \mu \nabla_{w_k} J(n)$$

a vector form $$\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta \mathbf{w}(n) = \mathbf{w}(n) - \mu \nabla_{\mathbf{w}} J(n)$$

☞ **updated filter weights** = **current weights** + **weight update**

Upon combining with (*) from the previous slide, we have

$$w_k(n+1) = w_k(n) + \mu \left[r_{dx}(k) - \sum_{j=1}^{p} w_j(n) r_x(j,k) \right], \quad k = 1, \ldots, p$$

or in a vector form: $$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu \big[\mathbf{r}_{dx} - \mathbf{R}\,\mathbf{w}(n) \big] \qquad (**)$$

The SD method is **exact** in that no approximations are made in the derivation $\looparrowright$ the key difference is that the solution is obtained iteratively.

Observe that there is no matrix inverse in the update of filter weights!

Method of steepest descent: Have you noticed?

We now have an adaptive parameter estimator in the sense

new parameter estimate = old parameter estimate + update

The derivation is based on minimising the mean squared error

$$J(n) = \frac{1}{2} E\{e^2(n)\}$$

For a **spatial filter** (sensor array), this cost function is an ***ensemble average*** taken at a time instant n, over an ensemble of *s*patial filters (e.g. nodes in sensor network).

For a **temporal filter**, the SD method can also be derived by minimising the ***sum of error squares***

$$\mathcal{E}_{total} = \sum_{i=1}^{n} \mathcal{E}(i) = \frac{1}{2} \sum_{i=1}^{n} e^2(i)$$

In this case the ACF etc. are defined as **time averages** rather than ensemble averages. If the physical processes considered are **jointly ergodic** then we are justified in substituting time averages for ensemble averages.

The role of learning rate (also called 'step size')

the step size governs the behaviour of gradient descent algorithms

Care must be taken when selecting the learning rate μ, because:

- For μ small enough, the method of SD converges to a stationary point of the cost function $J(e) \equiv J(\mathbf{w_o})$, for which $\nabla_{\mathbf{w}} J(\mathbf{w}_o) = \mathbf{0}$. This stationary point can be a local or a global minimum.
- The method of steepest descent is an iterative procedure, and its behaviour depends on the value assigned to the step–size parameter μ.
- When μ is small compared to a certain critical value μ_{crit}, the trajectory traced by the weight vector $\mathbf{w}(n)$ for increasing number of iterations, n, tends to be monotonic.
- When μ is allowed to approach (but remain less than) the critical value μ_{crit}, the trajectory is oscillatory or overdamped.
- When μ exceeds μ_{crit}, the trajectory becomes unstable.

Condition $\mu < \mu_{crit}$ corresponds to a convergent or stable system, whereas condition $\mu > \mu_{crit}$ corresponds to a divergent or unstable system. Therefore, finding μ_{crit} defines a stability bound. (see Slide 30)

The Least Mean Square (LMS) algorithm

unlike the block-based steepest desc., LMS operates in a real-time online fashion

Problem: The need for full second order statistics so far, e.g.

steepest descent: $$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\big[\mathbf{r}_{d\mathbf{x}} - \mathbf{R}\,\mathbf{w}(n)\big]$$

However, in unknown environments or for streaming data, the correlations $\mathbf{r}_{d\mathbf{x}}, \mathbf{R}$ are either not readily available or are time-consuming to compute.

Solution: The Least Mean Square (LMS) employs **instantaneous estimates** of the autocorrelation, $\mathbf{R}_{xx}$, and crosscorrelation, $\mathbf{r}_{d\mathbf{x}}$, given by

$$\hat{\mathbf{R}}(n) = \mathbf{x}(n)\mathbf{x}^T(n) \qquad \hat{\mathbf{r}}_{d\mathbf{x}}(n) = d(n)\mathbf{x}(n) \qquad J(n) = \frac{1}{2}e^2(n)$$

Substitute these into the equation for steepest descent above, to arrive at

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) + \mu\big[d(n)\mathbf{x}(n) - \mathbf{x}(n)\underbrace{\mathbf{x}^T(n)\mathbf{w}(n)}_{y(n)}\big] \\ &= \mathbf{w}(n) + \mu[\underbrace{d(n) - y(n)}_{e(n)}]\mathbf{x}(n) \end{aligned}$$

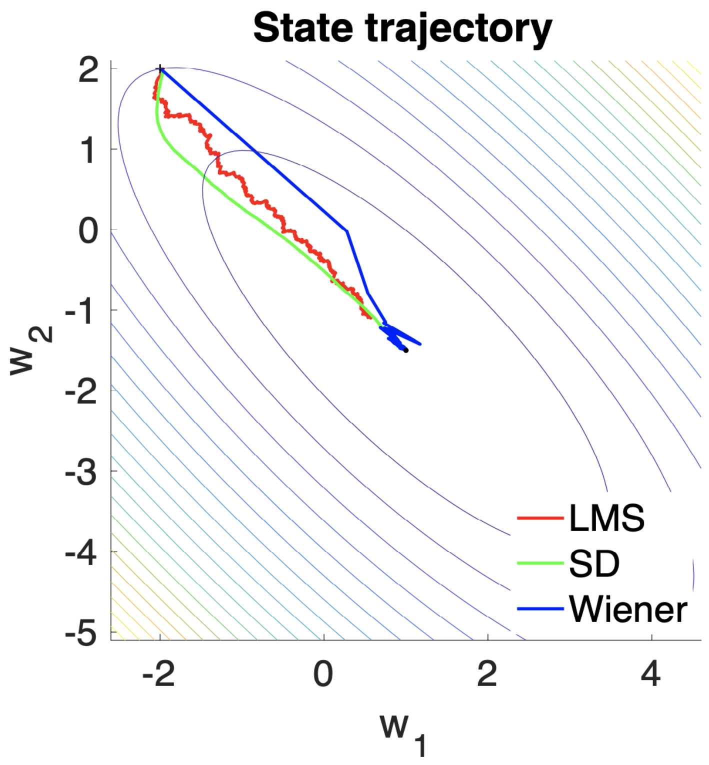
Finally, the LMS: $$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n)$$

Because of the 'instantaneous statistics' within the LMS, the weights follow a "zig-zag" path, converging to the optimum solution $\mathbf{w}_0$, if μ is chosen properly, as in Example 3.

Example 2: Wiener filter vs Steepest Descent vs LMS

LMS_SD_Wiener_OnePlot.m

Task: Adaptive SYS-ID of MA(2) system $y_n = x_n - 1.5x_{n-1} + q_n, \quad x, q \sim \mathcal{N}(0,1)$.
Filter length, $L = 2$, $\mu_{LMS} = 0.1$, $\mu_{SD} = 0.01$. LMS took longer to converge.



Steepest descent produces smoother learning trajectories, but LMS is computationally much cheaper to implement.

Summary of the LMS algorithm

(simple yet effective, also it operates in an "unknown" environment)

- The LMS operates in "unknown" environments, and the weight vector follows a **random** trajectory along the error performance surface
- Along the iterations, as $n \rightarrow \infty$ (steady state) the weights perform a random walk about the optimal solution $\mathbf{w}_0$ (measure of MSE)
- The cost function of LMS is based on an instantaneous estimate of the squared error. Consequently, the gradient vector in LMS is "random" and its direction accuracy improves "on the average" with increasing n

The LMS summary:

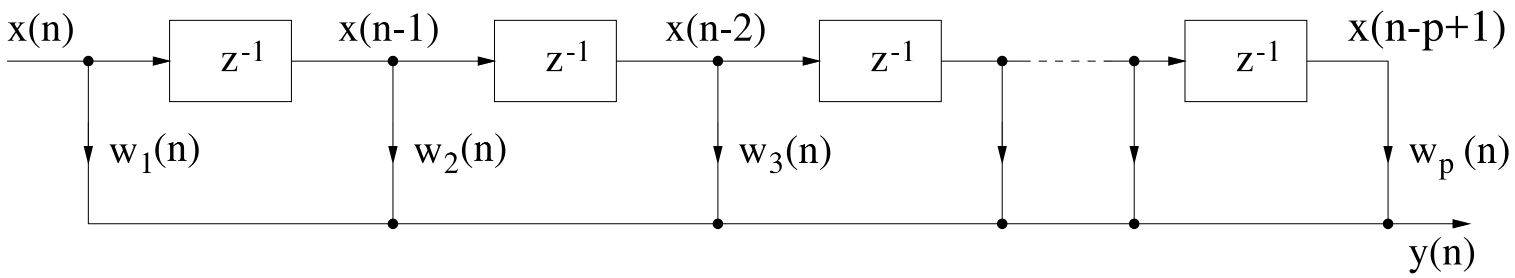
Initialisation. $w_k(0) = 0, \quad k = 1, \ldots, p \qquad \equiv \qquad \mathbf{w}(0) = \mathbf{0}$

Filtering. For $n = 1, \ldots, (\infty)$ compute

$$\begin{aligned} y(n) &= \sum_{j=1}^{p} w_j(n)x_j(n) = \mathbf{x}^T(n)\mathbf{w}(n) = \mathbf{w}^T(n)\mathbf{x}(n) \\ \mathbf{w}(n+1) &= \mathbf{w}(n) + \mu e(n)\mathbf{x}(n) \end{aligned}$$

Further perspective on LMS: Temporal problems

Our spatial problem with sensors signals: $\mathbf{x}(n) = [x_1(n), \ldots, x_p(n)]^T$ becomes a temporal one where $\mathbf{x}(n) = [x(n), \ldots, x(n-p+1)]^T$.



The output of this **temporal** filter of memory p is $y(n) = \mathbf{w}^T(n)\mathbf{x}(n)$

Alternative derivation of LMS: Since e(n)=d(n)-y(n)=$d(n)-\mathbf{w}^T(n)\mathbf{x}(n)$, then

$$J(n) = \frac{1}{2}\, e^2(n) \quad \rightarrow \quad \nabla_{\mathbf{w}} J(n) = \frac{\partial J(n)}{\partial \mathbf{w}(n)} = \frac{1}{2}\frac{\partial e^2(n)}{\partial e(n)}\frac{\partial e(n)}{\partial y(n)}\frac{\partial y(n)}{\partial \mathbf{w}(n)}$$

These partial gradients can be evaluated as

$$\frac{\partial e^2(n)}{\partial e(n)} = 2e(n), \quad \frac{\partial e(n)}{\partial y(n)} = -1, \quad \frac{\partial y(n)}{\partial \mathbf{w}(n)} = \mathbf{x}(n) \quad \Rightarrow \quad \frac{\partial e(n)}{\partial \mathbf{w}(n)} = -\,\mathbf{x}(n)$$

Finally, we obtain the same LMS equations

Finally

$$\frac{\partial J(n)}{\partial \mathbf{w}(n)} = -\,e(n)\mathbf{x}(n)$$

The set of equations that describes the LMS is therefore given by

$$\begin{aligned} y(n) &= \sum_{i=1}^{p} x_i(n)w_i(n) = \mathbf{w}^T(n)\mathbf{x}(n) \\ e(n) &= d(n) - y(n) \\ \mathbf{w}(n+1) &= \mathbf{w}(n) + \mu\, e(n)\mathbf{x}(n) \end{aligned}$$

∘ The LMS algorithm is a very simple yet extremely popular algorithm for adaptive filtering.

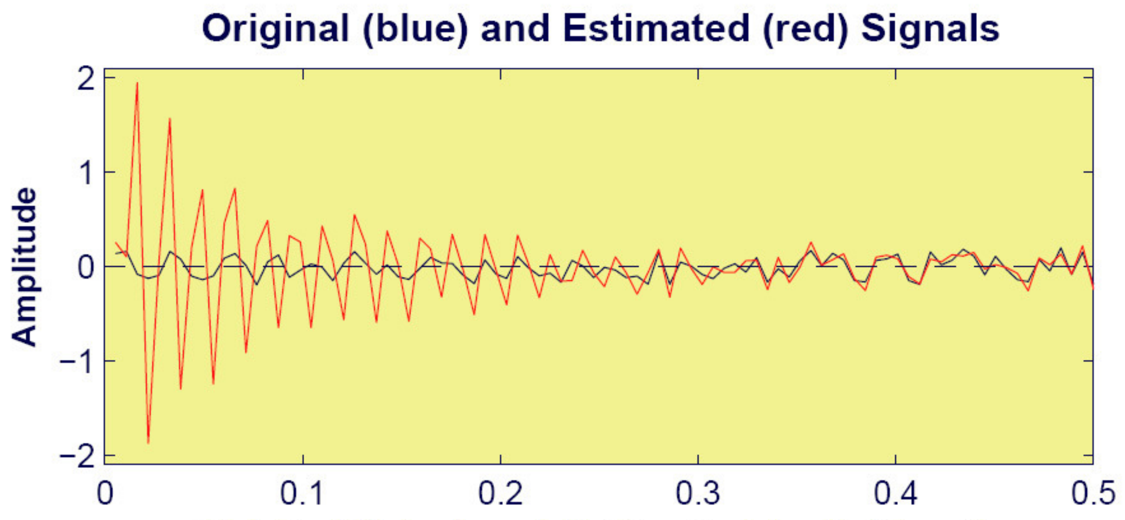
∘ LMS is robust (optimal in H^∞ sense) which justifies its practical utility.

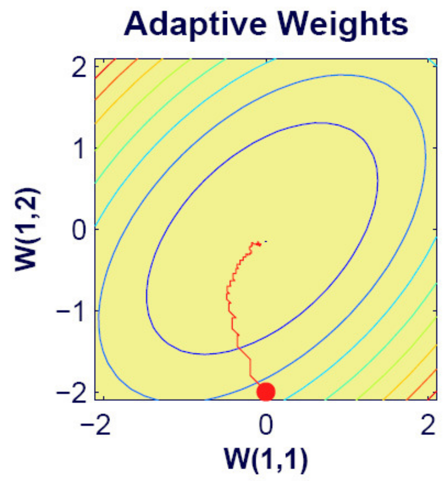
∘ The forms of the spatial and temporal LMS are identical $\looparrowright$ it us up to us to apply adaptive filters according to the problem in hand.

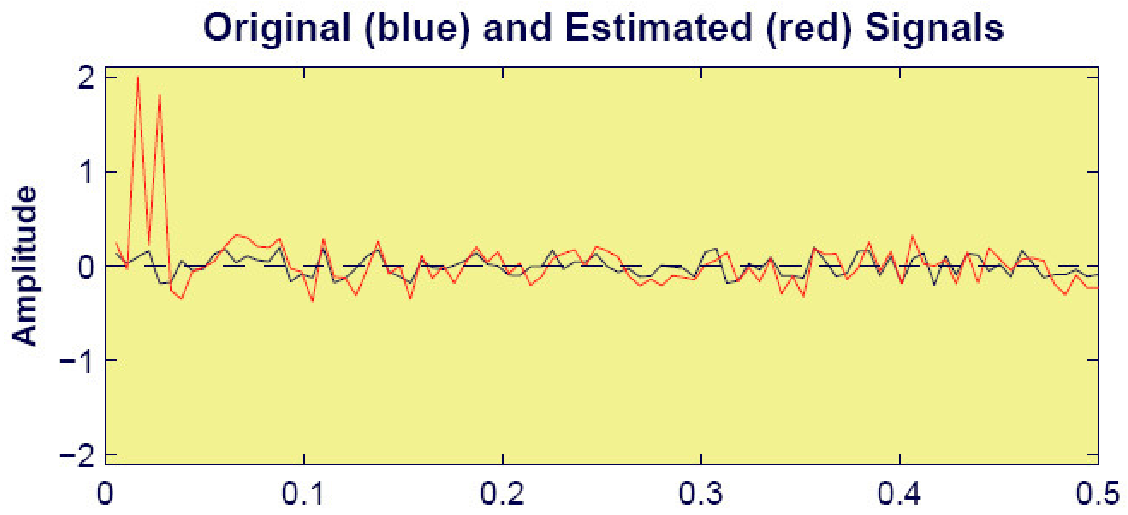
Example 3: LMS convergence vs. stepsize μ

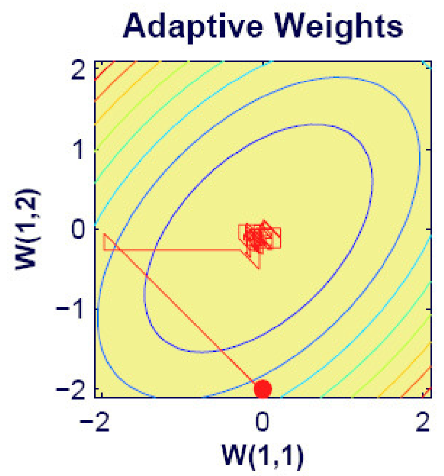
(Matlab function 'nnd10nc' or 'LMS_Contour_Convergence_nnd10nc_Mooh.m')

Original signal and its prediction Error contour surface









Top panel $\rightsquigarrow$ learning rate $\mu = 0.1$ Bottom panel $\rightsquigarrow$ learning rate $\mu = 0.9$

Convergence of LMS $\looparrowright$ parallels with MVU estimation.

The unknown vector parameter is the optimal filter weight vector $\mathbf{w}_o$

- **Convergence in the mean $\rightsquigarrow$ bias in parameter estimation** (think of the requirement for an unbiased optimal weight estimate)

$$E\{\mathbf{w}(n)\} \rightarrow \mathbf{w}_0 \qquad as \quad n \rightarrow \infty \quad \text{(steady state)}$$

- **Convergence in the mean square (MSE) $\rightsquigarrow$ estimator variance**, (fluctuation of the instantaneous weight vector estimates around $\mathbf{w}_o$)

we desire $$E\{e^2(n)\} \rightarrow \text{constant} \qquad \text{as} \quad n \rightarrow \infty \quad \text{(steady state)}$$

We can write this since the error is a function of the filter weights.

☞ We expect the MSE convergence condition to be tighter: If LMS is convergent in the mean square, then it is convergent in the mean. The converse is not necessarily true (if an estimator is unbiased $\looparrowright$ it is not necessarily minimum variance $\leftrightsquigarrow$ if it is min. var. $\looparrowright$ likely unbiased).

☞ The logarithmic plot of the mean squared error (MSE) along time, $10\log e^2(n)$ is called the **learning curve**.

For more on learning curves see your Coursework booklet and Slide 29

Example 4: Learning curves and performance measures

Task: Adaptively identify an AR(2) system given by

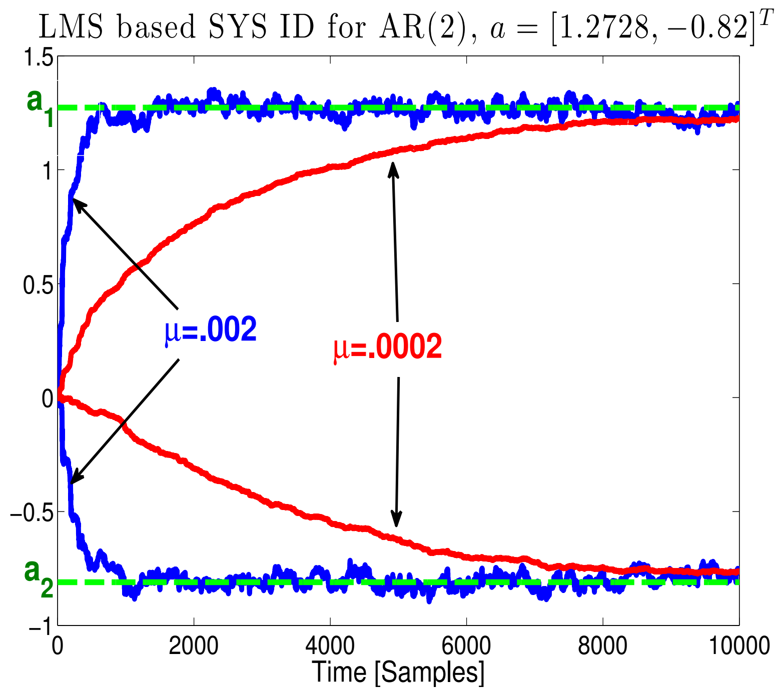
$$x(n) = 1.2728x(n-1) - 0.81x(n-2) + q(n), \quad q \sim \mathcal{N}(0, \sigma_q^2)$$

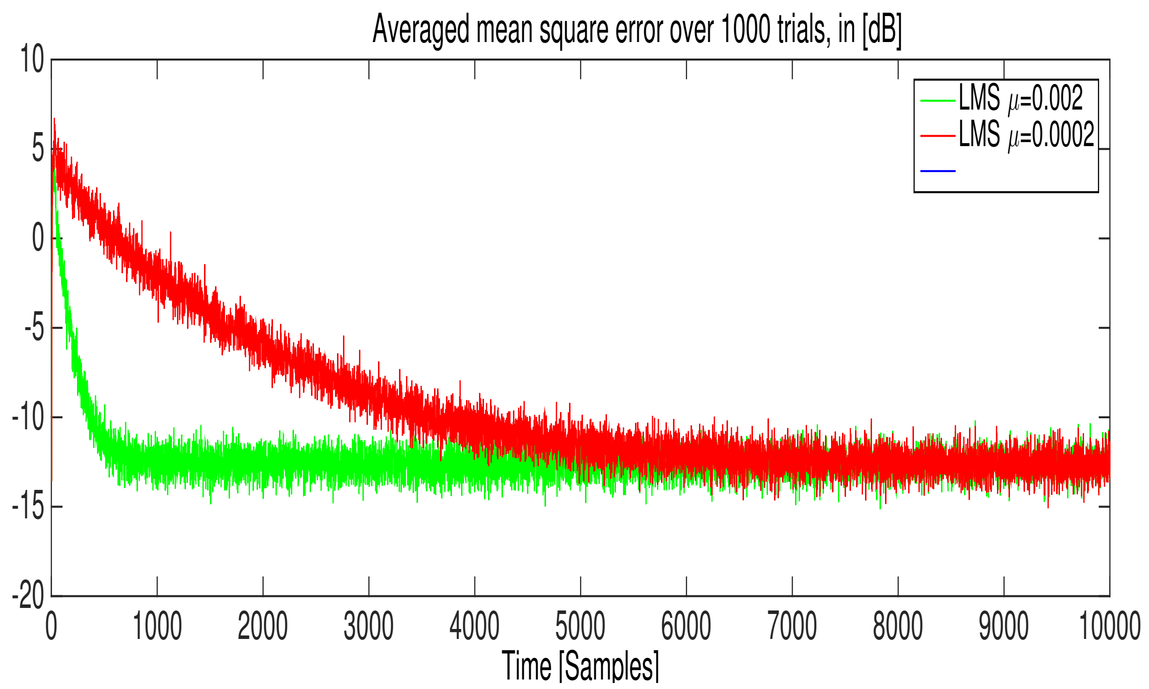
Adaptive system identification (SYS-ID) is performed based on:

LMS system model: $\hat{x}(n) = w_1(n)x(n-1) + w_2(n)x(n-2)$

LMS weights: (see slide 48 for the normalised LMS (NLMS))

LMS weights (i=1,2): $w_i(n+1) = w_i(n) + \mu e(n)x(n-i)$





Convergence analysis of LMS: Mean– and Mean–Square convergence of the weight vector   Contour_Plot, LMS_ConditionN

1) **Convergence in the mean**. Assume that the weight vector is uncorrelated with the input vector, $E\{\mathbf{w}(n)\mathbf{x}(n)\} = \mathbf{0}$, and $d \perp x$ (the usual "independence" assumptions but not true in practice)

Then, from Slide 19 $E\{\mathbf{w}(n+1)\} = \big[\mathbf{I} - \mu\mathbf{R}_{xx}\big]E\{\mathbf{w}(n)\} + \mu\mathbf{r}_{dx}$

The condition of convergence in the mean becomes (for an i.i.d input)

$$0 < \mu < \frac{2}{\lambda_{max}} \qquad \text{(see Appendix 4)}$$

where λ_{max} is the largest eigenvalue of the autocorrelation matrix $\mathbf{R}_{xx}$.

2) **Mean square convergence.** Analysis is more complicated and gives

$$\mu\sum_{k=1}^{p}\frac{\lambda_k}{1-\mu\lambda_k} < 1 \;\approx\; \text{bounded as} \quad 0 < \mu < \frac{2}{tr[\mathbf{R}_{xx}]} \quad \text{using} \quad tr[\mathbf{R}_{xx}] = \sum_{k=1}^{p}\lambda_k$$

Since **"The trace" = "Total Input Power"**, we have

$$0 < \mu < \frac{2}{\text{total input power}} = \frac{2}{p\,\sigma_x^2}$$

Adaptive filtering configurations
ways to connect the filter, input, and teaching signal

- LMS can operate in a stationary or nonstationary environment
- LMS not only seeks for the minimum point of the error surface, but it also *tracks* it if $\mathbf{w}_o$ is time–varying
- The smaller the stepsizse μ the better the tracking behaviour (at the steady state, in the MSE sense), however, this means slow adaptation.

Adaptive filtering configurations:

⊛ **Linear prediction.** The set of past values serves as the input vector, while the current input sample serves as the desired signal.

⊛ **Inverse system modelling.** The adaptive filter is connected in series with the unknown system, whose parameters we wish to estimate.

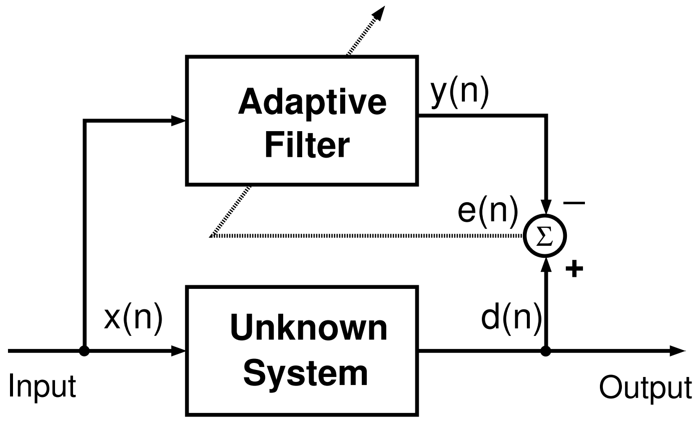
⊛ **Noise cancellation.** Reference noise serves as the input, while the measured noisy signal serves as the desired response, $d(n)$.

⊛ **System identification.** The adaptive filter is connected in parallel to the unknown system, and their outputs are compared to produce the estimation error which drives the adaptation.

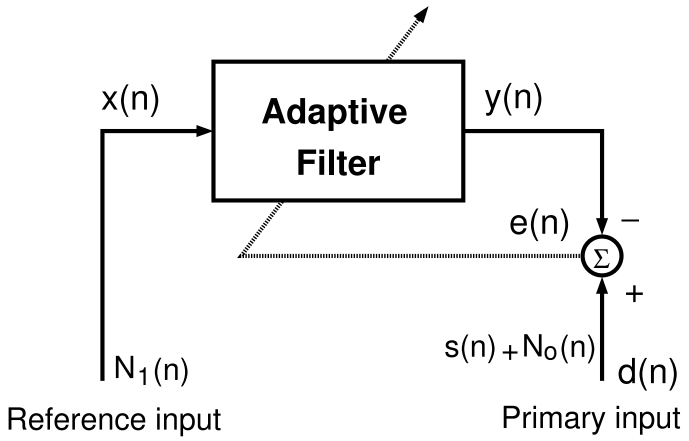
Adaptive filtering configurations: Block diagrams

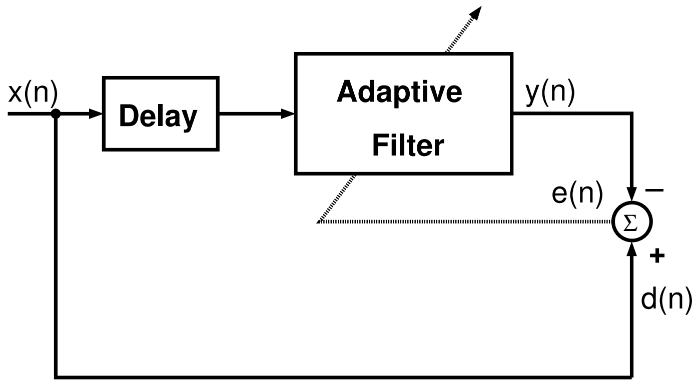
the same learning algorithm, e.g. the LMS, operates for any configuration

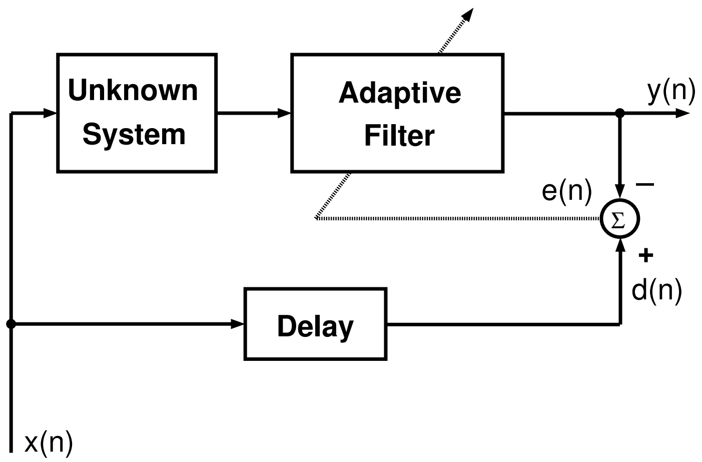
System identification



Noise cancellation







Adaptive prediction

Inverse system modelling

Applications of adaptive filters

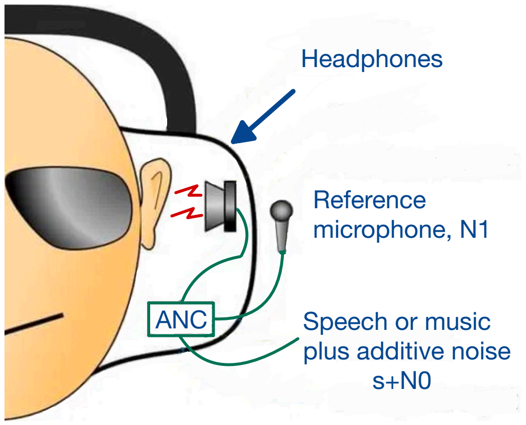
Adaptive filters have found an enormous number of applications.

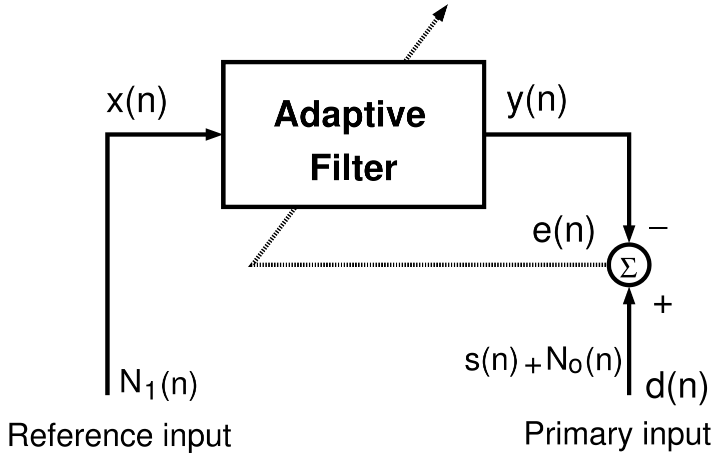
1. **Forward prediction** (the desired signal is the input signal advanced relative to the input of the adaptive filter). Applications in financial forecasting, wind prediction in renewable energy, power systems
2. **System identification** (the adaptive filter and the unknown system are connected in parallel and are fed with the same input signal $x(n)$). Applications in acoustic echo cancellation, feedback whistling removal in teleconference scenarios, hearing aids, power systems
3. **Inverse system modelling** (adaptive filter cascaded with the unknown system), as in channel equalisation in mobile telephony, wireless sensor networks, underwater communications, mobile sonar, mobile radar
4. **Noise cancellation** (the only requirement is that the noise in the primary input and the reference noise are correlated), as in noise removal from speech in mobile phones, denoising in biomedical scenarios, concert halls, hand-held multimedia recording

From Lecture 6: Adaptive noise canc. with reference

(such as in noise-canceling headphones on an airplane)

In the adaptive noise cancellation configuration (below right), the variables in the adaptive filter have the following roles.





Input to the filter, is the Reference Noise signal, that is, $x(n) = N_1(n)$. The only requirement is that N_1 is correlated with the measurement noise, N_0, but not with the signal of interest, $s(n)$. The filter aims to estimate N_0 from N_1, that is, $y = \hat{N}_0$.

Teaching signal, $d(n)$, is the noise-polluted signal of interest, $s(n) + N_0(n)$, which serves as the Primary Input to the filter. Since $s \perp N_1$, the filter can only yield $y = \hat{N}_0$.

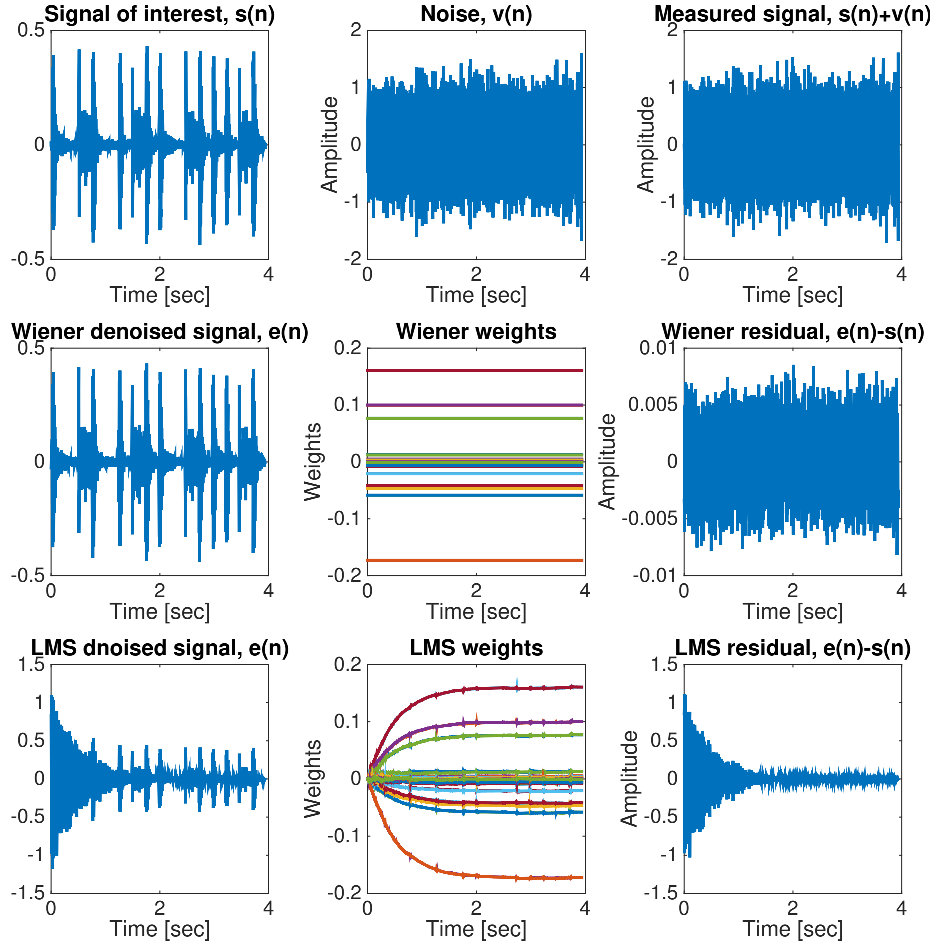
Filter output, $y = \hat{N}_0$, provides the best MSE estimate of the measurement noise, N_0, from the reference noise, N_1. The more correlated N_1 and N_0 the faster the convergence.

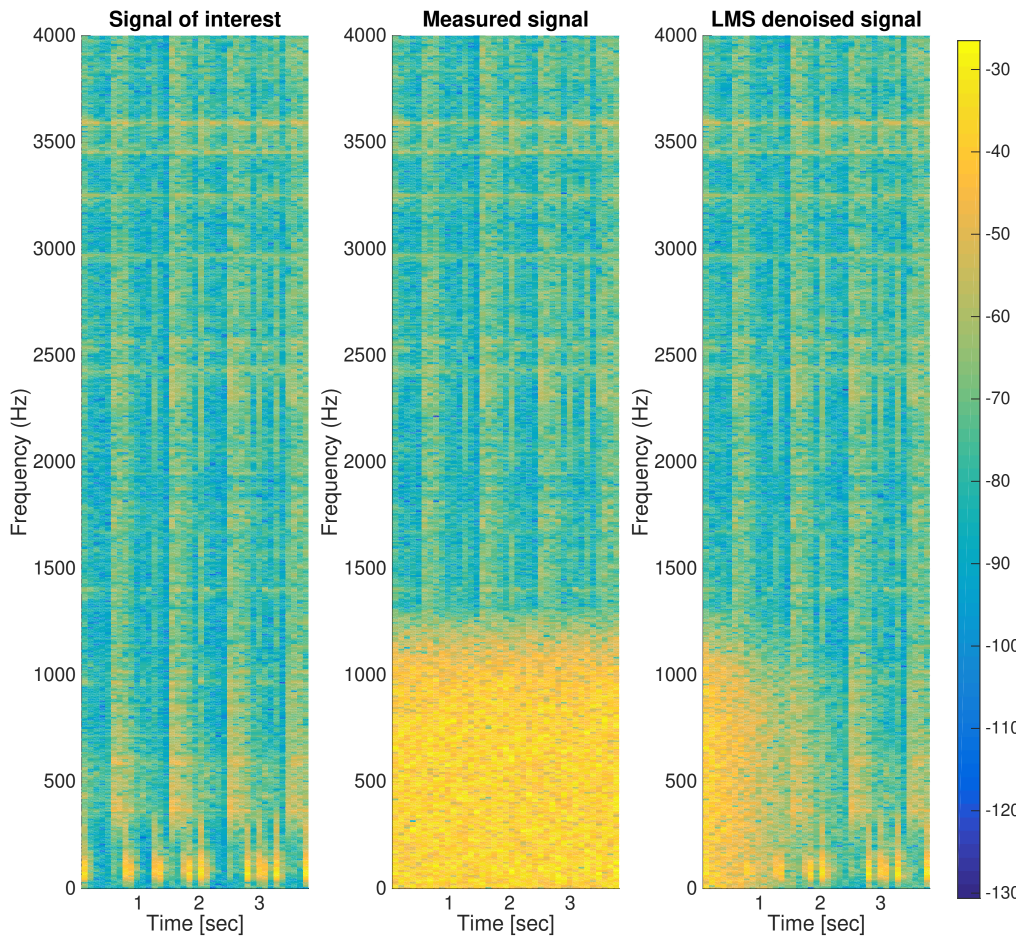
Output error, $e = s + N_0 - \hat{N}_0$, serves as a **"system output"**, whereby the adaptive filter aims to achieve $e \approx s$. In other words, the standard e serves as an output, $e = \hat{s}$.

Example 5: Adaptive LMS cancellation of cockpit noise

ALE_Handel, Denoising_Reference_Drum_WienerAndLMS

Consider an adaptive noise cancellation problem, like that in noise cancelling headphones when you are listening to music on the plane.

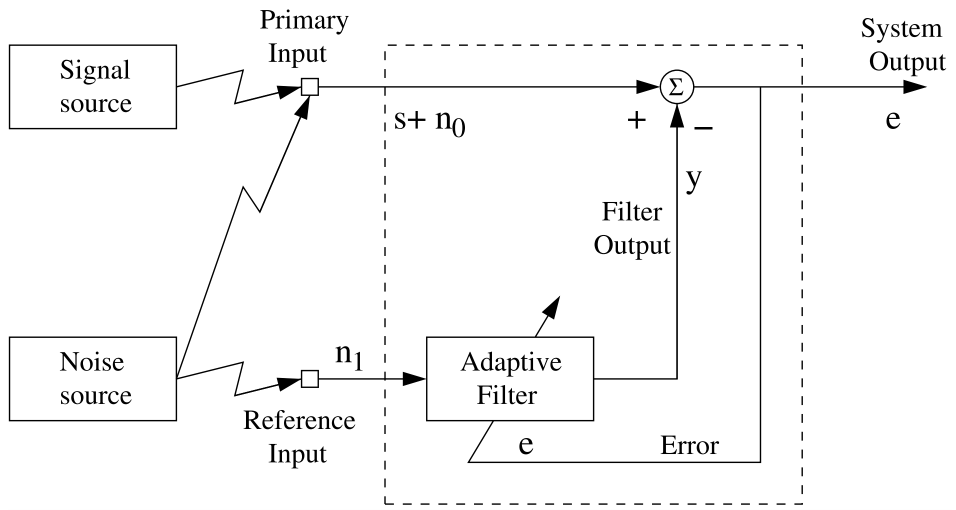


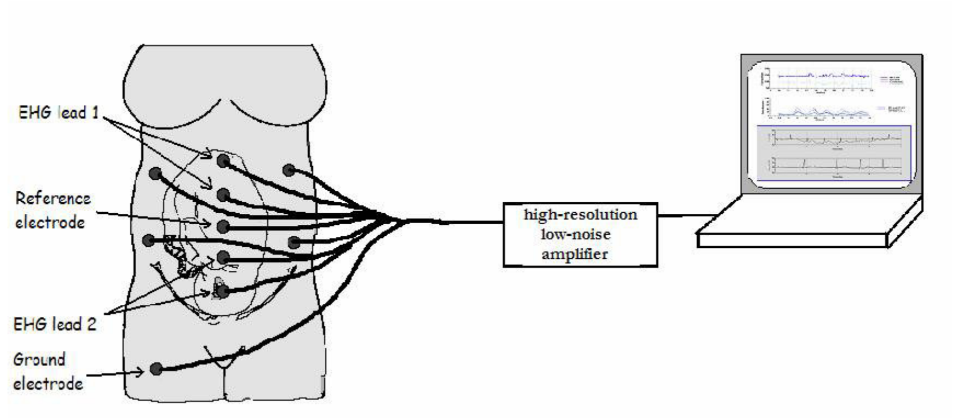


Right: The time–frequency representation of the performance of the LMS algorithm

Example 6: Noise cancellation for foetal ECG recovery
Data acquisition

ANC with Reference





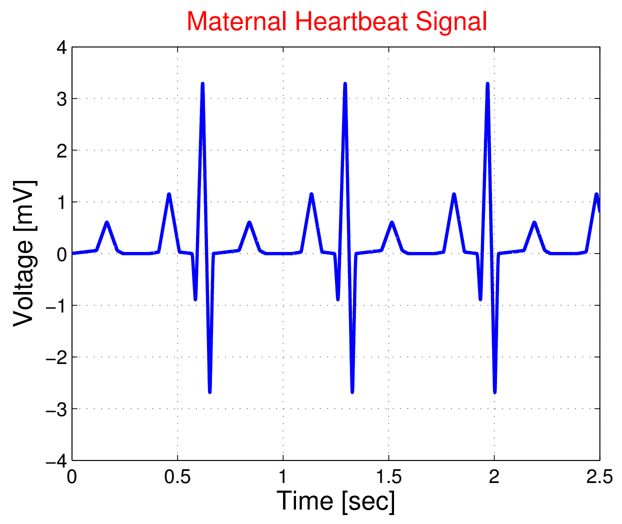
ECG recording (Reference electrode $\neq$ Reference input)

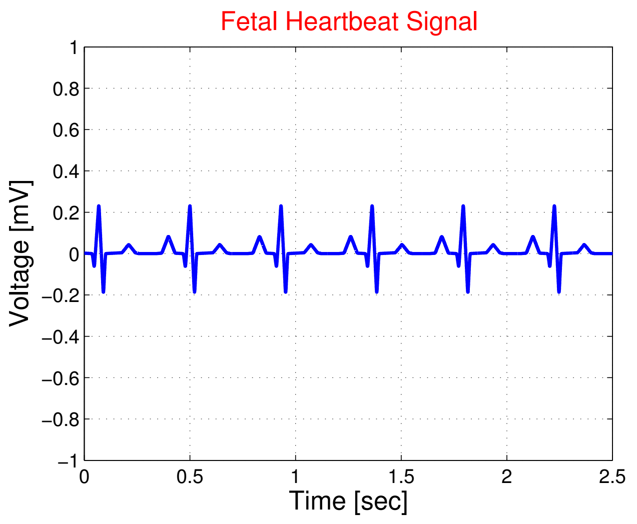
Example 6: Noise cancellation for foetal ECG estimation

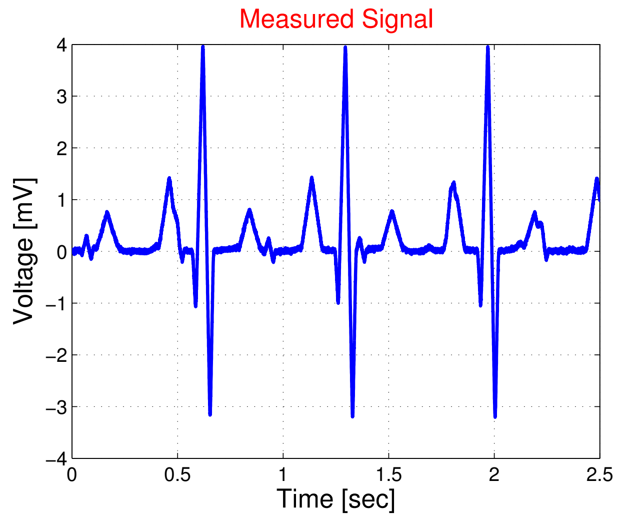
(similar to your CW Assignment IV)

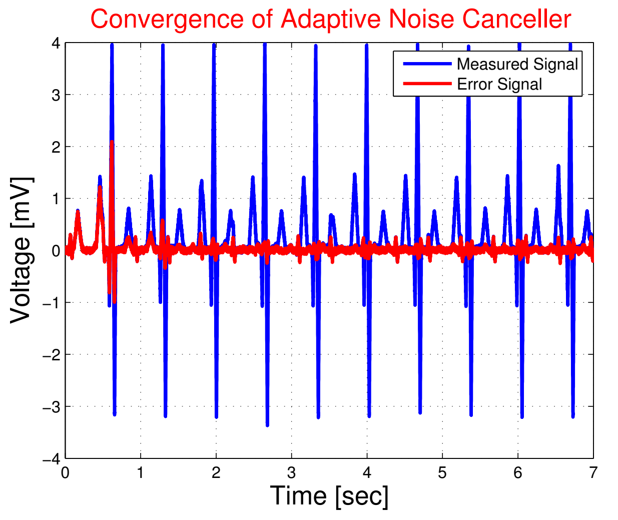
Maternal ECG signal

Foetal heartbeat







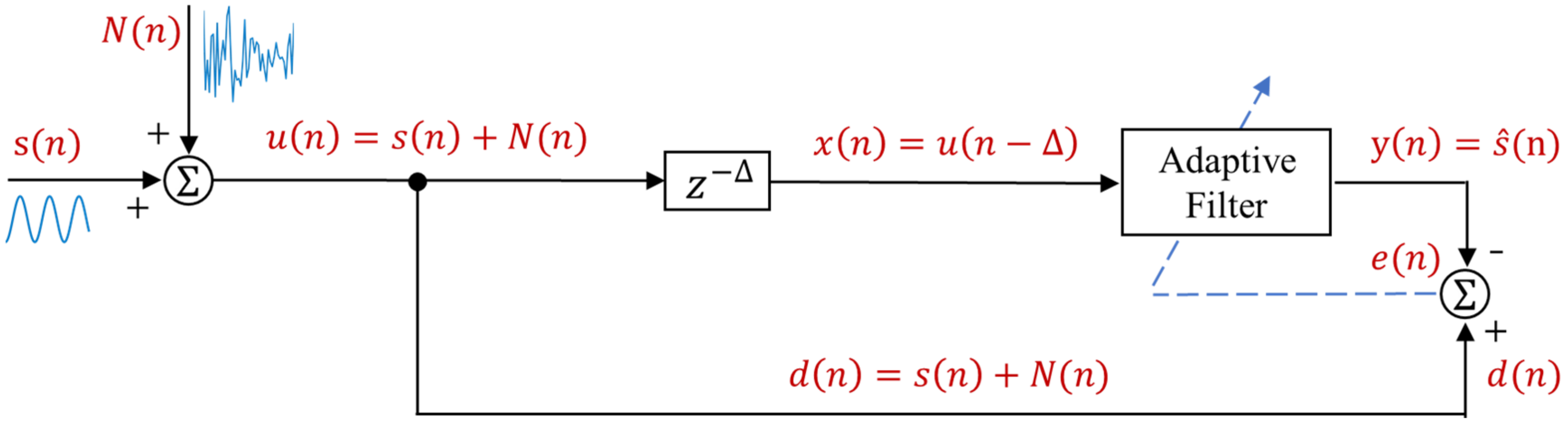


Measured foetal ECG

Maternal and foetal ECG

Example 7: Adaptive Line Enhancement $\looparrowright$ ANC with no reference

'lms_fixed_demo', 'Denoising_Adaptive_GUI', 'ALE_Handel'



Enhancement of a 100Hz signal in band–limited WN, with a $p = 30$ FIR LMS filter

- Adaptive line enhancement (ALE) refers to the case where a noisy signal, $u(n) = \sin(n) + w(n)$, is filtered to obtain $y(n) = \hat{s}(n) = \sin(n)$
- ALE consists of a de-correlation stage, denoted by $z^{-\Delta}$ and an adaptive predictor. The value of Δ should be greater than the ACF lag for w
- The de-correlation stage attempts to remove any correlation that may exist between the samples of noise, by shifting them Δ samples apart
- This results in a phase shift at the output (input lags Δ steps behind)

Lecture summary

- Adaptive filters are simple, yet very powerful, estimators which do not require any assumptions on the data, and adjust their coefficients in an online adaptive manner according the the minimisation of MSE
- In this way, they reach the optimal Wiener solution in a recursive fashion
- The steepest descent, LMS, NLMS, sign-algorithms etc. are **learning algorithms** which operate in certain **adaptive filtering configurations**
- Within each configuration, the function of the filter is determined by the way the input and teaching signal are connected to the filter (prediction, system identification, inverse system modelling, noise cancellation)
- The online adaptation makes adaptive filters suitable to operate in nonstationary environments, a typical case in practical applications
- Applications of adaptive filters are found everywhere (mobile phones, audio devices, biomedical, finance, seismics, radar, sonar, ...)
- Many more complex, models are based on adaptive filters (neural networks, deep learning, reservoir computing, etc.)
- Adaptive filters are indispensable for streaming Big Data

The next several slides introduce the concept of an artificial neuron – the building block of neural networks

Lecture supplement:

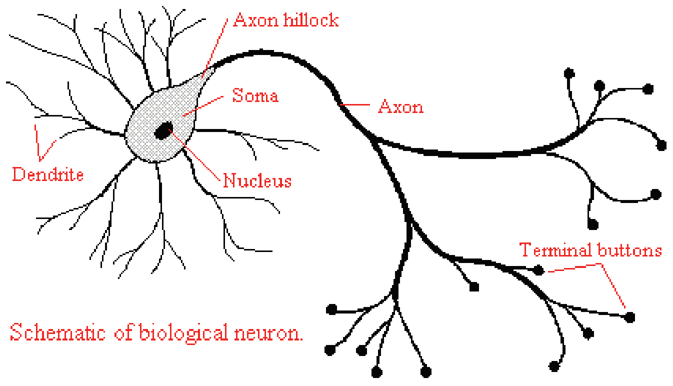
Elements of neural networks

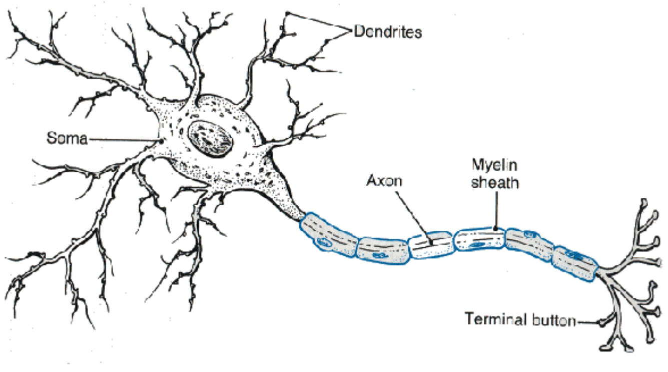
The need for nonlinear structures

There are situations in which the use of linear filters and models is suboptimal:

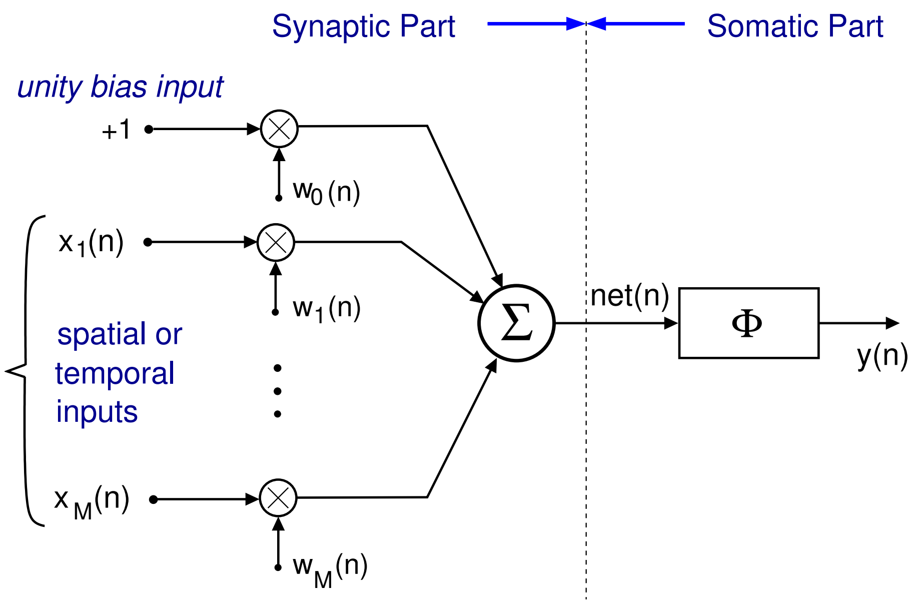
- When trying to identify dynamical signals/systems observed through a saturation type sensor nonlinearity, the use of linear models will be limited
- When separating signals with overlapping spectral components
- Systems which are naturally nonlinear or signals that are non-Gaussian, such as limit cycles, bifurcations and fixed point dynamics, cannot be captured by linear models
- Communications channels, for instance, often need nonlinear equalisers to achieve acceptable performance
- Signals from humans (ECG, EEG, ...) are typically nonlinear and physiological noise is not white $\rightsquigarrow$ it is the so-called 'pink noise' or 'fractal noise' for which the spectrum $\sim 1/f$

An artificial neuron $\looparrowright$ a nonlinear adaptive filter





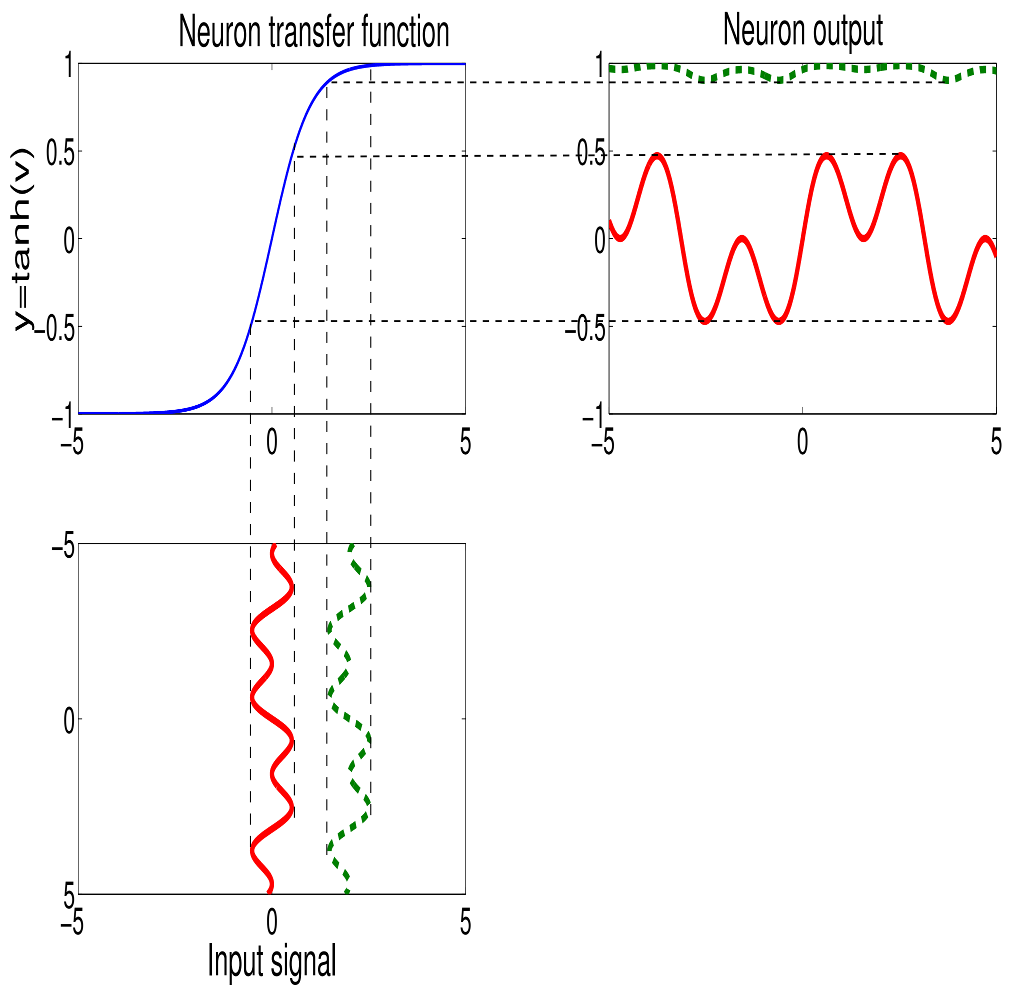
Biological neuron



Model of an artificial neuron

- delayed inputs $\mathbf{x}$
- bias input with unity value
- sumer and multipliers
- output nonlinearity (logistic, tanh, atan)

Effect of nonlinearity: An artificial neuron



Input: two identical signals with different DC offsets

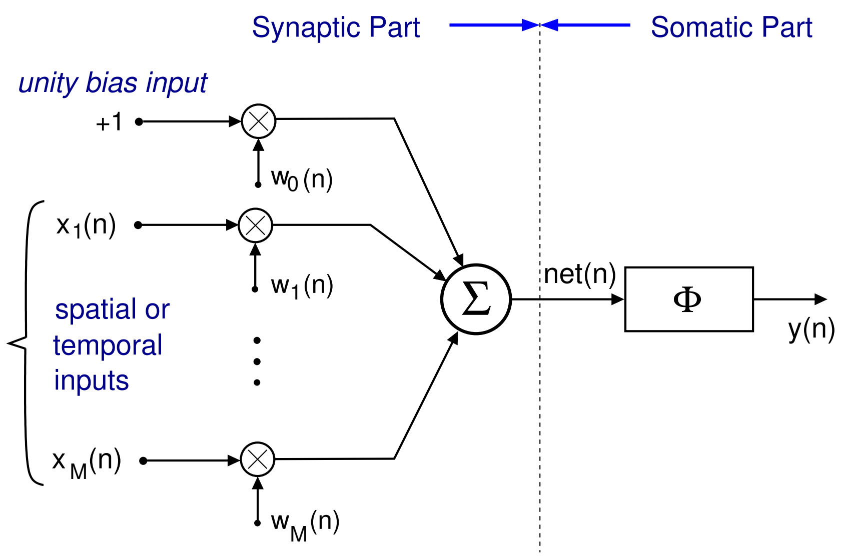
- Observe the different behaviour depending on the operating point
- The output behaviour varies from amplifying and slightly distorting the input signal to attenuating and considerably distorting
- From the viewpoint of system theory, neural networks represent nonlinear maps, mapping one metric space to another.

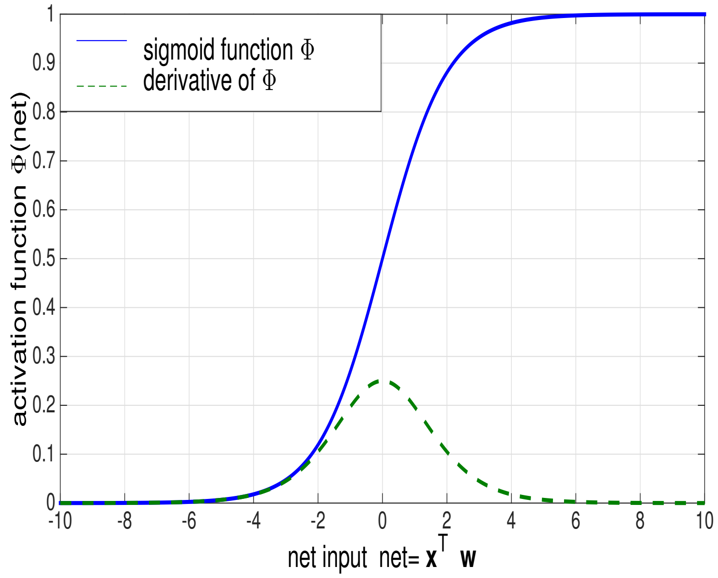
A simple nonlinear structure, referred to as the perceptron, or dynamical perceptron

- Consider a simple nonlinear FIR filter

$$\text{Nonlinear FIR filter} = \text{standard FIR filter} + \text{memoryless nonlinearity}$$

- This nonlinearity is of a saturation type, like $\tanh$, $\arctan$
- This structure can be seen as a single neuron with a dynamical FIR synapse. This FIR synapse provides memory to the neuron.

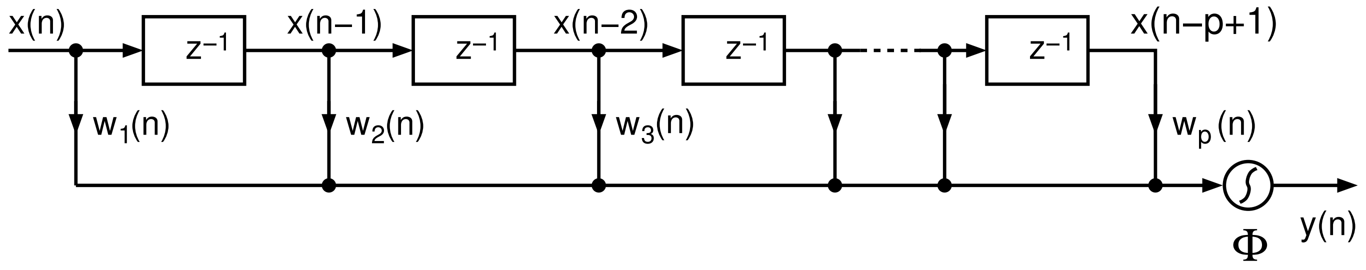




Model of artificial neuron (dynamical perceptron, nonlinear FIR filter)

Model of artificial neuron for temporal data (for simplicity, the bias input is omitted)

This is the adaptive filtering model of every single neuron in our brains



The output of this filter is given by

$$y(n) = \Phi\left(\mathbf{w}^T(n)\mathbf{x}(n)\right) = \Phi\big(net(n)\big) \quad \text{where} \quad net(n) = \mathbf{w}^T(n)\mathbf{x}(n)$$

The nonlinearity $\Phi(\cdot)$ after the tap–delay line is typically the so-called sigmoid, a saturation-type nonlinearity like that on the previous slide.

$$\begin{aligned} e(n) &= d(n) - \Phi\left(\mathbf{w}^T(n)\mathbf{x}(n)\right) = d(n) - \Phi\big(net(n)\big) \\ \mathbf{w}(n+1) &= \mathbf{w}(n) - \mu \nabla_{\mathbf{w}(n)} J(n) \end{aligned}$$

where $e(n)$ is the instantaneous error at the output of the neuron, $d(n)$ is some teaching (desired) signal, $\mathbf{w}(n) = [w_1(n), \ldots, w_p(n)]^T$ is the weight vector, and $\mathbf{x}(n) = [x(n), \ldots, x(n-p+1)]^T$ is the input vector.

Dynamical perceptron: Learning algorithm

Using the ideas from LMS, the cost function is given by

$$J(n) = \frac{1}{2}e^2(n)$$

The gradient $\nabla_{\mathbf{w}(n)} J(n)$ is calculated from

$$\frac{\partial J(n)}{\partial \mathbf{w}(n)} = \frac{1}{2}\frac{\partial e^2(n)}{\partial e(n)}\frac{\partial e(n)}{\partial y(n)}\frac{\partial y(n)}{\partial net(n)}\frac{\partial net(n)}{\partial \mathbf{w}(n)} = -\,e(n)\,\Phi'(n)\mathbf{x}(n)$$

where $\Phi'(n) = \Phi'\big(net(n)\big) = \Phi'\big(\mathbf{w}^T(n)\mathbf{x}(n)\big)$ denotes the first derivative of the nonlinear activation function $\Phi(\cdot)$.

The weight update equation for the dynamical perceptron finally becomes

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\Phi'(n)e(n)\mathbf{x}(n)$$

∘ This filter is BIBO (bounded input bounded output) stable, as the output range is limited due to the saturation type of nonlinearity Φ.

☞ For large inputs (outliers) due to the saturation type of the nonlinearity, Φ, for large *net inputs* $\rightsquigarrow \Phi' \approx 0$, and the above weight update $\Delta\mathbf{w}(n) \approx \mathbf{0}$.

Notes

◦

Appendix 1: Wiener filter vs. Yule–Walker equations

we start from $x(n) = a_1(n)x(n-1) + \cdots + a_p(n)x(n-p) + q(n)$, q=white

The estimate: $y(n) = E\{x(n)\} = a_1(n)x(n-1) + \cdots + a_p(n)x(n-p)$

Teaching signal: $d(n)$, **Output error:** $e(n) = d(n) - y(n)$

1) Yule–Walker solution

Fixed coefficients $\mathbf{a}$ & $x(n) = y(n)$

Autoregressive modelling

$$\begin{aligned}
r_{xx}(1) &= a_1 r_{xx}(0) + \cdots + a_p r_{xx}(p-1) \\
r_{xx}(2) &= a_1 r_{xx}(1) + \cdots + a_p r_{xx}(p-2) \\
\vdots &= \vdots \\
r_{xx}(p) &= a_1 r_{xx}(p-1) + \cdots + a_p r_{xx}(0) \\
\ldots & \quad \ldots \\
\mathbf{r}_{xx} &= \mathbf{R}_{xx}\mathbf{a}
\end{aligned}$$

Solution: $\mathbf{a} = \mathbf{R}_{xx}^{-1}\mathbf{r}_{xx}$

2) Wiener–Hopf solution

Fixed optimal coeff. $\mathbf{w}_o = \mathbf{a}_{opt}$

$$J = E\{\frac{1}{2}e^2(n)\} = \sigma_d^2 - 2\mathbf{w}^T\mathbf{r}_{dx} + \mathbf{w}^T\mathbf{R}\mathbf{w}$$

is quadratic in $\mathbf{w}$ and for a full rank $\mathbf{R}$, it has **one unique minimum.**

Now:

$$\frac{\partial J}{\partial \mathbf{w}} = -\mathbf{r}_{dx} + \mathbf{R}\cdot\mathbf{w} = \mathbf{0}$$

Solution: $\mathbf{w}_0 = \mathbf{R}^{-1}\mathbf{r}_{dx}$

Appendix 2: LMS algorithm, scalar derivation

unlike the block-based steepest desc., LMS operates in a real-time online fashion

The LMS is based on the use of **instantaneous** estimates of the autocorrelation function $r_x(j,k)$ and the crosscorrelation function $r_{dx}(k)$

$$\hat{r}_x(j,k;n) = x_j(n)x_k(n) \qquad \hat{r}_{dx}(k;n) = x_k(n)d(n) \qquad J(n) = \frac{1}{2}e^2(n)$$

Substituting these into the method of steepest descent in (**) we have

$$\begin{aligned} w_k(n+1) &= w_k(n) + \mu\Big[x_k(n)d(n) - \sum_{j=1}^{p} w_j(n)x_j(n)x_k(n)\Big] \\ &= w_k(n) + \mu\Big[d(n) - \sum_{j=1}^{p} w_j(n)x_j(n)\Big]x_k(n) \\ &= w_k(n) + \mu[d(n) - y(n)]x_k(n) = w_k(n) + \mu e(n)x_k(n), \;\; \mathrm{k} = 1, \ldots, \mathrm{p} \end{aligned}$$

or, the LMS in the vector form: $\quad \mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n)$

Because of the 'instantaneous statistics' used in LMS derivation, the weights follow a "zig-zag" trajectory along the error surface, converging at the optimum solution $\mathbf{w}_0$, if μ is chosen properly.

Appendix 3: Improving the convergence and stability of LMS. The Normalised Least Mean Square (NLMS) alg.

Uses an adaptive step size by normalising μ by the signal power in the filter memory, that is

$$\text{from fixed } \mu \quad \rightsquigarrow \quad \text{data adaptive } \mu(n) = \frac{\mu}{\mathbf{x}^T(n)\mathbf{x}(n)} = \frac{\mu}{\| \mathbf{x}(n) \|_2^2}$$

Can be derived from the Taylor Series Expansion of the output error

$$e(n+1) = e(n) + \sum_{k=1}^{p} \frac{\partial e(n)}{\partial w_k(n)} \Delta w_k(n) + \underbrace{\text{higher order terms}}_{=0, \; since \; the \; filter \; is \; linear}$$

Since $\partial e(n)/\partial w_k(n) = -x_k(n)$ and $\Delta w_k(n) = \mu e(n) x_k(n)$, we have

$$e(n+1) = e(n)\Big[1 - \mu \sum_{k=1}^{p} x_k^2(n)\Big] = \big[1 - \mu \| \mathbf{x}(n) \|_2^2\big] \quad \text{as } \Big(\sum_{k=1}^{p} x_k^2 = \| \mathbf{x} \|_2^2 \Big)$$

To minimise the error, set $e(n+1) = 0$, to arrive at the NLMS step size:

$$\mu = \frac{1}{\| \mathbf{x}(n) \|_2^2} \qquad \text{however, in practice we use} \qquad \mu(n) = \frac{\mu}{\| \mathbf{x}(n) \|_2^2 + \varepsilon}$$

where $0 < \mu < 2$, $\mu(n)$ is time-varying, and ε is a small "regularisation" constant, added to avoid division by 0 for small values of input, $\| \mathbf{x} \| \to 0$

Appendix 4: Derivation of the formula for the convergence in the mean on Slide 30

○ For the weights to converge in the mean (unbiased condition), we need

$$E\{\mathbf{w}(n)\} = \mathbf{w}_o \quad \text{as } n \to \infty \qquad \Leftrightarrow \qquad E\{\mathbf{w}(n+1)\} = E\{\mathbf{w}(n)\} \quad \text{as } n \to \infty$$

○ The LMS update is given by: $\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n)$, then

$$\begin{aligned} E\{\mathbf{w}(n+1)\} &= E\Big\{\mathbf{w}(n) + \mu\big[d(n) - \mathbf{x}^T(n)\mathbf{w}(n)\big]\mathbf{x}(n)\Big\} \\ &\quad \text{since } \ d(n) - \mathbf{x}^T(n)\mathbf{w}(n) \ \text{ is a scalar, we can write} \\ &= E\Big\{\mathbf{w}(n) + \mu\mathbf{x}(n)\big[d(n) - \mathbf{x}^T(n)\mathbf{w}(n)\big]\Big\} \\ &= \Big[\mathbf{I} - \mu\underbrace{E\{\mathbf{x}(n)\mathbf{x}^T(n)\}}_{\mathbf{R}_{xx}}\Big]\mathbf{w}(n) + \mu\underbrace{E\{\mathbf{x}(n)d(n)\}}_{\mathbf{r}_{dx}} \\ &= \big[\mathbf{I} - \mu\mathbf{R}_{xx}\big]\mathbf{w}(n) + \mu\mathbf{r}_{dx} = \big[\mathbf{I} - \mu\mathbf{R}_{xx}\big]^n\mathbf{w}(0) + \mu\mathbf{r}_{dx} \end{aligned}$$

↬ the convergence is governed by the **homogeneous** part $\big[\mathbf{I} - \mu\mathbf{R}_{xx}\big]$

⇝ the filter will converge **in the mean** if $|\mathbf{I} - \mu\mathbf{R}_{xx}| < 1$

Appendix 5: Sign algorithms

Simplified LMS, derived based on $sign[e] = |e|/e$ and $\nabla|e| = sign[e]$

Good for hardware implementation and high speed applications.

- **The Sign Algorithm** (The cost function is $J(n) = |e(n)|$)

  Replace $e(n)$ by its sign to obtain

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\, sign[e(n)]\, \mathbf{x}(n)$$

- **The Signed Regressor Algorithm**

  Replace $\mathbf{x}(n)$ by $sign[\mathbf{x}(n)]$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\, e(n)\, sign[\mathbf{x}(n)]$$

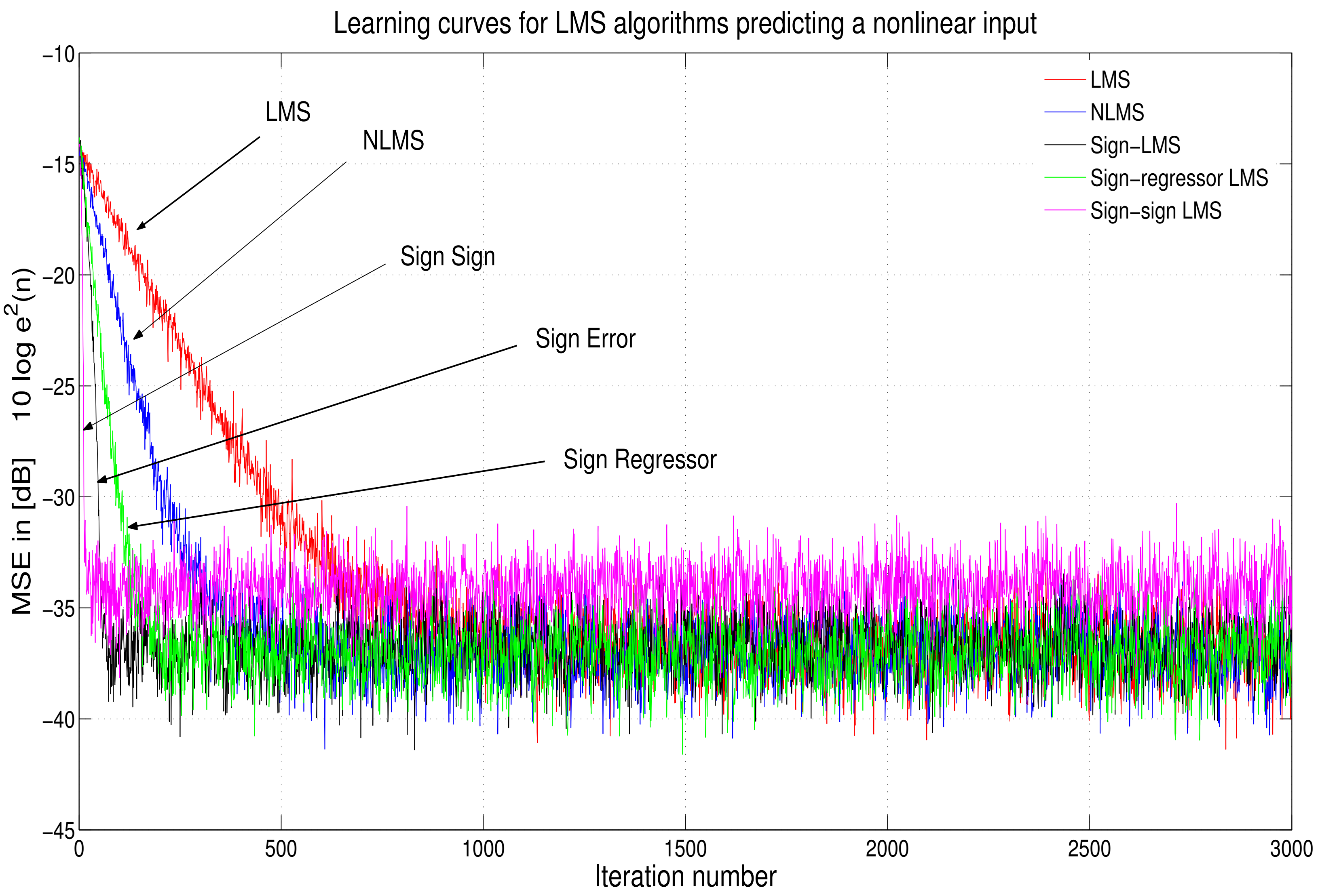
  Performs much better than the sign algorithm.

- **The Sign-Sign Algorithm**

  Combines the above two algorithms

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\, sign[e(n)]\, sign[\mathbf{x}(n)]$$

Appendix 5a): Performance of sign algorithms



Notes

○

Notes

○

Notes

○