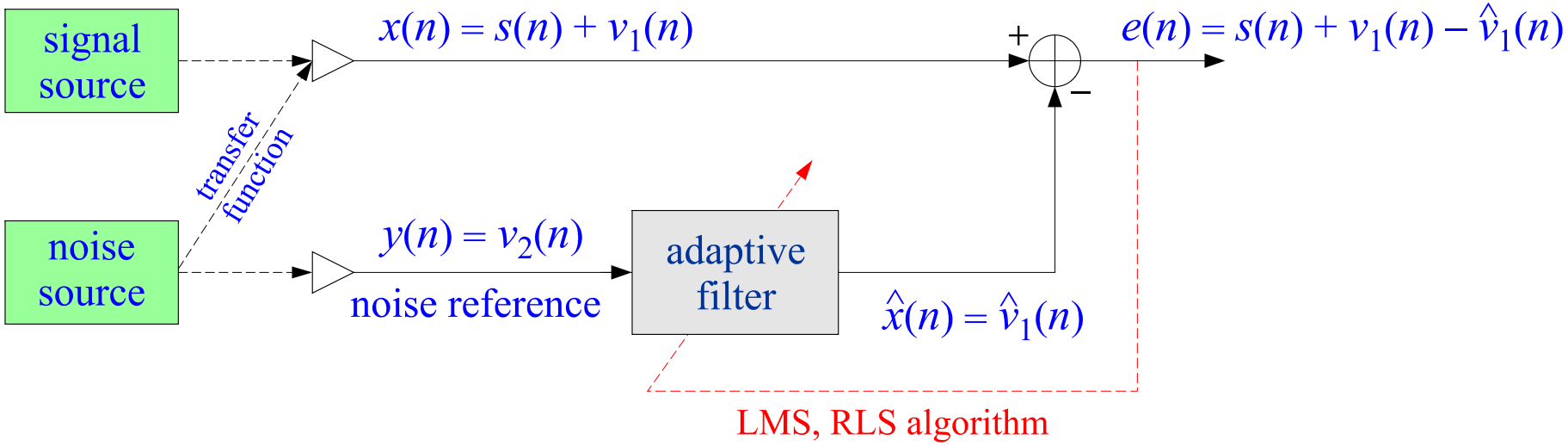
DSA – April 5 & April 12, 2021

Topics: Adaptive filtering and prediction, correlation canceling and optimum estimation, decorrelated bases, correlation canceler loop, LMS algorithm, gradient descent, learning speed, eigenvalue spread, accelerated LMS, Newton's iterative method, RLS algorithm, adaptive filtering applications, adaptive linear prediction.

Neural networks, activation functions, LMS and backpropagation, NN examples, XOR problem, NN prediction of sunspot and airline data.



Optimum Estimation – Wiener Filters

Correlation Canceling and Optimum Estimation

The concept of correlation canceling plays a central role in the development of many optimum signal processing algorithms, because a correlation canceler is also the optimum linear processor for estimating one signal from another.

Consider two zero-mean random vectors $\mathbf{x}$ and $\mathbf{y}$ of dimensions N and M, respectively. If $\mathbf{x}$ and $\mathbf{y}$ are correlated with each other in the sense that $R_{xy} = E[\mathbf{x}\mathbf{y}^T] \neq 0$, then we may remove such correlations by means of a linear transformation of the form

$$\mathbf{e} = \mathbf{x} - H\mathbf{y}$$

where the $N \times M$ matrix H must be suitably chosen such that the new pair of vectors $\mathbf{e}, \mathbf{y}$ are no longer correlated with each other, that is, we require

$$R_{ey} = E[\mathbf{e}\mathbf{y}^T] = 0$$

Using this condition, we obtain

$$R_{ey} = E[\mathbf{e}\mathbf{y}^T] = E\big[(\mathbf{x} - H\mathbf{y})\mathbf{y}^T\big] = E[\mathbf{x}\mathbf{y}^T] - HE[\mathbf{y}\mathbf{y}^T] = R_{xy} - HR_{yy}$$

Then, the condition $R_{ey} = 0$ immediately implies that

$$\boxed{H = R_{xy}R_{yy}^{-1} = E[\mathbf{x}\mathbf{y}^T]E[\mathbf{y}\mathbf{y}^T]^{-1}}$$

Using $R_{ey} = 0$, the covariance matrix of the resulting vector $\mathbf{e}$ is easily found to be

$$\begin{aligned} R_{ee} &= E[\mathbf{e}\mathbf{e}^T] = E\big[\mathbf{e}(\mathbf{x}^T - \mathbf{y}^T H)\big] \\ &= R_{ex} - R_{ey}H^T = R_{ex} = E\big[(\mathbf{x} - H\mathbf{y})\mathbf{x}^T\big], \quad \text{or,} \end{aligned}$$

$$R_{ee} = R_{xx} - HR_{yx} = R_{xx} - R_{xy}R_{yy}^{-1}R_{yx}$$

The vector,

$$\boxed{\hat{\mathbf{x}} = H\mathbf{y} = R_{xy}R_{yy}^{-1}\mathbf{y} = E[\mathbf{x}\mathbf{y}^T]E[\mathbf{y}\mathbf{y}^T]^{-1}\mathbf{y}}$$

obtained by linearly processing the vector $\mathbf{y}$ by the matrix H is called the *linear regression*, or *orthogonal projection*, of $\mathbf{x}$ on the vector $\mathbf{y}$. In a sense to be made precise later, $\hat{\mathbf{x}}$ also represents the best "copy," or *estimate*, of $\mathbf{x}$ that can be made on the basis of the vector $\mathbf{y}$. Thus, $\mathbf{e} = \mathbf{x} - H\mathbf{y} = \mathbf{x} - \hat{\mathbf{x}}$ may be thought of as the *estimation error*.

Actually, it is better to think of $\hat{\mathbf{x}} = H\mathbf{y}$ not as an estimate of $\mathbf{x}$ but rather as an estimate of *that part* of $\mathbf{x}$ which is correlated with $\mathbf{y}$. Indeed, suppose that $\mathbf{x}$ consists of two parts

$$\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2$$

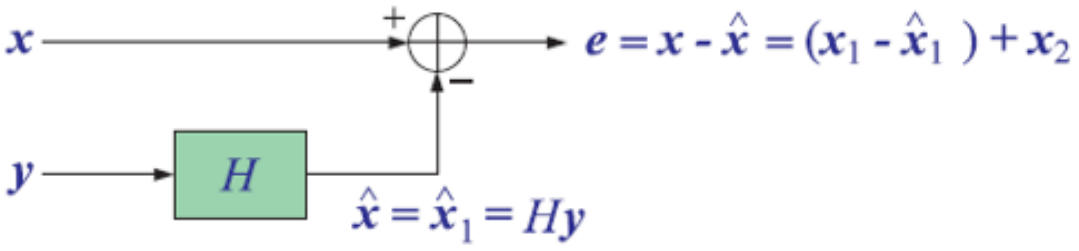
such that $\mathbf{x}_1$ is correlated with $\mathbf{y}$, but $\mathbf{x}_2$ is not, that is, $R_{x2y} = E[\mathbf{x}_2\mathbf{y}^T] = 0$. Then,

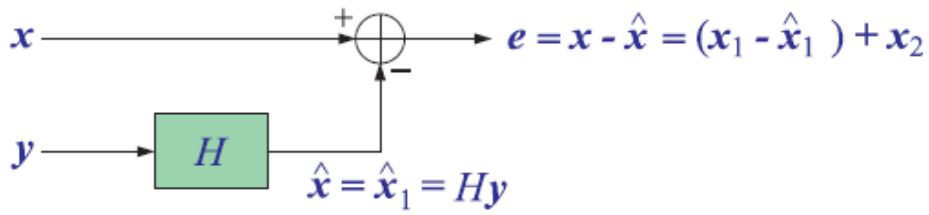
$$R_{xy} = E[\mathbf{x}\mathbf{y}^T] = E\big[(\mathbf{x}_1 + \mathbf{x}_2)\mathbf{y}^T\big] = R_{x1y} + R_{x2y} = R_{x1y}$$

and therefore,

$$\hat{\mathbf{x}} = R_{xy}R_{yy}^{-1}\mathbf{y} = R_{x1y}R_{yy}^{-1}\mathbf{y} = \hat{\mathbf{x}}_1$$

The vector $\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}} = \mathbf{x}_1 + \mathbf{x}_2 - \hat{\mathbf{x}}_1 = (\mathbf{x}_1 - \hat{\mathbf{x}}_1) + \mathbf{x}_2$ consists of the estimation error $(\mathbf{x}_1 - \hat{\mathbf{x}}_1)$ of the $\mathbf{x}_1$-part plus the $\mathbf{x}_2$-part. Both of these terms are separately uncorrelated from $\mathbf{y}$. These operations are summarized in block diagram form below.





The most important feature of this arrangement is the *correlation cancellation* property which may be summarized as follows: If $\mathbf{x}$ has a part $\mathbf{x}_1$ which is correlated with $\mathbf{y}$, then this part will tend to be canceled as much as possible from the output $\mathbf{e}$. The linear processor H accomplishes this by converting $\mathbf{y}$ into the *best possible copy* $\hat{\mathbf{x}}_1$ of $\mathbf{x}_1$ and then proceeds to cancel it from the output. The output vector $\mathbf{e}$ is no longer correlated with $\mathbf{y}$. The part $\mathbf{x}_2$ of $\mathbf{x}$ which is uncorrelated with $\mathbf{y}$ remains entirely unaffected. It cannot be estimated in terms of $\mathbf{y}$.

The correlation canceler may also be thought of as an *optimal signal separator*. Indeed, the output of the processor H is essentially the $\mathbf{x}_1$ component of $\mathbf{x}$, whereas the output $\mathbf{e}$ is essentially the $\mathbf{x}_2$ component. The separation of $\mathbf{x}$ into $\mathbf{x}_1$ and $\mathbf{x}_2$ is optimal in the sense that the $\mathbf{x}_1$ component of $\mathbf{x}$ is removed as much as possible from $\mathbf{e}$.

Next, we discuss the *best linear estimator property* of the correlation canceler. The choice $H = R_{xy}R_{yy}^{-1}$, which guarantees correlation cancellation, is also the choice that gives the *best estimate* of $\mathbf{x}$ as a *linear* function of $\mathbf{y}$ in the form $\hat{\mathbf{x}} = H\mathbf{y}$. It is the best estimate in the sense that it produces the lowest *mean-square* estimation error. To see this, express the covariance matrix of the estimation error in terms of H, as follows:

$$\begin{aligned} R_{ee} &= E[\mathbf{e}\mathbf{e}^T] = E\big[(\mathbf{x} - H\mathbf{y})(\mathbf{x}^T - \mathbf{y}^T H^T)\big] \\ &= E[\mathbf{x}\mathbf{x}^T] - HE[\mathbf{y}\mathbf{x}^T] - E]\mathbf{x}\mathbf{y}^T]H^T + HE[\mathbf{y}\mathbf{y}^T]H^T \\ &= R_{xx} - HR_{yx} - R_{xy}H^T + HR_{yy}H^T \end{aligned}$$

Minimizing this expression with respect to H yields the optimum choice:

$$H_{\text{opt}} = R_{xy}R_{yy}^{-1}$$

$$\hat{\mathbf{x}} = H_{\text{opt}}\mathbf{y} = R_{xy}R_{yy}^{-1}\mathbf{y}$$

with the minimum value for R_{ee} given by:

$$R_{ee}^{\text{min}} = R_{xx} - R_{xy}R_{yy}^{-1}R_{yx} = E[\mathbf{x}\mathbf{y}^T]\,E[\mathbf{y}\mathbf{y}^T]^{-1}\mathbf{y}$$

Any other value will result in a larger value for R_{ee}. An alternative way to see this is to consider a deviation ΔH of H from its optimal value, that is, replace H by

$$H = H_{\text{opt}} + \Delta H = R_{xy} R_{yy}^{-1} + \Delta H$$

Then R_{ee} may be expressed in terms of ΔH as follows:

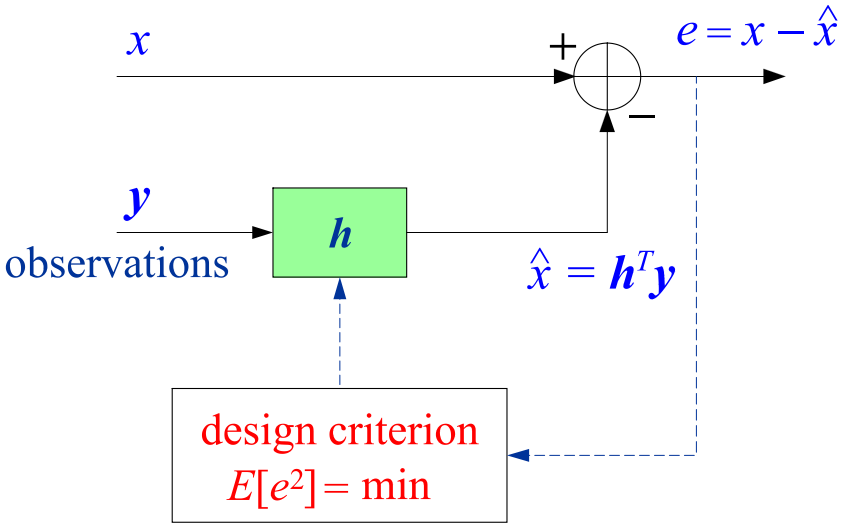
$$R_{ee} = R_{ee}^{\min} + (\Delta H)\, R_{yy}\, (\Delta H)^T$$

Since R_{yy} is positive definite, the second term always represents a non-negative contribution above the minimum value $R_{ee}^{\min}$, so that $\left(R_{ee} - R_{ee}^{\min}\right)$ is positive semi-definite.

In summary, there are three useful ways to think of the correlation canceler:

1. Optimal estimator of **x** from **y**.
2. Optimal canceler of that part of **x** which is correlated with **y**.
3. Optimal signal separator

The point of view is determined by the application. The first view is typified by Kalman filtering, channel equalization, and linear prediction applications. The second view is taken in echo canceling, noise canceling, and sidelobe canceling applications. The third view is useful in the adaptive line enhancer, which is a method of adaptively separating a signal into its broadband and narrowband components. All of these applications are considered later on.



Special Scalar Case

The above results, apply equally well to the case when $\mathbf{x}$ is a single scalar random variable, x, to be estimated from the vector of observations $\mathbf{y}$. In this case, we define

$$\mathbf{h}^T = H = E[x\mathbf{y}^T]\,E[\mathbf{y}\mathbf{y}^T]^{-1} = \text{row vector}$$

$$\mathbf{h} = E[\mathbf{y}\mathbf{y}^T]^{-1}\,E[x\mathbf{y}] = \text{column vector}$$

$$\mathbf{h} = R^{-1}\mathbf{r} = \text{optimum weights, Wiener solution}$$

$$R = E[\mathbf{y}\mathbf{y}^T] = \text{autocorrelation matrix}$$

$$\mathbf{r} = E[x\mathbf{y}] = \text{cross-correlation vector}$$

and obtain the optimum estimate and estimation error,

$$\boxed{\begin{aligned} \hat{x} &= \mathbf{h}^T\mathbf{y} = E[x\mathbf{y}^T]\,E[\mathbf{y}\mathbf{y}^T]^{-1}\mathbf{y} \\ e &= x - \hat{x} = x - \mathbf{h}^T\mathbf{y} \end{aligned}}$$

and the orthogonality and normal equations,

$$E[e\mathbf{y}] = 0 \quad \text{(orthogonality equations)}$$

$$E[\mathbf{y}\mathbf{y}^T]\mathbf{h} = E[x\mathbf{y}]\,, \quad \text{or,} \quad R\mathbf{h} = \mathbf{r} \quad \text{(normal equations)}$$

Performance index,

$$\mathcal{E}(\mathbf{h}) = E[e^2] = E\left[(x - \mathbf{h}^T\mathbf{y})^2\right] = E\left[x^2 - 2x(\mathbf{h}^T\mathbf{y}) + (\mathbf{h}^T\mathbf{y})(\mathbf{h}^T\mathbf{y})\right]$$
$$= E\left[x^2 - 2\mathbf{h}^T(x\mathbf{y}) + (\mathbf{h}^T\mathbf{y})(\mathbf{y}^T\mathbf{h})\right]$$
$$= E[x^2] - 2\mathbf{h}^T\mathbf{r} + \mathbf{h}^T R\,\mathbf{h} = \min$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{h}} = 2E\left[e\,\frac{\partial e}{\partial \mathbf{h}}\right], \quad \frac{\partial e}{\partial \mathbf{h}} = \frac{\partial}{\partial \mathbf{h}}(x - \mathbf{h}^T\mathbf{y}) = -\mathbf{y}$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{h}} = -2E[e\mathbf{y}] = 2(R\mathbf{h} - \mathbf{r}) = 0 \quad \Rightarrow \quad R\mathbf{h} = \mathbf{r} \quad \Rightarrow \quad \mathbf{h} = R^{-1}\mathbf{r}$$

gradient descent algorithm

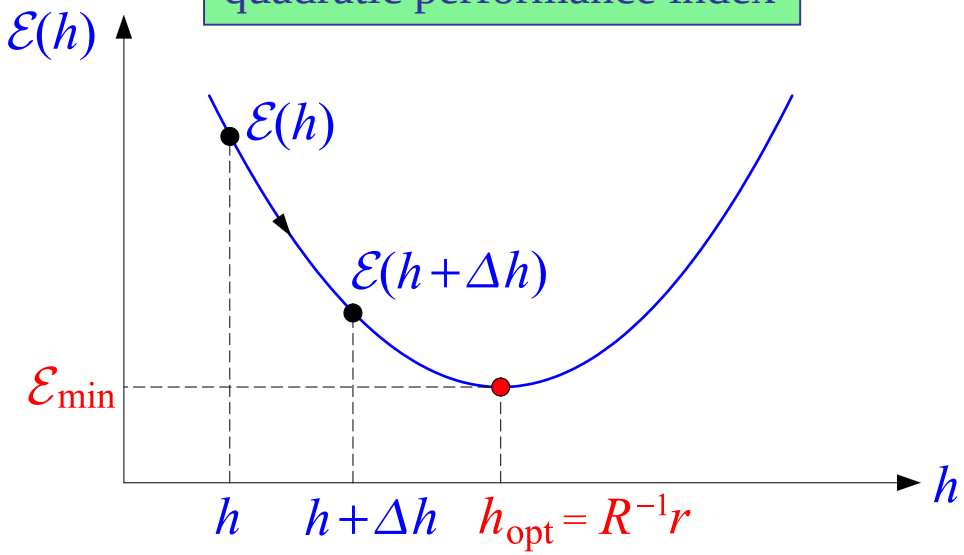
$$\Delta h = -\mu\,\frac{\partial \mathcal{E}(h)}{\partial h}$$

$$\mathcal{E}(h + \Delta h) \simeq \mathcal{E}(h) + \Delta h \cdot \frac{\partial \mathcal{E}(h)}{\partial h}$$

$$\mathcal{E}(h + \Delta h) = \mathcal{E}(h) - \mu \left|\frac{\partial \mathcal{E}(h)}{\partial h}\right|^2$$

$$\mathcal{E}(h + \Delta h) \le \mathcal{E}(h)$$

quadratic performance index



Notation change to $M+1$ observations and weights,

$$\mathbf{h} = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_M \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_M \end{bmatrix}$$

$$\hat{x} = \mathbf{h}^T\mathbf{y} = \begin{bmatrix} h_0, h_1, \ldots, h_M \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_M \end{bmatrix} = \sum_{i=0}^{M} h_i y_i$$

$$R = E[\mathbf{y}\mathbf{y}^T] = (M+1) \times (M+1) \text{ matrix}$$

$$\mathbf{r} = E[x\mathbf{y}] = (M+1) \times 1 \text{ column vector}$$

with matrix elements,

$$R_{ij} = E[y_i y_j], \quad i, j = 0, 1, \ldots, M$$

$$r_i = E[x y_i], \quad i = 0, 1, \ldots, M$$

For time-series applications and spatial arrays, the weights and observation vectors are defined at times instant n, as follows,

$$\mathbf{h} = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_M \end{bmatrix}$$

delay line

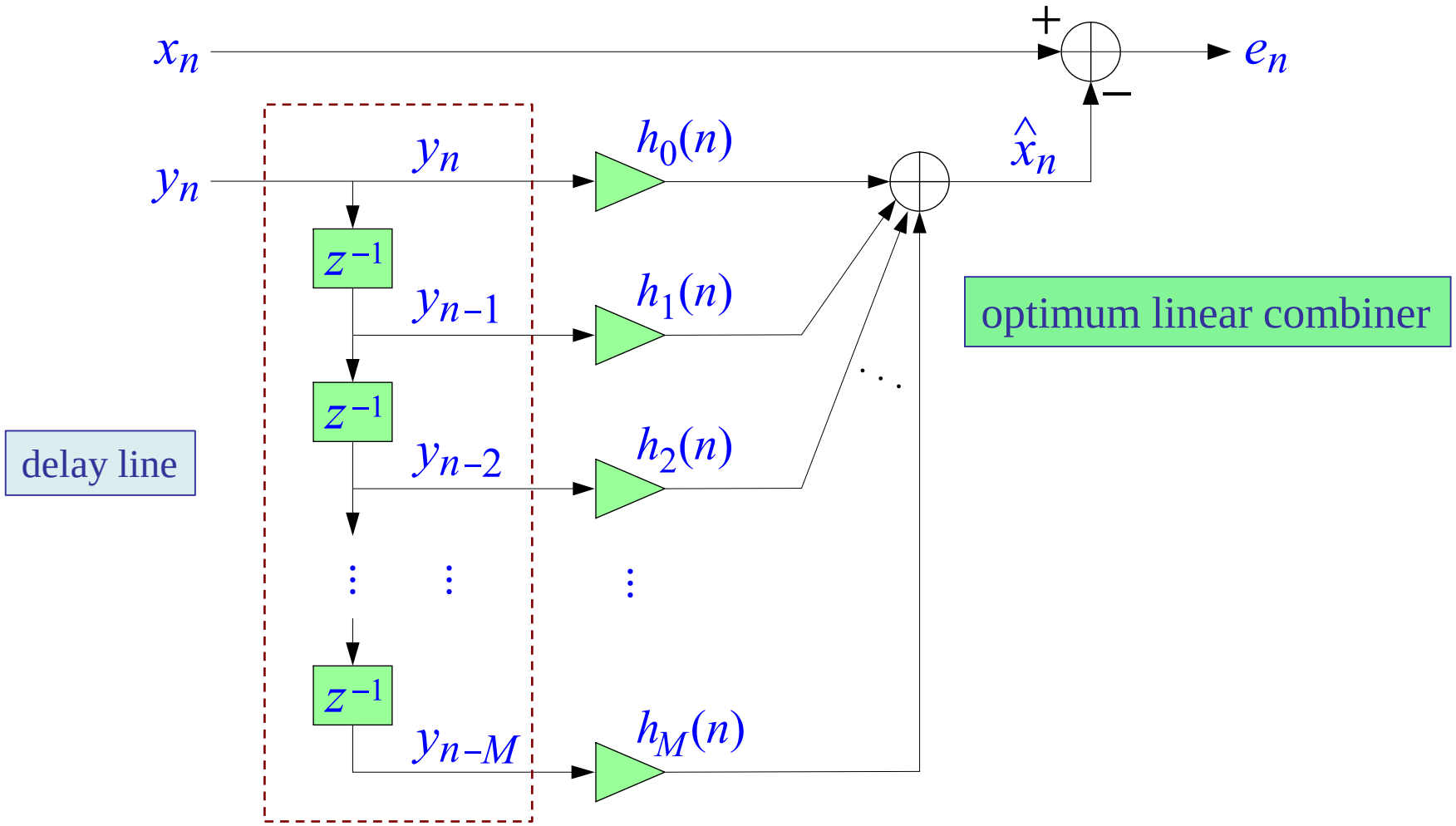
$$\mathbf{y}(n) = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-M} \end{bmatrix} = \text{time series}, \quad \mathbf{y}(n) = \begin{bmatrix} y_0(n) \\ y_1(n) \\ \vdots \\ y_M(n) \end{bmatrix} = \text{spatial arrays}$$

array elements

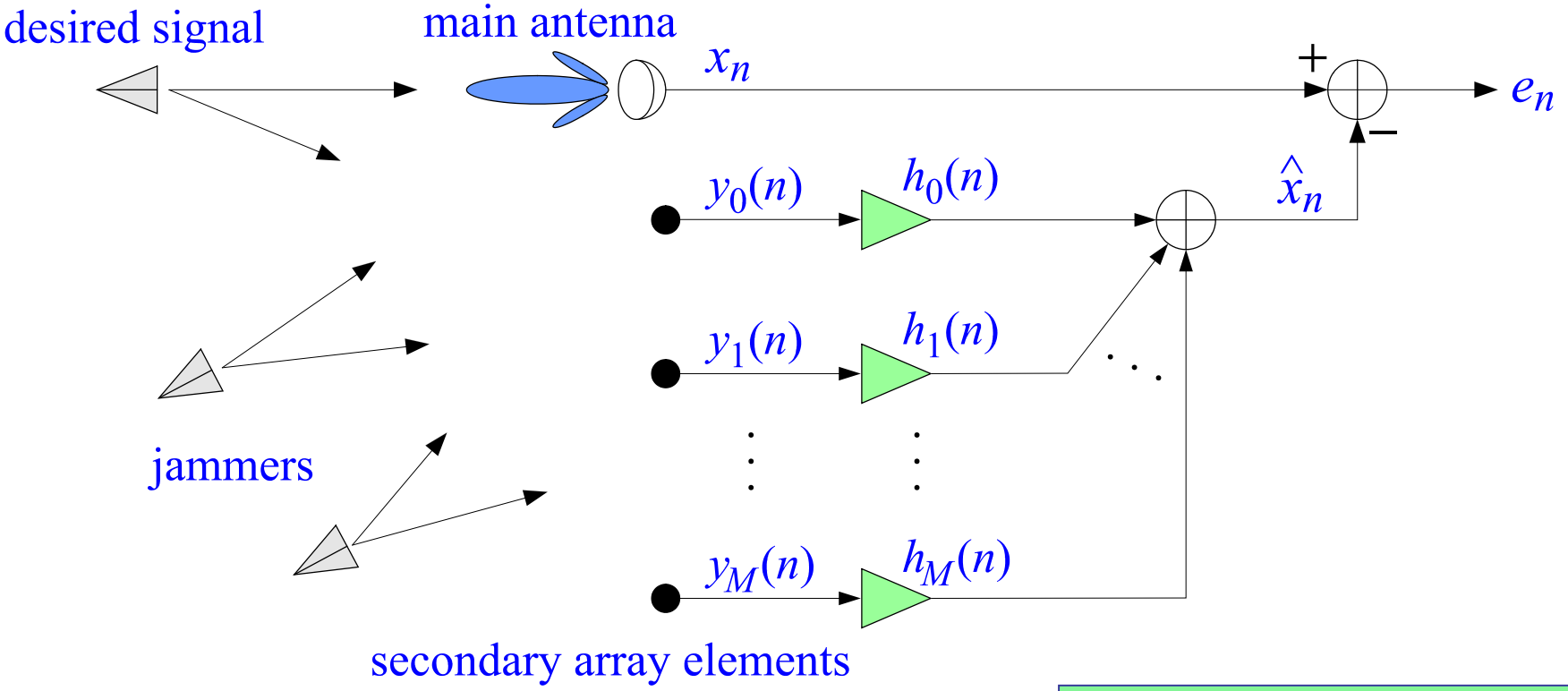
and the estimate at time n,

$$\hat{x}_n = \mathbf{h}^T \mathbf{y}(n)$$

time-series – adaptive Wiener filter



spatial arrays – adaptive antennas



Other Optimum Estimators

The estimate, $\hat{x} = \mathbf{h}^T\mathbf{y}$, is the optimum linear mean-square estimate of x, in the sense that it it minimizes the mean-square estimation error,

$$\mathcal{E} = E[e^2] = \min$$

under the assumption that $\hat{x}$ is a linear function of the observations.

There are other estimation criteria that result in estimators, $\hat{x}(\mathbf{y})$ that may be nonlinear functions of $\mathbf{y}$. Such criteria make use of the joint density $p(x, \mathbf{y})$ of the random variables x and $\mathbf{y}$, and the corresponding conditional densities.

$$p(x, \mathbf{y}) = \text{joint of density of } x \text{ and } \mathbf{y}$$

$$p(x|\mathbf{y}) = \text{conditional density of } x \text{ given } \mathbf{y}$$

$$p(\mathbf{y}|x) = \text{conditional density of } \mathbf{y} \text{ given } x$$

$$p(x) = \text{a priori density of } x$$

$$p(\mathbf{y}) = \text{joint of a priori density of } \mathbf{y}$$

From Bayes' rule, we have the relationships,

$$p(x, \mathbf{y}) = p(x|\mathbf{y})\, p(x) = p(\mathbf{y}|x)\, p(x)$$

The alternative estimators are:

1. Maximum a posteriori (MAP) estimate. Finds the x that maximizes the a posteriori conditional density,

$$p(x|\mathbf{y}) = \max$$

2. Maximum likelihood (ML) estimate. Finds the x that maximizes the conditional density,

$$p(\mathbf{y}|x) = \max$$

3. Unconstrained mean-square (MS) estimate: Finds the $\hat{x}(\mathbf{y})$ that minimizes the mean-square error without assuming linear dependence,

$$E[e^2] = E\big[\,\big(x - \hat{x}(\mathbf{y})\big)^2\,\big] = \min$$

Its solution is the conditional mean relative to the a posteriori conditional density, $p(x|\mathbf{y})$,

$$\hat{x}(\mathbf{y}) = E[x|\mathbf{y}] = \int_{-\infty}^{\infty} x p(x|\mathbf{y})\, dx$$

Ideally, the MAP estimate would be the best, in the sense that it finds the most likely x given that a particular realization of $\mathbf{y}$ has been observed. However, it is in general difficult to solve because we may not know the conditional density $p(x\mathbf{y})$. For the same reason, the MS estimate is also difficult to obtain.

The ML estimate is generally easier to obtain, but it is a bit questionable because, in maximizing $p(\mathbf{y}|x)$, it finds that x that renders the observed vector $\mathbf{y}$ the most likely to have been observed.

Theorem: For zero-mean, jointly gaussian, random variables $x, \mathbf{y}$, the MS, the MAP, and the linear MS estimates are the same.

Observation Bases

The linear mean-square estimate,

$$\hat{x} = \mathbf{h}^T \mathbf{y} = E[x\mathbf{y}^T] E[\mathbf{y}\mathbf{y}^T]^{-1} \mathbf{y}$$

is basis-independent, in the sense that if we define a new vector of observations $\mathbf{z}$ as an arbitrary linear combination of the original basis,

$$\mathbf{y} = B\mathbf{z} \quad \Rightarrow \quad \mathbf{z} = B^{-1}\mathbf{y}$$

where B is an arbitrary but invertible $(M+1) \times (M+1)$ matrix, then the optimum estimate remains invariant under such base change, that is,

$$\hat{x} = E[x\mathbf{y}^T] E[\mathbf{y}\mathbf{y}^T]^{-1} \mathbf{y} = E[x\mathbf{z}^T] E[\mathbf{z}\mathbf{z}^T]^{-1} \mathbf{z}$$

defining the weights with respect to the new basis,

$$\mathbf{g} = E[\mathbf{z}\mathbf{z}^T]^{-1} E[x\mathbf{z}]$$

we have,

$$\hat{x} = \mathbf{g}^T \mathbf{z} = \mathbf{h}^T \mathbf{y}$$

$$\mathbf{g} = B^T \mathbf{h}$$

Proof:

$$\mathbf{y} = B\mathbf{z}$$

$$E[\mathbf{y}\mathbf{y}^T] = E\left[(B\mathbf{z})(B\mathbf{z})^T\right] = B\,E[\mathbf{z}\mathbf{z}^T]\,B^T$$

$$E[\mathbf{y}\mathbf{y}^T]^{-1} = B^{-T}\,E[\mathbf{z}\mathbf{z}^T]^{-1}B^{-1}$$

$$E[x\mathbf{y}] = E\left[x(B\mathbf{z})\right] = B\,E[x\mathbf{z}]$$

$$E[x\mathbf{y}^T] = E[x\mathbf{z}^T]\,B^T$$

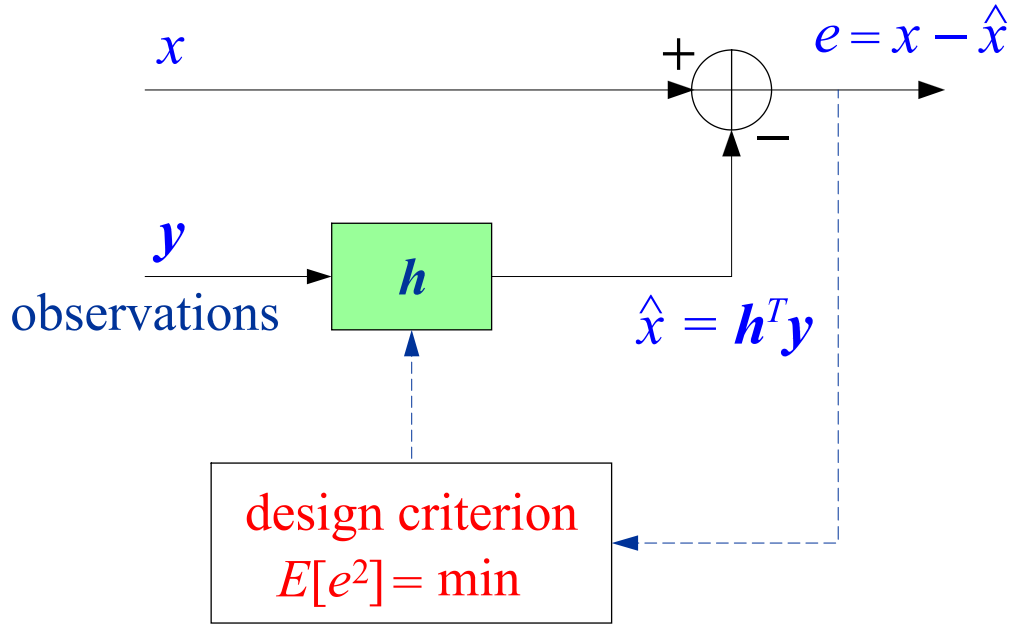
$$E[x\mathbf{y}^T]\,E[\mathbf{y}\mathbf{y}^T]^{-1}\mathbf{y} = E[x\mathbf{z}^T]\,B^T B^{-T}\,E[\mathbf{z}\mathbf{z}^T]^{-1}B^{-1}B\mathbf{z}$$

$$E[x\mathbf{y}^T]\,E[\mathbf{y}\mathbf{y}^T]^{-1}\mathbf{y} = E[x\mathbf{z}^T]\,E[\mathbf{z}\mathbf{z}^T]^{-1}\mathbf{z}$$

$$\mathbf{h} = E[\mathbf{y}\mathbf{y}^T]^{-1}E[x\mathbf{y}] = B^{-T}\,E[\mathbf{z}\mathbf{z}^T]^{-1}B^{-1}BE[x\mathbf{z}] = B^{-T}\mathbf{g}$$

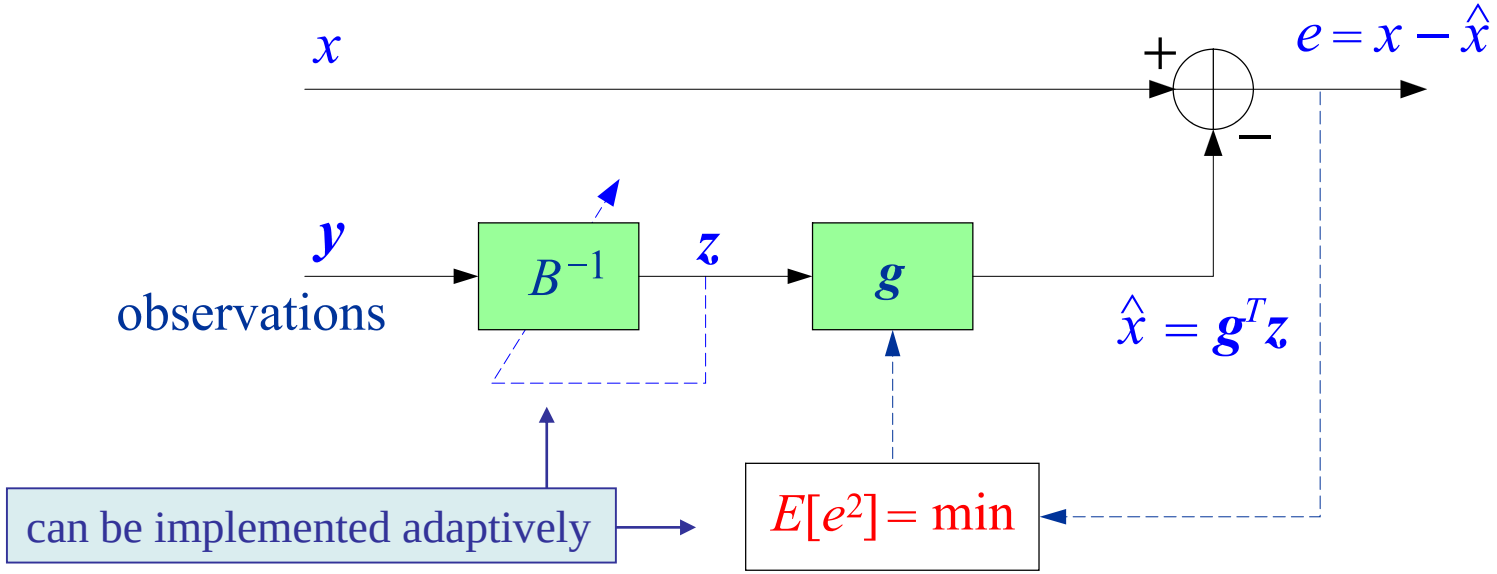
$$\mathbf{g} = B^T\mathbf{h}$$

realizations



$$\mathbf{y} = B\mathbf{z}$$

$$\mathbf{z} = B^{-1}\mathbf{y}$$



Decorrelated Bases

The best basis would have a diagonal autocorrelation matrix. Let us denote such basis by the $(M+1)$-dimensional column vector, $\boldsymbol{\epsilon}$, chosen such that,

$$\boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_M \end{bmatrix}, \quad D = E[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T] = \begin{bmatrix} E_0 & 0 & \cdots & 0 \\ 0 & E_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & E_M \end{bmatrix} = \text{diagonal}$$

$$E_i = E[\epsilon_i^2], \quad i = 0, 1, \ldots, M$$

$$D_{ij} = E[\epsilon_i \epsilon_j] = E_i \, \delta_{ij}, \quad i, j = 0, 1, \ldots, M$$

decorrelated components

$$\mathbf{y} = B\boldsymbol{\epsilon}$$

$$\hat{x} = E[x\mathbf{y}^T] E[\mathbf{y}\mathbf{y}^T]^{-1} \mathbf{y} = E[x\boldsymbol{\epsilon}^T] E[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T]^{-1} \boldsymbol{\epsilon}$$

diagonal matrix inversion

Thus, $R = E[\mathbf{y}\mathbf{y}^T]$, is factored with a positive-definite diagonal D,

$$R = B\,D\,B^T$$

There are two ways to achieve this:

1. Eigenvalue decomposition of R. Leads to Karhunen-Loeve transform and principal component analysis, which becomes practical via the SVD. In this case, the matrix B is the eigenvector matrix of R. See AOSP-Ch.15.
2. Gram-Schmidt orthogonalization of the random variables $\mathbf{y}$. Leads to lattice adaptive filters, which are computationally efficient and exhibit very fast convergence. In this case, B is a lower-triangular matrix and corresponds to the Cholesky factorization of R. See AOSP-Ch.1 & 16.

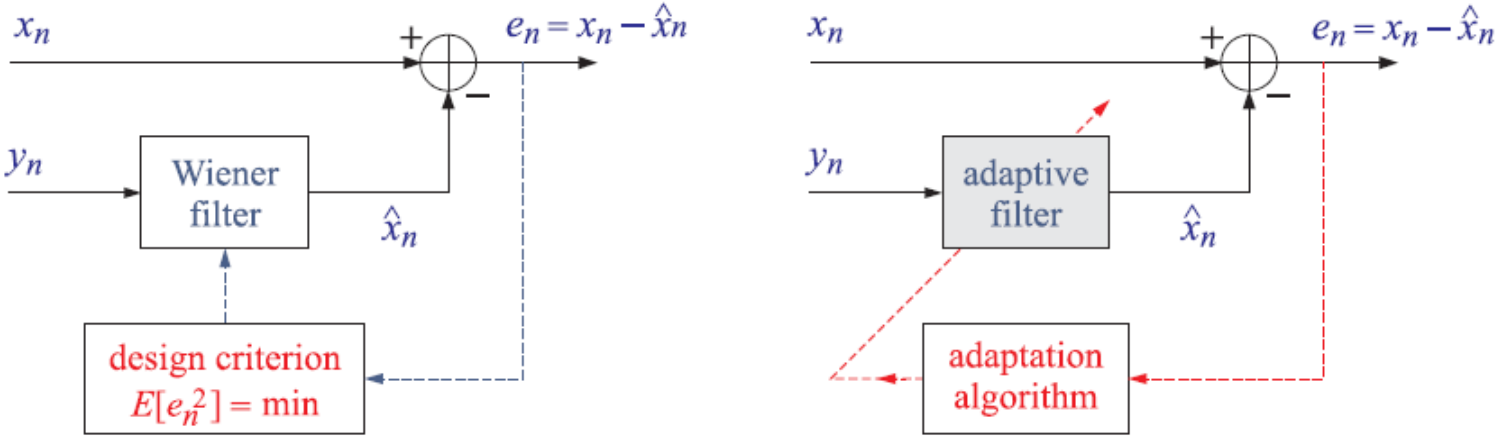
We may think of the basis transformation as a signal model for $\mathbf{y}$, being synthesized from a decorrelated random vector, $\boldsymbol{\epsilon}$, that is, with, $A = B^{-1}$,

$$\boldsymbol{\epsilon} \longrightarrow \boxed{B} \longrightarrow \mathbf{y} = B\boldsymbol{\epsilon} \qquad \text{(synthesis)}$$

$$\mathbf{y} \longrightarrow \boxed{A} \longrightarrow \boldsymbol{\epsilon} = A\mathbf{y} \qquad \text{(analysis)}$$

Adaptive Implementation of Wiener Filters

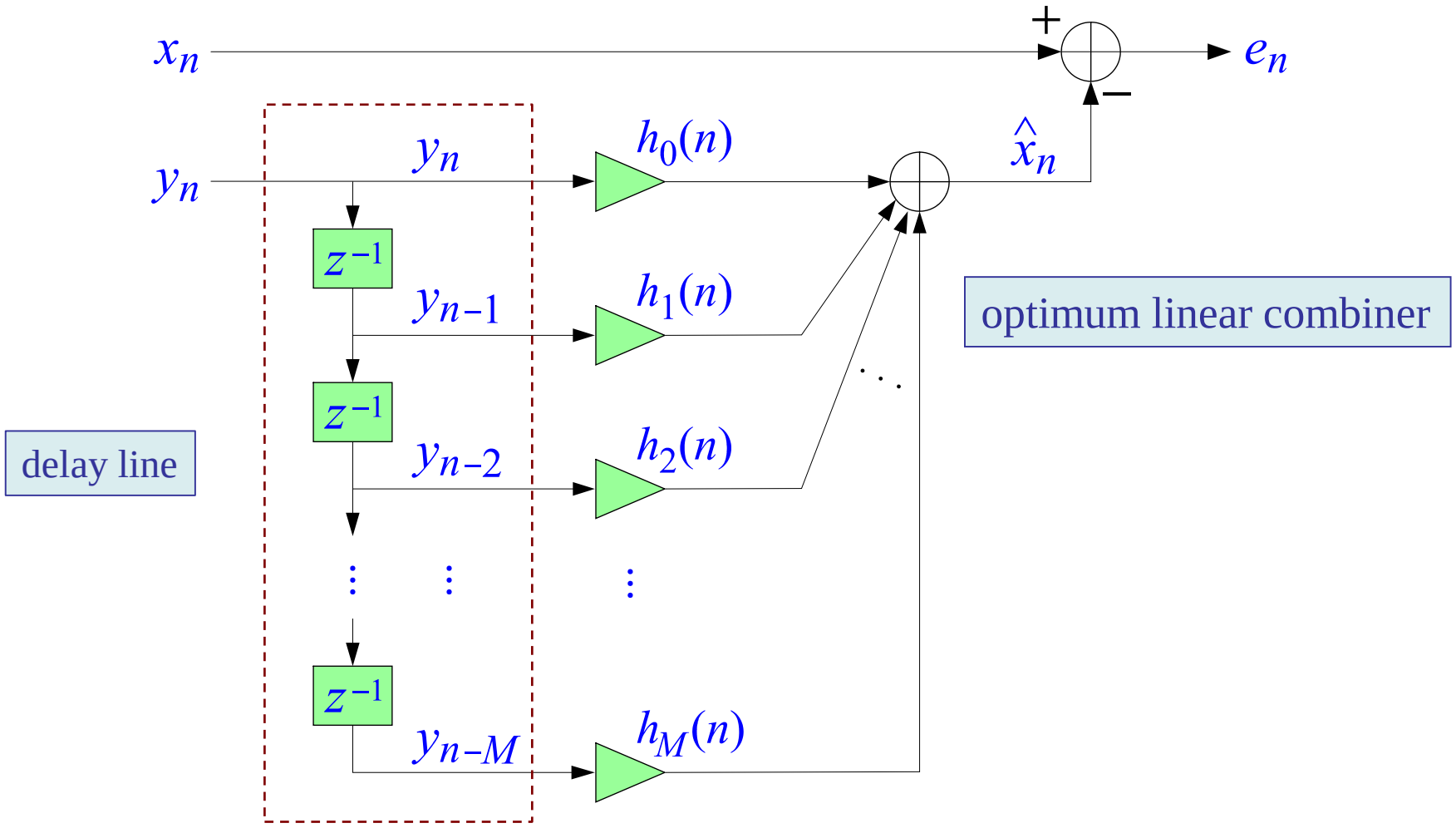
A Wiener filter and its adaptive implementation are shown below The filter processes the observation signal y_n to make an estimate $\hat{x}_n$ of the primary x_n, and the resulting error output e_n is fed back into an adaptation algorithm, such as the LMS or RLS, that changes the filter weights to be used at the next time instant.



where, for an order-M FIR Wiener filter, the estimate $\hat{x}_n$ is,

$$\hat{x}_n = \sum_{i=0}^{M} h_i y_{n-i} = [h_0, h_1, \ldots, h_M] \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-M} \end{bmatrix} = \mathbf{h}^T \mathbf{y}(n)$$

adaptive Wiener filter



There are three basic issues in any adaptive implementation:

1. The learning or convergence speed of the algorithm.
2. The computational complexity of the algorithm.
3. The numerical accuracy and stability of the algorithm.

The convergence speed is an important factor because it determines the maximum rate of change of the input non-stationarities that can be usefully tracked by the filter. The computational complexity refers to the number of operations required to update the filter from one time instant to the next. The table below shows how various adaptive algorithms fare under these requirements.

algorithm	speed	complexity	stability
LMS	slow	simple, $O(2M)$	stable
RLS	fast	complex, $O(M^2)$	stable
Fast RLS	fast	simple, $O(7M)$	unstable
Lattice	fast	simple, $O(16M)$	stable

Only adaptive lattice algorithms satisfy all three requirements.

For stationary, zero-mean, input signals x_n, y_n, the optimum Wiener filter coefficients $\mathbf{h}$ are given in terms of the stationary $(M+1)\times(M+1)$ autocorrelation matrix R of the data vector $\mathbf{y}(n)$ and the $(M+1)\times 1$ cross-correlation vector $\mathbf{r}$ between x_n and $\mathbf{y}(n)$, that is,

$$\begin{aligned} R &= E\big[\mathbf{y}(n)\mathbf{y}^T(n)\big] \\ \mathbf{r} &= E\big[x_n\mathbf{y}(n)\big] \end{aligned} \quad \Rightarrow \quad \mathbf{h} = R^{-1}\mathbf{r}$$

We note that R is an $(M+1)\times(M+1)$ symmetric and positive-definite matrix, and $\mathbf{r}$ is an $(M+1)\times 1$ column vector, with matrix elements that are independent of n because of the assumed stationarity,

$$\begin{aligned} R_{ij} &= E\big[y_{n-i}y_{n-j}\big]\,, \quad 0 \le i,j \le M \\ r_i &= E\big[x_n y_{n-i}\big]\,, \quad 0 \le i \le M \end{aligned}$$

The optimum solution is derived by minimizing the mean-square estimation error with respect to vector of filter weights $\mathbf{h}$,

$$\boxed{\mathcal{E} = E[e_n^2] = \min}\,, \quad e_n = x_n - \hat{x}_n = x_n - \mathbf{h}^T\mathbf{y}(n)$$

that is, setting the gradient with respect to $\mathbf{h}$ to zero,

$$\frac{\partial\mathcal{E}}{\partial\mathbf{h}} = -2E\big[e_n\mathbf{y}(n)\big] = 2\big(R\mathbf{h}-\mathbf{r}\big) = 0 \quad \Rightarrow \quad \mathbf{h} = R^{-1}\mathbf{r}$$

In the LMS adaptation algorithm, the weights $\mathbf{h}$ are replaced by time-varying ones, $\mathbf{h}(n)$, which are updated in time by the gradient-descent method, but using an instantaneous gradient obtained by dropping the expectation values in the theoretical gradient, that is, making the replacement,

$$\frac{\partial \mathcal{E}}{\partial \mathbf{h}} = -2E\big[e_n \mathbf{y}(n)\big] \quad \Rightarrow \quad \frac{\widehat{\partial \mathcal{E}}}{\partial \mathbf{h}} = -2e_n \mathbf{y}(n)$$

so that the weights to be used at the next time instant are calculated by,

$$\mathbf{h}(n+1) = \mathbf{h}(n) - \mu \frac{\widehat{\partial \mathcal{E}}}{\partial \mathbf{h}} = \mathbf{h}(n) + 2\mu\, e_n \mathbf{y}(n)$$

This leads to the following Widrow-Hoff LMS algorithm that combines the filtering and weight-adaptation parts,

for each time instant n,

given x_n, $\mathbf{y}(n)$, $\mathbf{h}(n)$, do,

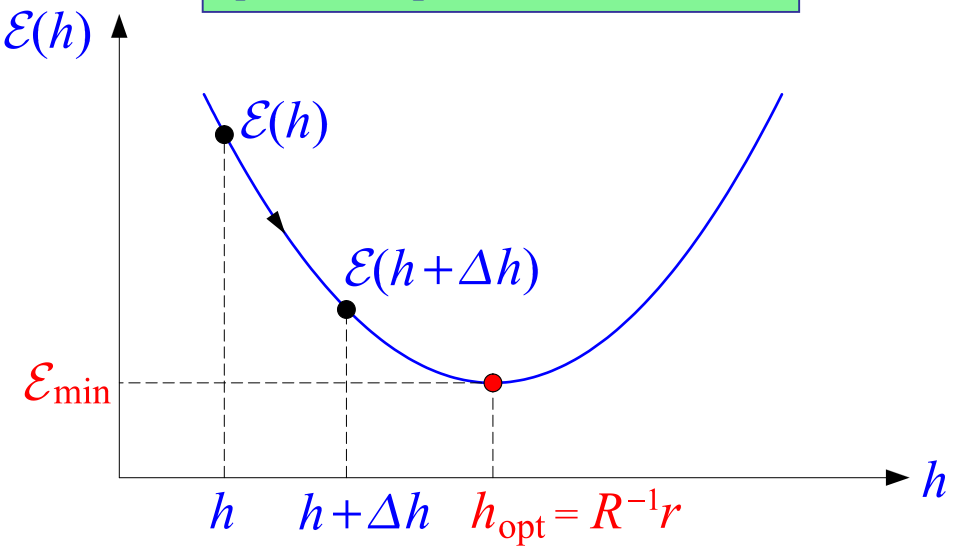
$$\hat{x}_n = \mathbf{h}^T(n)\mathbf{y}(n)$$

$$e_n = x_n - \hat{x}_n$$

$$\mathbf{h}(n+1) = \mathbf{h}(n) + 2\mu\, e_n \mathbf{y}(n)$$

(LMS algorithm)

quadratic performance index



gradient descent algorithm

$$\Delta h = -\mu \frac{\partial \mathcal{E}(h)}{\partial h}$$

$$\mathcal{E}(h+\Delta h) \simeq \mathcal{E}(h) + \Delta h \cdot \frac{\partial \mathcal{E}(h)}{\partial h}$$

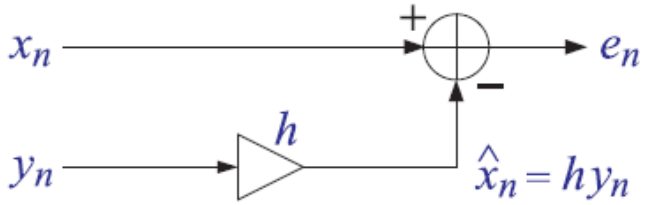
$$\mathcal{E}(h+\Delta h) = \mathcal{E}(h) - \mu \left| \frac{\partial \mathcal{E}(h)}{\partial h} \right|^2$$

$$\mathcal{E}(h+\Delta h) \leq \mathcal{E}(h)$$

the brilliant insight of the LMS algorithm was taking the gradient descent iteration index to be the time index *n*, so that optimization iterations are carried out at each time instant.

Correlation Canceler Loop (CCL)

To illustrate the basic principles behind adaptive filters and understand their convergence properties, consider the simplest possible filter, that is, a filter with only a scalar weight and scalar observation,



the weight h must be selected optimally so as to produce the optimum estimate of x_n :

$$\hat{x}_n = hy_n$$

The estimation error is expressed as

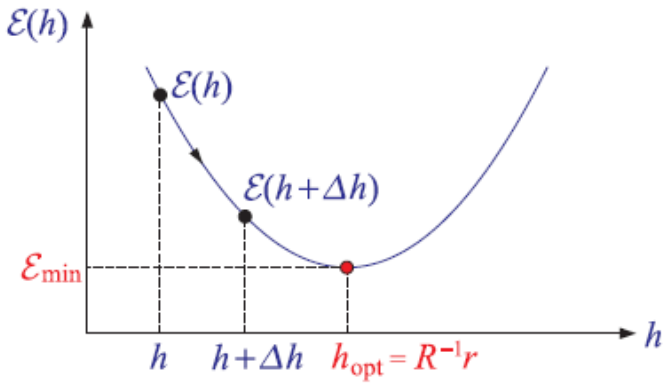
$$\mathcal{E} = E[e_n^2] = E\left[(x_n - hy_n)^2)\right] = E[x_n^2] - 2hE[x_n y_n] + E[y_n^2]h^2$$

$$\mathcal{E} = E[x_n^2] - 2hr + Rh^2$$

The minimization condition is,

$$\frac{\partial \mathcal{E}}{\partial h} = -2E[e_n y_n] = -2r + 2Rh = 0 \quad \Rightarrow \quad h_{\text{opt}} = R^{-1}r$$

The dependence of the error $\mathcal{E}$ on the filter parameter h is parabolic, with a global minimum occurring at the optimal value $h_{\text{opt}} = R^{-1}r$, as shown,



In the adaptive version, the filter parameter h is made *time-dependent*, $h(n)$, and is updated from one time instant to the next as follows

$$h(n+1) = h(n) + \Delta h(n) \tag{1}$$

where the correction term $\Delta h(n)$ must be chosen properly in order to ensure that eventually the time-varying weight $h(n)$ will converge to the optimum:

$$h(n) \to h_{\text{opt}} = R^{-1}r \quad \text{as} \quad n \to \infty$$

The filtering operation is still *linear*, but *time non-invariant*, so that at two successive time instants, the estimate must be computed as,

$$\hat{x}_n = h(n)\, y_n$$
$$\hat{x}_{n+1} = h(n+1)\, y_{n+1}$$

The simplest way to choose the correction term $\Delta h(n)$ is the gradient-descent, or steepest-descent, method. The change $h \to h + \Delta h$ must move the performance index closer to its minimum than before, that is, Δh must be such that

$$\mathcal{E}(h + \Delta h) \leq \mathcal{E}(h)$$

Therefore, if we always demand this inequality, the repetition of the procedure will lead to smaller and smaller values of $\mathcal{E}$ until the smallest value has been attained. Assuming that Δh is sufficiently small, we may expand to first order and obtain the condition

$$\mathcal{E}(h) + \Delta h \frac{\partial \mathcal{E}(h)}{\partial h} \leq \mathcal{E}(h) \quad \Rightarrow \quad \Delta h \frac{\partial \mathcal{E}(h)}{\partial h} \leq 0$$

If Δh is selected as the *negative gradient* of $\mathcal{E}$, then this inequality will be guaranteed,

$$\Delta h = -\mu \frac{\partial \mathcal{E}(h)}{\partial h}$$

indeed,

$$\mathcal{E}(h) + \Delta h \frac{\partial \mathcal{E}(h)}{\partial h} = \mathcal{E}(h) - \mu \left| \frac{\partial \mathcal{E}(h)}{\partial h} \right|^2 \leq \mathcal{E}(h)$$

The adaptation parameter μ must be small enough to justify keeping only the first-order terms in the above Taylor expansion. Thus, the updating of the filter weight would be,

$$h(n+1) = h(n) + \Delta h(n) = h(n) - \mu \frac{\partial \mathcal{E}\big(h(n)\big)}{\partial h}$$

Using the expression for the theoretical gradient,

$$\frac{\partial \mathcal{E}(h)}{\partial h} = -2r + 2Rh$$

we find,

$$h(n+1) = h(n) - \mu\big[-2r + 2Rh(n)\big]$$

$$h(n+1) = (1 - 2\mu R)h(n) + 2\mu r$$

This difference equation may be solved in closed form. For example, using z-transforms with any initial conditions $h(0)$, we find,

$$h(n) = h_{\text{opt}} + (1 - 2\mu R)^n (h(0) - h_{\text{opt}})\,, \quad \text{where} \quad h_{\text{opt}} = R^{-1} r$$

The coefficient $h(n)$ will converge to its optimal value h_{opt}, regardless of the starting value $h(0)$, provided μ is selected such that

$$|1 - 2\mu R| < 1 \quad \Rightarrow \quad -1 < 1 - 2\mu R < 1$$

or, since μ and R are positive, we obtain the convergence condition,

$$\boxed{0 < \mu < \frac{1}{R}}$$

To select μ, one must have some a priori knowledge of the magnitude of the input variance $R = E[y_n^2]$. Such choice for μ will guarantee convergence, but the speed of convergence is controlled by how close the number $1 - 2\mu R$ is to one.

The closer it is to unity, the slower the speed of convergence. As μ is selected closer to zero, the closer $1 - 2\mu R$ moves towards one, and thus the slower the convergence rate.

Thus, the adaptation parameter μ must be selected to be small enough to guarantee convergence but not too small to cause a very slow convergence. The special choice,

$$\mu = \frac{1}{2R} \quad \Rightarrow \quad 1 - 2\mu R = 0$$

will result in convergence in one time step, and is equivalent to Newton's method of solving the optimization equations, and eventually will be realized via the RLS algorithm.

LMS Adaptation Algorithm

From the practical point of view, the above implementation is still not computable since the adaptation of the weights requires a priori knowledge of the correlations R and r. In the Widrow-Hoff algorithm the above theoretical adaptation algorithm is replaced with one that is computable. The theoretical gradient,

$$h(n+1) = h(n) - \mu \frac{\partial \mathcal{E}\big(h(n)\big)}{\partial h}$$

is replaced by an instantaneous gradient by *ignoring* the expectation instructions, that is, the theoretical gradient

$$\frac{\partial \mathcal{E}\big(h(n)\big)}{\partial h} = -2E[e_n y_n] = -2r + 2Rh(n) = -2E[x_n y_n] + 2E[y_n^2]h(n)$$

is replaced by

$$\frac{\partial \mathcal{E}}{\partial h} = -2e_n y_n = -2\big(x_n - h(n)y_n\big)y_n = -2x_n y_n + 2y_n^2 h(n)$$

so that the weight-adjustment algorithm becomes

$$h(n+1) = h(n) + 2\mu e_n y_n$$

In summary, the required computations are done in the following order:

1. At time n, the filter weight $h(n)$ is available.
2. Compute the filter output, $\hat{x}_n = h(n)y_n$.
3. Compute the estimation error, $e_n = x_n - \hat{x}_n$.
4. Compute the next filter weight, $h(n+1) = h(n) + 2\mu e_n y_n$.
5. Go to next time instant $n \rightarrow n+1$.

The following remarks are in order:

1. The output error e_n is fed back and used to control the adaptation of the filter weight $h(n)$.
2. The filter tries to decorrelate the secondary signal from the output e_n. This, is easily seen as follows: If the weight $h(n)$ has more or less reached its optimum value, then $h(n+1) \simeq h(n)$, and the adaptation equation implies also approximately that $e_n y_n \simeq 0$.

3. Actually, the weight $h(n)$ never really reaches the theoretical limiting value $h_{\text{opt}} = R^{-1}r$. Instead, it stabilizes about this value, and continuously fluctuates about it. A measure of these fluctuations is the mean-square deviation of $h(n)$ from h_{opt}, that is, $E\big[\big(h(n) - h_{\text{opt}}\big)^2\big]$. Under some restrictive conditions, it has been shown by Widrow that,

$$E\big[\big(h(n) - h_{\text{opt}}\big)^2\big] \to \mu\,\mathcal{E}_{\text{min}} \quad \text{(for large } n\text{)}$$

thus, the accuracy of the converged weights will improve if μ is chosen smaller, however, this will also cause slower convergence speed. This is the basic trade-off of the LMS algorithm.

4. The approximation of ignoring the expectation instruction in the gradient is known as the *stochastic approximation*. It complicates the mathematical aspects of the problem considerably. Indeed, the difference equation

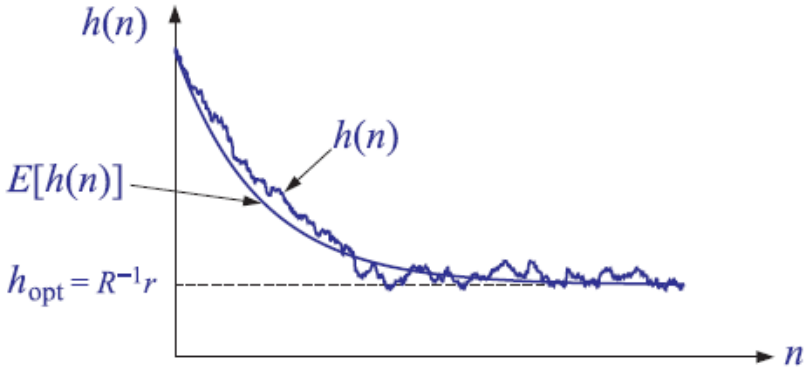
$$h(n+1) = h(n) + 2\mu e_n y_n = h(n) + 2\mu\big(x_n - h(n)y_n\big)y_n$$

makes $h(n)$ depend on the random variable y_n in highly nonlinear fashion, and it is very difficult to discuss even the average behavior of $h(n)$.

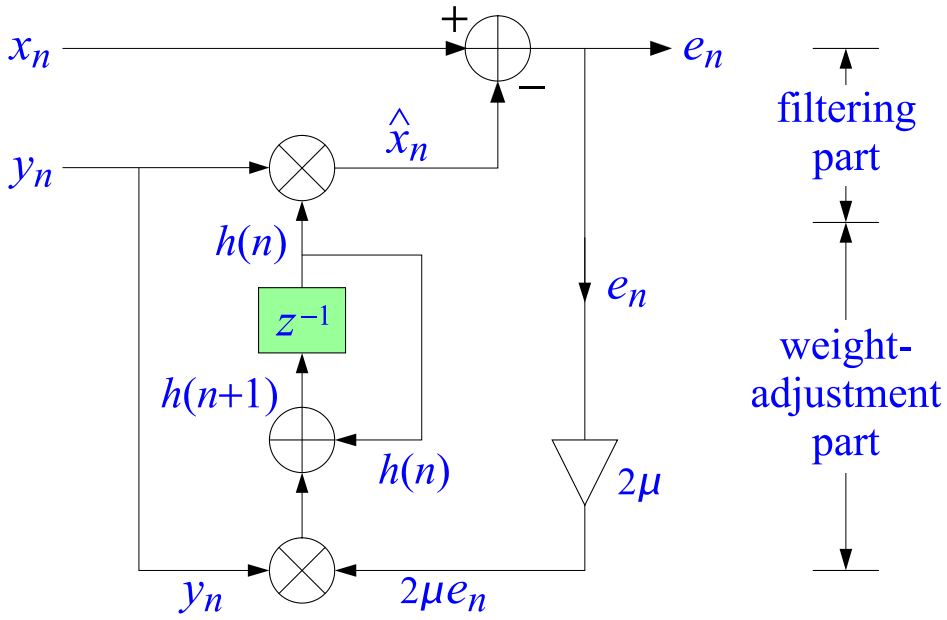
5. In discussing the average behavior of the weight $h(n)$, the following approximation is typically (almost invariably) made in the literature

$$\begin{aligned} E\big[h(n+1)\big] &= E\big[h(n)\big] + 2\mu E\big[x_n y_n\big] - 2\mu E\big[h(n) y_n^2\big] \\ &= E\big[h(n)\big] + 2\mu E\big[x_n y_n\big] - 2\mu E\big[h(n)\big] E\big[y_n^2\big] \\ &= E\big[h(n)\big] + 2\mu r - 2\mu E\big[h(n)\big] R \\ &= \big(1 - 2\mu R\big) E\big[h(n)\big] + 2\mu r \end{aligned}$$

where in the last term, $E\big[h(n)y_n^2\big]$, the expectation $E\big[h(n)\big]$ was factored out, as though $h(n)$ were independent of y_n. With this approximation, the average $E\big[h(n)\big]$ satisfies the same difference equation as the theoretical one. Typically, the weight $h(n)$ will be fluctuating about the theoretical convergence curve as it converges to the optimal value, as shown below



A realization of the CCL is shown below. The filtering part of the realization must be clearly distinguished from the feedback control loop that performs the adaptation of the filter weight.



Historically, the correlation canceler loop was first introduced in the context of adaptive antennas as a *sidelobe canceler*. The CCL is the simplest possible adaptive filter, and forms the elementary *building block* of more complicated, higher-order adaptive filters.

Next, consider a simulation example of the CCL loop. The primary signal x_n was defined by,

$$x_n = -0.8y_n + v_n$$

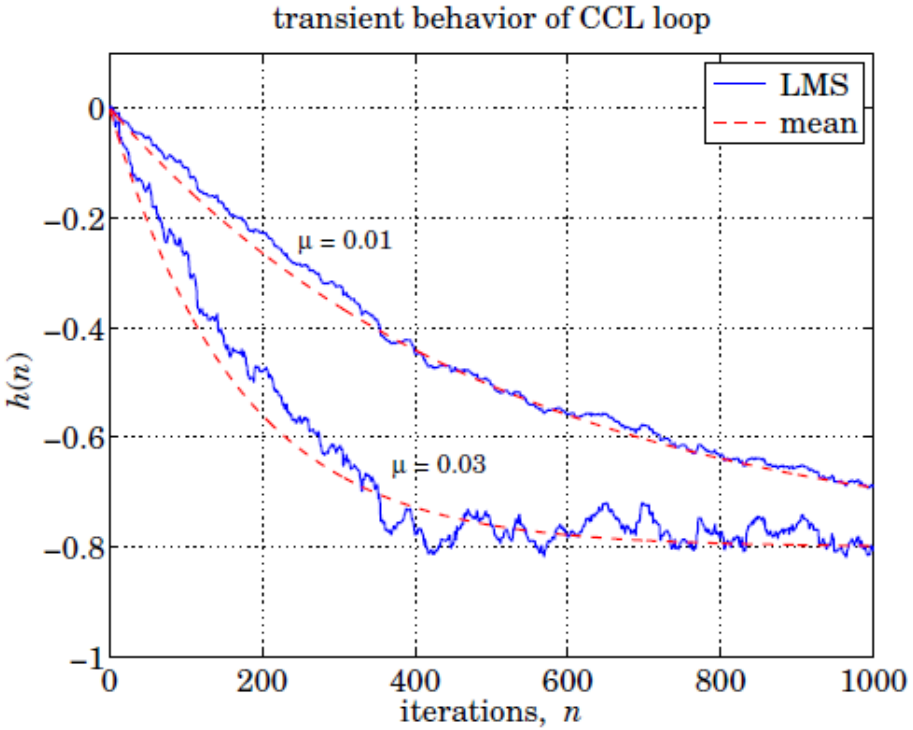
where the first term represents that part of x_n which is correlated with y_n. The part v_n is not correlated with y_n. The theoretical value of the CCL weight is found as follows:

$$r = E[x_n y_n] = -0.8E[y_n y_n] + E[v_n y_n] = -0.8R + 0 \quad \Rightarrow$$
$$h_{\text{opt}} = R^{-1} r = -0.8$$

The corresponding output of the CCL will be $\hat{x}_n = h_{\text{opt}}\, y_n = -0.8y_n$, and therefore it will completely cancel the first term of x_n leaving at the output $e_n = x_n - \hat{x}_n = v_n$.

In the simulation we generated 1000 samples of a zero-mean white-noise signal y_n of variance 0.1, and another independent set of 1000 samples of a zero-mean white-noise signal v_n also of variance 0.1, and computed x_n. The adaptation algorithm was initialized, as is usually done, to zero initial weight $h(0) = 0$.

The figure below shows the transient behavior of the adaptive weight $h(n)$, as well as the theoretical weight $E\big[h(n)\big]$, as a function of the number of iterations n, for the two values, $\mu = 0.03$ and $\mu = 0.01$.

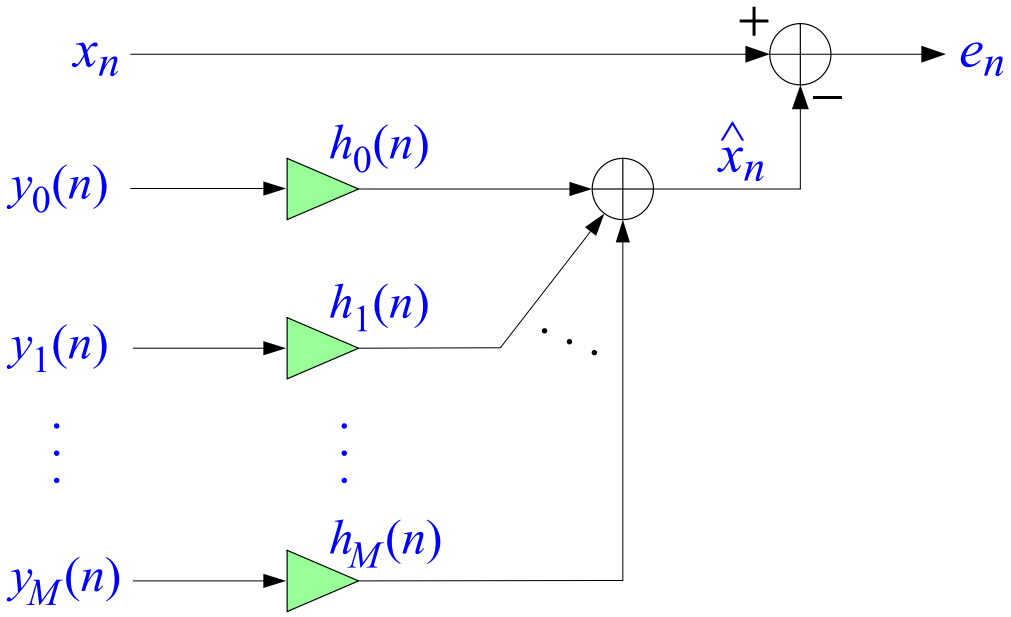


Note that in both cases, the adaptive weight converges to the theoretical value $h_{\text{opt}} = -0.8$, and that the smaller μ is slower but the fluctuations are also smaller. After the adaptive weight has reached its asymptotic value, the CCL begins to operate optimally, removing the correlated part of x_n from the output e_n.

Adaptive Linear Combiner

A straightforward generalization of the correlation canceler loop is the adaptive linear combiner, where one has available a main signal x_n and a number of secondary signals $y_m(n)$, $m = 0, 1, \ldots, M$. These $(M+1)$ secondary signals are to be linearly combined with appropriate weights $h_0, h_1, \ldots, h_M$ to form an estimate of x_n:

$$\hat{x}_n = \sum_{m=0}^{M} h_m(n) y_m(n) = [h_0(n), h_1(n), \ldots, h_M(n)] \begin{bmatrix} y_0(n) \\ y_1(n) \\ \vdots \\ y_M(n) \end{bmatrix} = \mathbf{h}(n)^T \mathbf{y}(n)$$



For time-series applications and spatial arrays, the weights and observation vectors are defined at times instant n, as follows,

$$\mathbf{h} = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_M \end{bmatrix}$$

delay line

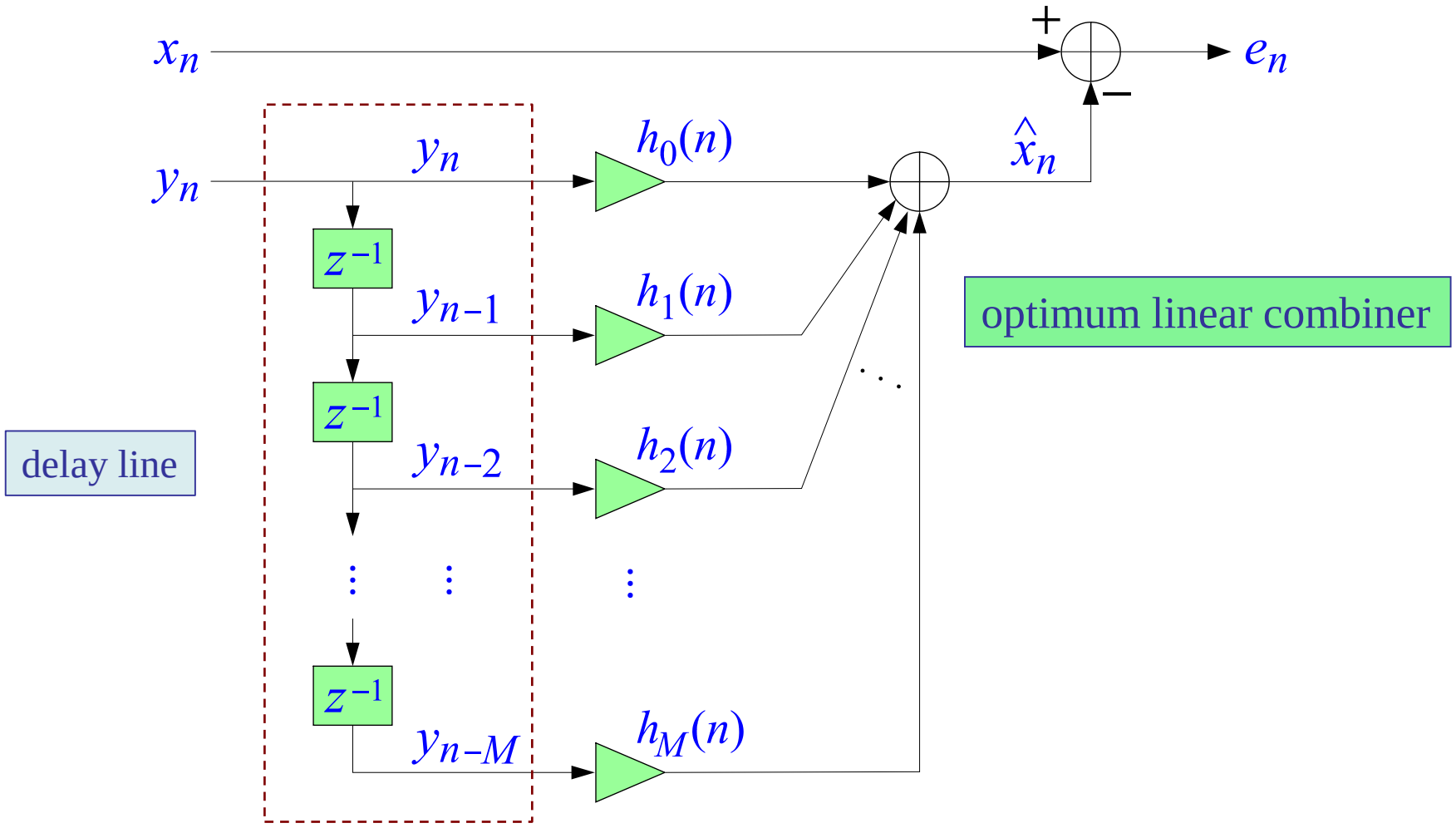
$$\mathbf{y}(n) = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-M} \end{bmatrix} = \text{time series}, \quad \mathbf{y}(n) = \begin{bmatrix} y_0(n) \\ y_1(n) \\ \vdots \\ y_M(n) \end{bmatrix} = \text{spatial arrays}$$

array elements

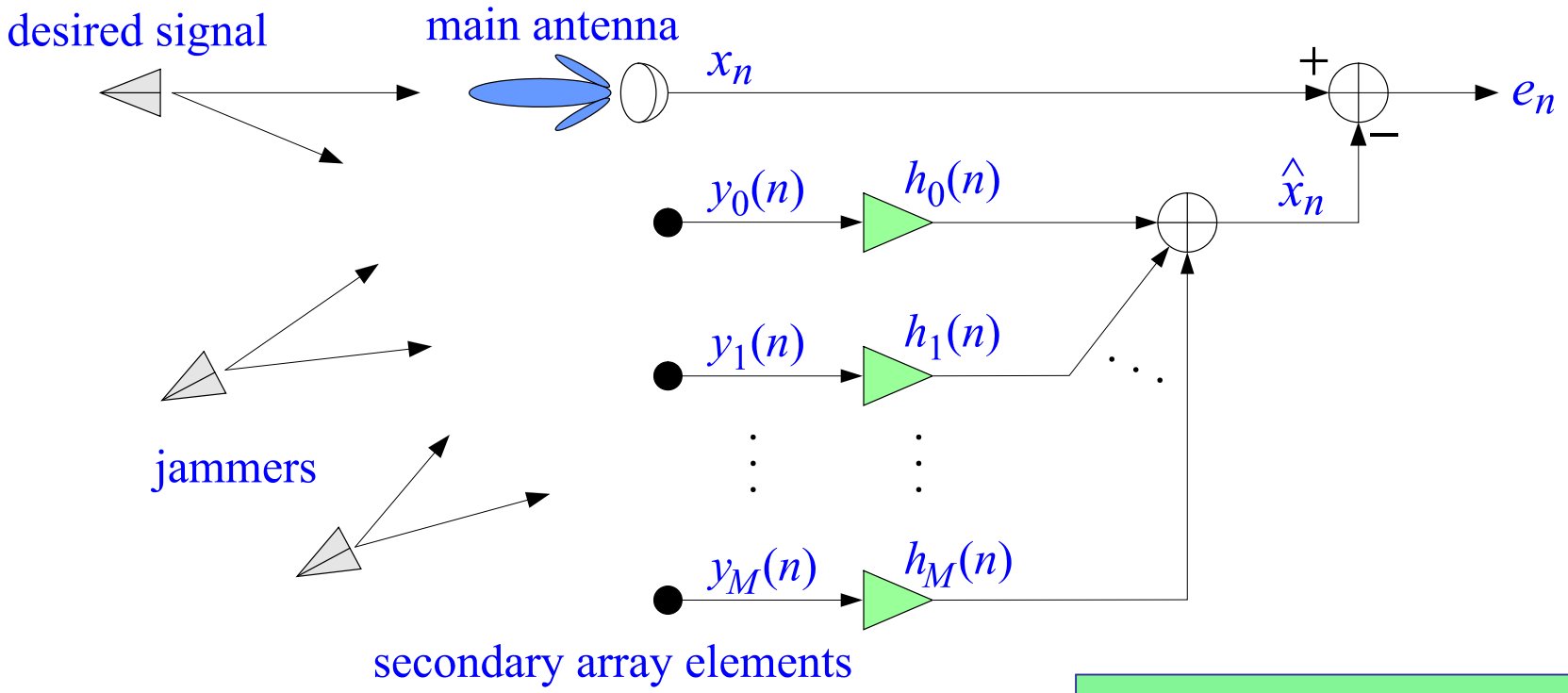
and the estimate at time n,

$$\hat{x}_n = \mathbf{h}^T \mathbf{y}(n)$$

time-series – adaptive Wiener filter



spatial arrays – adaptive antennas



optimum linear combiner

The theoretical and instantaneous gradients are,

$$\frac{\partial \mathcal{E}}{\partial \mathbf{h}} = -2E\big[e_n \mathbf{y}(n)\big] = 2\big(R\mathbf{h}(n) - \mathbf{r}\big)$$

$$\frac{\widehat{\partial \mathcal{E}}}{\partial \mathbf{h}} = -2e_n \mathbf{y}(n)$$

and the resulting LMS algorithm, $\mathbf{h}(n+1) = \mathbf{h}(n) - \mu \dfrac{\widehat{\partial \mathcal{E}}}{\partial \mathbf{h}} = \mathbf{h}(n) + 2\mu\, e_n \mathbf{y}(n)$

for each time instant n,

given x_n, $\mathbf{y}(n)$, $\mathbf{h}(n)$, do,

$\hat{x}_n = \mathbf{h}^T(n)\mathbf{y}(n)$ (LMS algorithm)

$e_n = x_n - \hat{x}_n$

$\mathbf{h}(n+1) = \mathbf{h}(n) + 2\mu\, e_n \mathbf{y}(n)$

mut be appended by the updating of the observation vector $\mathbf{y}(n)$

where in the time-series case, the vector $\mathbf{y}(n)$ must be updated by the delay-line shift,

$$\mathbf{y}(n) = \begin{bmatrix} y_n \\ y_{n-1} \\ y_{n-2} \\ \vdots \\ y_{n-M+1} \\ y_{n-M} \end{bmatrix} \quad \Rightarrow \quad \mathbf{y}(n+1) = \begin{bmatrix} y_{n+1} \\ y_n \\ y_{n-1} \\ \vdots \\ y_{n-M+2} \\ y_{n-M+1} \end{bmatrix}$$

whereas in the array case, $\mathbf{y}(n)$ and $\mathbf{y}(n+1)$ represent the "snapshots" of the incident fields measured at the array elements at times n and $n+1$,

$$\mathbf{y}(n) = \begin{bmatrix} y_0(n) \\ y_1(n) \\ \vdots \\ y_M(n) \end{bmatrix} \quad \Rightarrow \quad \mathbf{y}(n+1) = \begin{bmatrix} y_0(n+1) \\ y_1(n+1) \\ \vdots \\ y_M(n+1) \end{bmatrix}$$

For the time-series case, the following version, which can be translated easily into MATLAB, uses a temporary $(M+1)\times 1$ state vector $\mathbf{w}$ that represents $\mathbf{y}(n)$ and gets updated at each iteration. Using MATLAB notation,

for $n = 1, 2, 3, \ldots,$

read input pair $x_n,\ y_n$

$w(1) = y_n$

$\hat{x}_n = \mathbf{h}^T \mathbf{w}$

$e_n = x_n - \hat{x}_n$

$\mathbf{h} = \mathbf{h} + 2\mu\, e_n \mathbf{w}$

$\mathbf{w} = [w(1);\ \mathbf{w}(1 : \text{end}-1)]$

$$\mathbf{w} = \begin{bmatrix} w(1) \\ w(2) \\ \vdots \\ w(M+1) \end{bmatrix}$$

updates **y**(*n*) to **y**(*n*+1)

The algorithm is initialized to zero initial weights and state vector, that is, in MATLAB notation,

$$\mathbf{h} = \text{zeros}(M+1, 1)$$
$$\mathbf{w} = \text{zeros}(M+1, 1)$$

Selecting the filter order

How does one select the order M of the adaptive filter? The rule-of-thumb is that the filter must have at least as many delays as that part of x_n which is correlated with y_n. To see this, suppose x_n is related to y_n by

$$x_n = c_0 y_n + c_1 y_{n-1} + \cdots + c_L y_{n-L} + u_n$$

where u_n is uncorrelated with y_n. Then, the filter order must be at least L. If $M \geq L$, we can write:

$$x_n = c_0 y_n + c_1 y_{n-1} + \cdots + c_M y_{n-M} + u_n = \mathbf{c}^T \mathbf{y}(n) + u_n$$

where $\mathbf{c}$ is the extended vector having, $c_i = 0$ for $L+1 \leq i \leq M$. The cross-correlation between x_n and $\mathbf{y}(n)$ is then,

$$\mathbf{r} = E\big[x_n \mathbf{y}(n)\big] = E\big[\big(\mathbf{y}^T(n)\mathbf{c}\big)\mathbf{y}(n)\big] = E\big[\mathbf{y}(n)\mathbf{y}^T(n)\big]\mathbf{c} = R\,\mathbf{c}$$

Thus, the Wiener solution will be, $\mathbf{h} = R^{-1}\mathbf{r} = \mathbf{c}$. This, in turn, implies the complete cancellation of the y-dependent part of x_n. Indeed, $\hat{x}_n = \mathbf{h}^T \mathbf{y}(n) = \mathbf{c}^T \mathbf{y}(n)$, and

$$e_n = x_n - \hat{x}_n = \big(\mathbf{c}^T \mathbf{y}(n) + u_n\big) - \mathbf{c}^T \mathbf{y}(n) = u_n$$

What happens if we underestimate the filter order and choose $M < L$? In this case, we expect to cancel completely the first M terms of x_n, and to cancel the remaining terms as much as possible. To see this, we separate out the first M terms writing

$$x_n = [c_0, \ldots, c_M] \begin{bmatrix} y_n \\ \vdots \\ y_{n-M} \end{bmatrix} + [c_{M+1}, \ldots, c_L] \begin{bmatrix} y_{n-M-1} \\ \vdots \\ y_{n-L} \end{bmatrix} + u_n$$

$$x_n = \mathbf{c}_1^T \mathbf{y}_1(n) + \mathbf{c}_2^T \mathbf{y}_2(n) + u_n$$

The problem of estimating x_n using an Mth order filter is equivalent to the problem of estimating x_n from $\mathbf{y}_1(n)$. The cross-correlation between x_n and $\mathbf{y}_1(n)$ is,

$$E\big[x_n \mathbf{y}_1(n)\big] = E\big[\mathbf{y}_1(n)\mathbf{y}_1^T(n)\big]\mathbf{c}_1 + E\big[\mathbf{y}_1(n)\mathbf{y}_2^T(n)\big]\mathbf{c}_2$$

It follows that the optimum estimate of x_n based on $\mathbf{y}_1(n)$ will be,

$$\begin{aligned}\hat{x}_n &= E\big[x_n\mathbf{y}_1^T(n)\big]E\big[\mathbf{y}_1(n)\mathbf{y}_1^T(n)\big]^{-1}\mathbf{y}_1(n)\\ &= \big(\mathbf{c}_1^T E\big[\mathbf{y}_1(n)\mathbf{y}_1^T(n)\big] + \mathbf{c}_2^T E\big[\mathbf{y}_2(n)\mathbf{y}_1^T(n)\big]\big)E\big[\mathbf{y}_1(n)\mathbf{y}_1^T(n)\big]^{-1}\mathbf{y}_1(n)\\ &= \big(\mathbf{c}_1^T + \mathbf{c}_2^T E\big[\mathbf{y}_2(n)\mathbf{y}_1^T(n)\big]E\big[\mathbf{y}_1(n)\mathbf{y}_1^T(n)\big]^{-1}\big)\mathbf{y}_1(n)\\ &= \mathbf{c}_1^T\mathbf{y}_1(n) + \mathbf{c}_2^T\hat{\mathbf{y}}_{2/1}(n)\end{aligned}$$

where,

$$\hat{\mathbf{y}}_{2/1}(n) = E\big[\mathbf{y}_2(n)\mathbf{y}_1^T(n)\big]E\big[\mathbf{y}_1(n)\mathbf{y}_1^T(n)\big]^{-1}\mathbf{y}_1(n)$$

and is recognized as the optimum estimate of $\mathbf{y}_2(n)$ based on $\mathbf{y}_1(n)$. Thus, the estimation error will be,

$$\begin{aligned}e_n &= x_n - \hat{x}_n = \big[\mathbf{c}_1^T\mathbf{y}_1(n) + \mathbf{c}_2^T\mathbf{y}_2(n) + u_n\big] - \big[\mathbf{c}_1\mathbf{y}_1(n) + \mathbf{c}_2^T\hat{\mathbf{y}}_{2/1}(n)\big]\\ e_n &= \mathbf{c}_2^T\big[\mathbf{y}_2(n) - \hat{\mathbf{y}}_{2/1}(n)\big] + u_n\end{aligned}$$

which shows that the $\mathbf{y}_1(n)$ part is removed completely, and the $\mathbf{y}_2(n)$ part is removed as much as possible.

Speed of Convergence

The convergence properties of the LMS algorithm may be derived by restoring the expectation values where they should be, that is,

$$\frac{\partial \mathcal{E}}{\partial \mathbf{h}} = -2E\big[e_n \mathbf{y}(n)\big] = 2\big(R\mathbf{h}(n) - \mathbf{r}\big)$$

resulting in the difference equation for the weight vector,

$$\mathbf{h}(n+1) = \mathbf{h}(n) - \mu \frac{\partial \mathcal{E}}{\partial \mathbf{h}}$$

$$\mathbf{h}(n+1) = \mathbf{h}(n) - 2\mu\big(R\mathbf{h}(n) - \mathbf{r}\big)$$

$$\mathbf{h}(n+1) = \mathbf{h}(n) - 2\mu R\mathbf{h}(n) + 2\mu\mathbf{r}$$

$$\mathbf{h}(n+1) = (I - 2\mu R)\mathbf{h}(n) + 2\mu\mathbf{r}$$

with solution, where, $\mathbf{h}_{\text{opt}} = R^{-1}\mathbf{r}$,

$$\boxed{\mathbf{h}(n) = \mathbf{h}_{\text{opt}} + (I - 2\mu R)^n\big(\mathbf{h}(0) - \mathbf{h}_{\text{opt}}\big)}$$

Convergence to $\mathbf{h}_{\text{opt}}$ requires that the quantity, $(1 - 2\mu\lambda)$, for every eigenvalue λ of R, have magnitude less than one (we assume that R has full rank and therefore all its eigenvalues are positive):

$$|1 - 2\mu\lambda| < 1 \quad \Leftrightarrow \quad -1 < 1 - 2\mu\lambda < 1 \quad \Leftrightarrow \quad 0 < \mu < \frac{1}{\lambda}$$

This condition will be guaranteed if we require this inequality for λ_{max}, the maximum eigenvalue:

$$\boxed{0 < \mu < \frac{1}{\lambda_{\text{max}}}} \qquad \text{(convergence condition)}$$

Note that λ_{max} can be bounded from above by the trace of R,

$$\lambda_{\text{max}} < \text{tr}(R)$$

and one may require instead the easier condition,

$$\mu < \frac{1}{\text{tr}(R)} < \frac{1}{\lambda_{\text{max}}}$$

As for the speed of convergence, suppose that μ is selected half-way within its allowed range, i.e., near $1/2\lambda_{\max}$, then the rate of convergence will depend on the slowest converging term of the form, $(1-2\mu\lambda)^n$, that is, the term having $|1-2\mu\lambda|$ as close to one as possible.

This occurs for the smallest eigenvalue $\lambda = \lambda_{\min}$. Thus, the slowest converging term is effectively given by,

$$(1-2\mu\lambda_{\min})^n = \left(1-\frac{\lambda_{\min}}{\lambda_{\max}}\right)^n$$

The effective time constant in seconds is obtained by writing $t = nT$, where T is the sampling period, and using the approximation,

$$\left(1-\frac{\lambda_{\min}}{\lambda_{\max}}\right)^n \simeq \exp\left(-\frac{\lambda_{\min}}{\lambda_{\max}}n\right) = e^{-t/\tau}$$

where

$$\boxed{\tau = T\,\frac{\lambda_{\max}}{\lambda_{\min}} = \text{learning time constant}}$$

The eigenvalue spread of the covariance matrix R,

$$\frac{\lambda_{\max}}{\lambda_{\min}}$$

controls, therefore, the speed of convergence, or, the learning time constant.

The convergence can be as fast as one sampling instant T if the eigenvalue spread is small, i.e., $\lambda_{\max}/\lambda_{\min} \simeq 1$.

But, the convergence will be slow if the eigenvalue spread is large. As we shall see shortly, a large spread in the eigenvalues of R corresponds to a highly self-correlated signal y_n.

Thus, we obtain the qualitative result that in situations where the secondary signal y_n is strongly self-correlated, the convergence of the gradient-based LMS algorithm will be slow.

In many applications, such as channel equalization, the convergence must be as quick as possible. Alternative adaptation schemes exist that combine the computational simplicity of the LMS algorithm with a fast speed of convergence. Examples are the fast RLS and the adaptive lattice algorithms.

Accelerating the LMS Algorithm – Newton's Method

The possibility of accelerating the convergence rate may be seen by considering a more general version of the gradient-descent algorithm in which the time update for the weight vector is chosen as

$$\Delta \mathbf{h} = -\mathcal{M} \frac{\partial \mathcal{E}}{\partial \mathbf{h}}$$

where $\mathcal{M}$ is a *positive definite and symmetric* matrix. The LMS steepest descent case is obtained as a special case of this when $\mathcal{M}$ is proportional to the unit matrix, $\mathcal{M} = \mu I$. The above choice guarantees convergence towards the minimum of the performance index $\mathcal{E}(\mathbf{h})$, indeed,

$$\mathcal{E}(\mathbf{h} + \Delta \mathbf{h}) \simeq \mathcal{E}(\mathbf{h}) + \Delta \mathbf{h}^T \left(\frac{\partial \mathcal{E}}{\partial \mathbf{h}} \right)$$

$$\mathcal{E}(\mathbf{h} + \Delta \mathbf{h}) = \mathcal{E}(\mathbf{h}) - \left(\frac{\partial \mathcal{E}}{\partial \mathbf{h}} \right)^T \mathcal{M} \left(\frac{\partial \mathcal{E}}{\partial \mathbf{h}} \right) \leq \mathcal{E}(\mathbf{h})$$

since $\mathcal{M}$ was assumed to be positive-definite and symmetric.

Since the performance index is

$$\mathcal{E} = E[e_n^2] = E\big[\big(x_n - \mathbf{h}^T\mathbf{y}(n)\big)^2\big] = E[x_n^2] - 2\mathbf{h}^T\mathbf{r} + \mathbf{h}^T R\,\mathbf{h}$$

it follows that $\partial\mathcal{E}/\partial\mathbf{h} = -2(\mathbf{r} - R\,\mathbf{h})$, and the difference equation for the adaptive weights will become,

$$\mathbf{h}(n+1) = \mathbf{h}(n) + \Delta\mathbf{h}(n) = \mathbf{h}(n) + 2\mathcal{M}\big(\mathbf{r} - R\,\mathbf{h}(n)\big)$$

or,

$$\mathbf{h}(n+1) = (I - 2\mathcal{M}R)\mathbf{h}(n) + 2\mathcal{M}\mathbf{r}$$

with solution for $n \geq 0$,

$$\boxed{\mathbf{h}(n) = \mathbf{h}_{\text{ opt}} + (I - 2\mathcal{M}R)^n(\mathbf{h}(0) - \mathbf{h}_{\text{ opt}})}$$

where, $\mathbf{h}_{\text{ opt}} = R^{-1}\mathbf{r}$, is the asymptotic value, and $\mathbf{h}(0)$, the initial value. The choice of $\mathcal{M}$ can drastically affect the speed of convergence. For example, if $\mathcal{M}$ is chosen as

$$\mathcal{M} = (2R)^{-1}$$

then

$$I - 2\mathcal{M}R = 0$$

and the convergence occurs in just one time step!

This choice of $\mathcal{M}$ is equivalent to Newton's method of solving the system of equations,

$$\mathbf{f}(\mathbf{h}) = \frac{\partial \mathcal{E}}{\partial \mathbf{h}} = 0$$

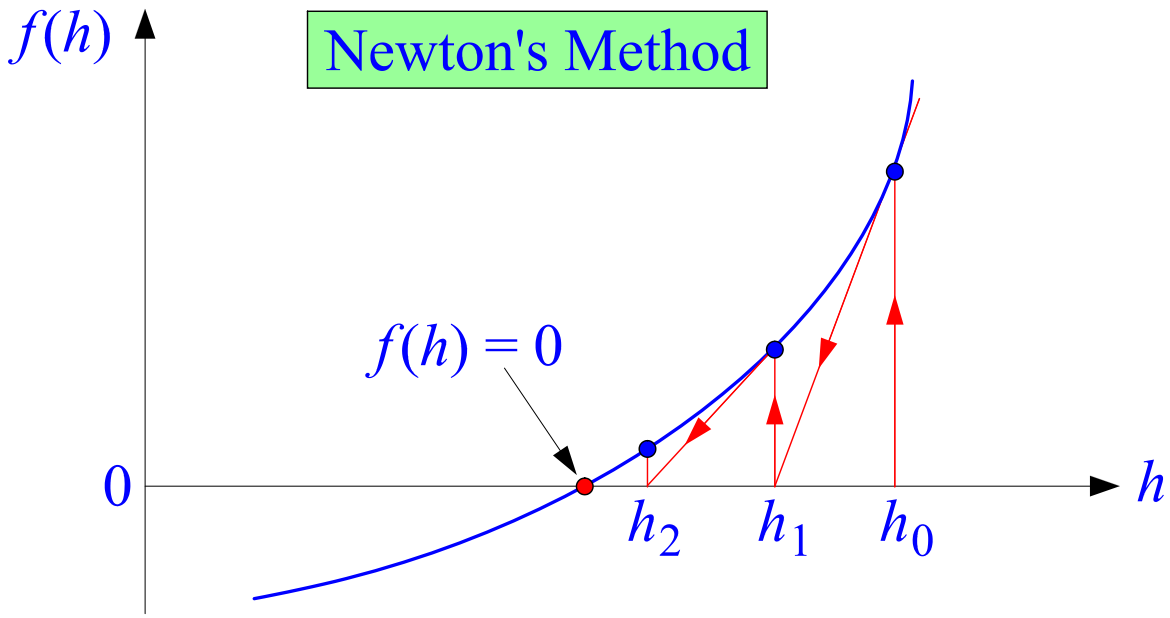
for the optimal weights. Indeed, Newton's method linearizes about each point $\mathbf{h}$ to get the next point, that is, $\Delta\mathbf{h}$ is selected such that

$$\mathbf{f}(\mathbf{h} + \Delta\mathbf{h}) \simeq \mathbf{f}(\mathbf{h}) + \left(\frac{\partial \mathbf{f}}{\partial \mathbf{h}}\right) \Delta\mathbf{h} = 0$$

where we expanded to first order in $\Delta\mathbf{h}$. Solving for $\Delta\mathbf{h}$, we obtain

$$\Delta\mathbf{h} = -\left(\frac{\partial \mathbf{f}}{\partial \mathbf{h}}\right)^{-1} \mathbf{f}(\mathbf{h})$$

$$\mathbf{f}(\mathbf{h} + \Delta\mathbf{h}) \simeq \mathbf{f}(\mathbf{h}) + \left(\frac{\partial \mathbf{f}}{\partial \mathbf{h}}\right) \Delta\mathbf{h} = 0$$



$$\Delta\mathbf{h} = -\left(\frac{\partial\mathbf{f}}{\partial\mathbf{h}}\right)^{-1}\mathbf{f}(\mathbf{h})$$

But since, $\mathbf{f}(\mathbf{h}) = -2(\mathbf{r} - R\mathbf{h})$, we have, $\partial\mathbf{f}/\partial\mathbf{h} = 2R$. Therefore, the choice, $\mathcal{M} = (2R)^{-1}$, corresponds precisely to Newton's update.

Note that the property that Newton's method converges in one step is a well-known property valid for *quadratic* performance indices (in such cases, the gradient $\mathbf{f}(\mathbf{h})$ is already linear in $\mathbf{h}$ and therefore Newton's local linearization is exact).

The important property about the choice $\mathcal{M} = (2R)^{-1}$ is that $\mathcal{M}$ is proportional to the inverse of R. An alternative choice could have been $\mathcal{M} = \alpha R^{-1}$. In this case $I - 2\mathcal{M}R$ becomes proportional to the identity matrix,

$$I - 2\mathcal{M}R = (1 - 2\alpha)I$$

and having equal eigenvalues. Stability requires that $|1 - 2\alpha| < 1$, or equivalently, $0 < \alpha < 1$, with Newton's choice corresponding exactly to the middle of this interval, $\alpha = 1/2$.

Therefore, the disparity between the eigenvalues that could slow down the convergence rate is eliminated, and all eigenmodes converge at the same rate (which is faster the more $\mathcal{M}$ resembles $(2R)^{-1}$).

The implementation of such Newton-like methods requires knowledge of R, which we do not have (if we did, we would simply compute the Wiener solution, $\mathbf{h}_{\text{opt}} = R^{-1}\mathbf{r}$.)

However, as we shall see later, the so-called recursive least-squares algorithms (RLS) effectively provide an implementation of Newton-type methods, and that is the reason for their extremely fast convergence.

Adaptive lattice filters also have very fast convergence properties. In that case, because of the orthogonalization of the successive lattice stages of the filter, the matrix R is already diagonal (in the decorrelated basis) and the matrix $\mathcal{M}$ can also be chosen to be diagonal so as to equalize and speed up the convergence rate of all the filter coefficients.

Recursive least-squares and adaptive lattice filters are discussed in AOSP-Sections 16.16 and 16.18, respectively.

Finally, we would like to demonstrate the previous statement that a strongly correlated signal y_n has a large spread in the eigenvalue spectrum of its covariance matrix. For simplicity, consider the 2×2 case

$$R = E\big[\mathbf{y}(n)\mathbf{y}^T(n)\big] = E\left[\begin{bmatrix} y_n \\ y_{n-1} \end{bmatrix}\begin{bmatrix} y_n, & y_{n-1} \end{bmatrix}\right] = \begin{bmatrix} R(0) & R(1) \\ R(1) & R(0) \end{bmatrix}$$

The two eigenvalues are easily found to be

$$\begin{aligned} \lambda_{\text{min}} &= R(0) - |R(1)| \\ \lambda_{\text{max}} &= R(0) + |R(1)| \end{aligned}$$

and therefore, the ratio $\lambda_{\text{max}}/\lambda_{\text{min}}$ is given by

$$\frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} = \frac{R(0) + |R(1)|}{R(0) - |R(1)|}$$

Since for an autocorrelation function we always have, $|R(1)| \leq R(0)$, it follows that the largest value of $|R(1)|$ is $\pm R(0)$, implying that for highly correlated signals the ratio $\lambda_{\text{max}}/\lambda_{\text{min}}$ will be very large.

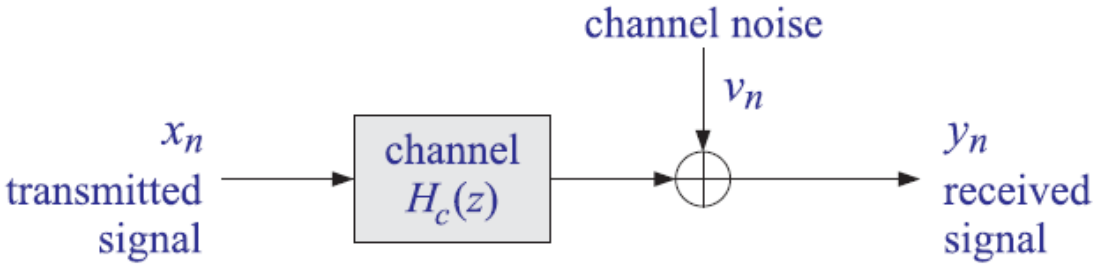
On the other hand, for uncorrelated signals, $R(1) \approx 0$, and the eigenvalue spread becomes unity, $\lambda_{\text{max}}/\lambda_{\text{min}} \approx 1$.

Next, we look briefly at some applications:

- adaptive channel equalizers
- adaptive echo cancelers
- adaptive system identification
- adaptive inverse modeling
- adaptive noise canceling
- adaptive antenna sidelobe cancelers
- adaptive line enhancer
- adaptive linear prediction
- adaptive spectrum estimation
- adaptive angle-of-arrival estimation

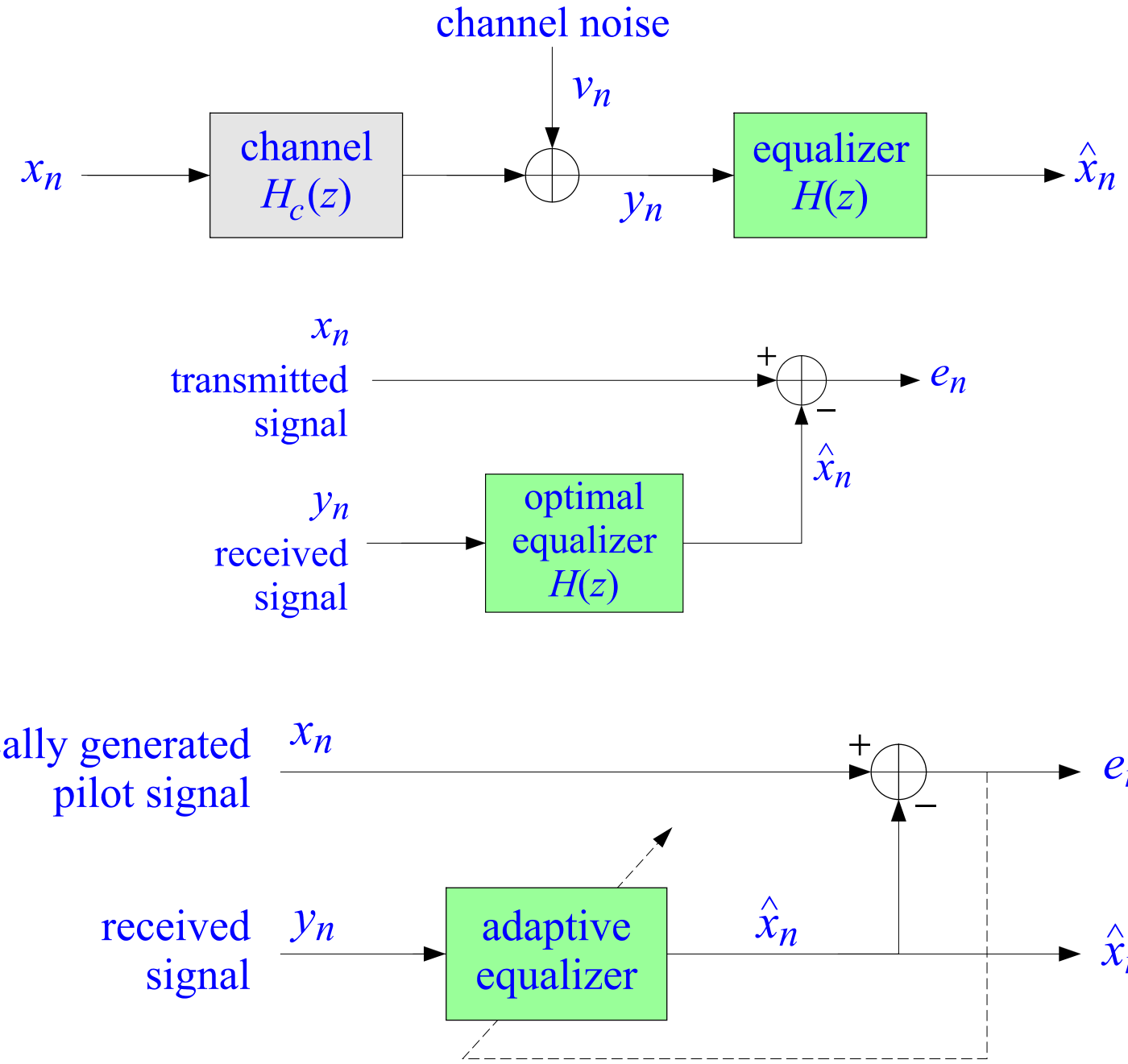
Adaptive Channel Equalizers

Channels used in digital data transmissions can be modeled very often by linear time-invariant systems. The standard model for such a channel including channel noise is shown below.



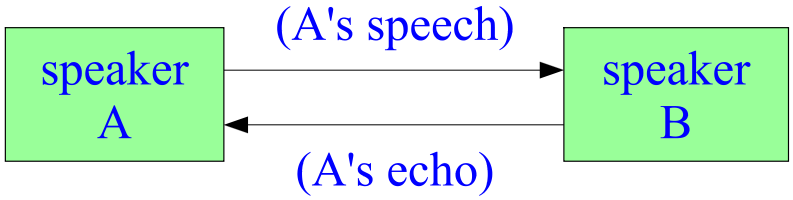
where $H_c(z)$ is the transfer function for the channel and v_n, the channel noise, assumed to be additive white gaussian noise. The transfer function $H_c(z)$ incorporates the effects of the modulator and demodulator filters, as well as the channel distortions.

The purpose of a channel equalizer is to undo the distorting effects of the channel and recover, from the received waveform y_n, the signal x_n that was transmitted. Typically, a channel equalizer will be an FIR filter with enough taps to approximate the inverse transfer function of the channel.

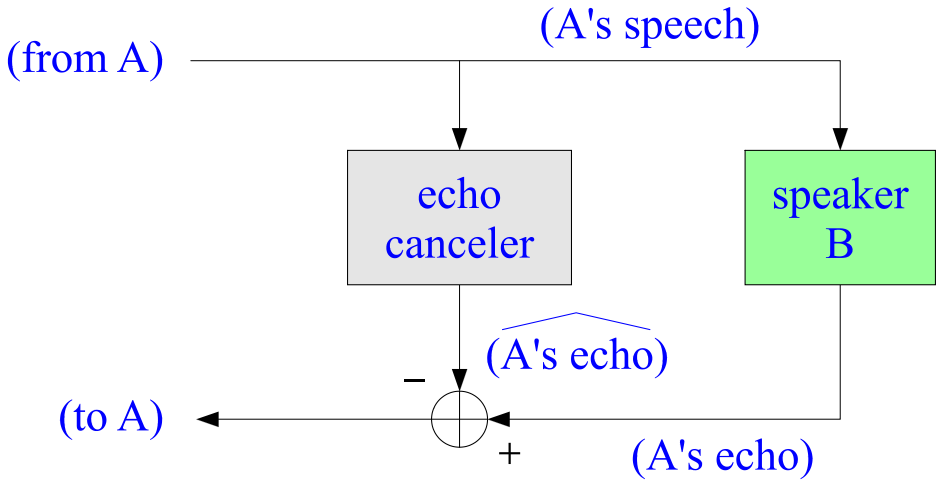

channel noise
v_n
x_n
channel
$H_c(z)$
y_n
equalizer
$H(z)$
$\hat{x}_n$
x_n
transmitted
signal
+
−
e_n
$\hat{x}_n$
y_n
received
signal
optimal
equalizer
$H(z)$
locally generated
pilot signal
x_n
+
−
e_n
received
signal
y_n
adaptive
equalizer
$\hat{x}_n$
$\hat{x}_n$

Adaptive Echo Cancelers

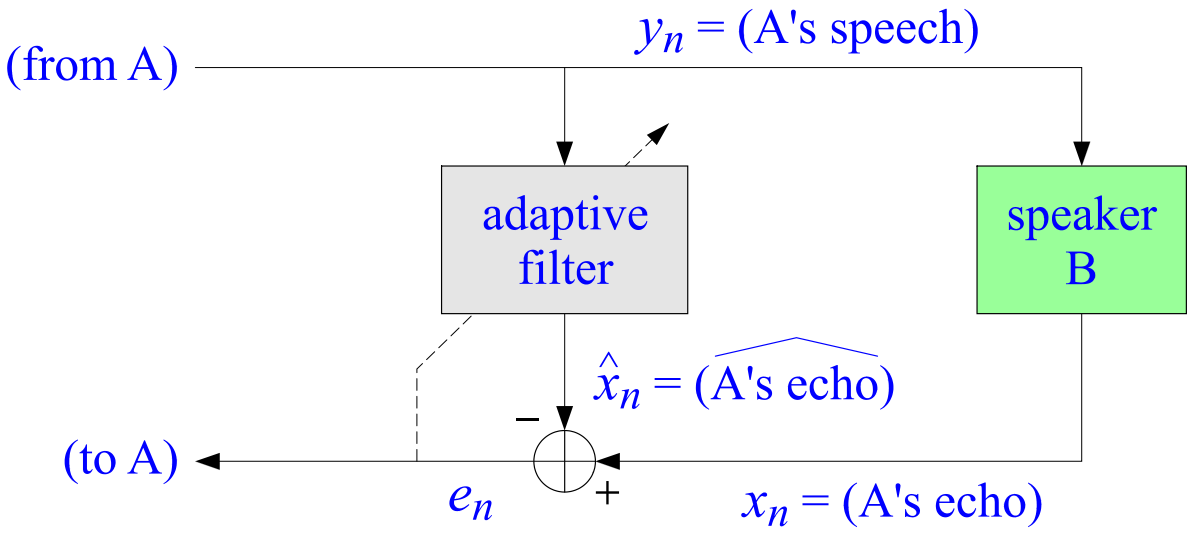
Consider two speakers A and B connected to each other by the telephone network. As a result of various impedance mismatches, when A's speech reaches B, it manages to "leak" through and echoes back to speaker A, as though it were B's speech.



An echo canceler may be placed near B's end, as shown.



It produces an (optimum) estimate of A's echo through B's circuits, and then proceeds to cancel it from the signal returning to speaker A. Again, this is another case for which optimal filtering ideas are ideally suited. An adaptive echo canceler is an adaptive FIR filter placed as shown.



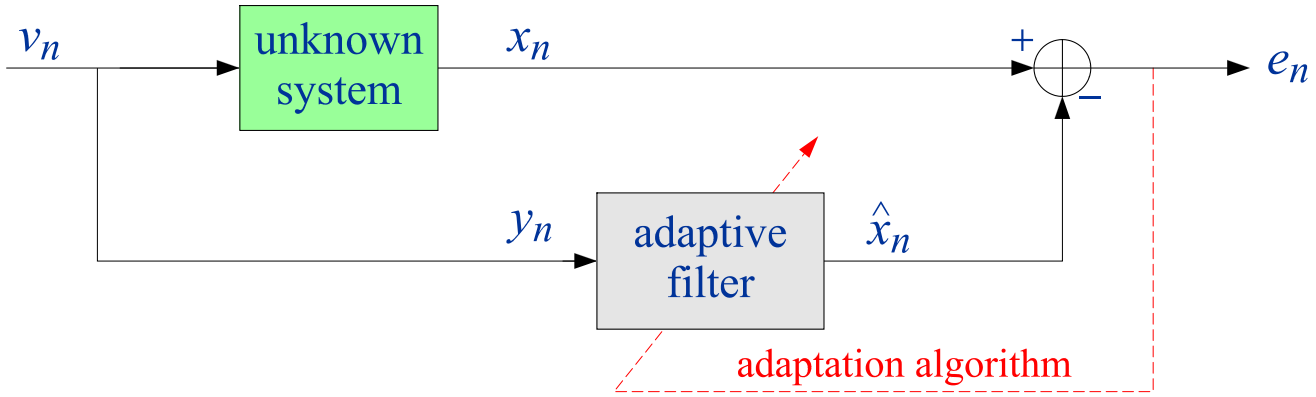
As always, the adaptive filter will adjust itself to cancel any correlations that might exist between the secondary signal y_n (A's speech) and the primary signal x_n (A's echo).

Adaptive System Identification

An adaptive filter can be used to identify an unknown system. A known signal v_n is sent to the input of the system, and its output x_n serves as the primary input to the adaptive filter, while $y_n = v_n$, serves as its secondary input.

The adaptive filter will try to adjust itself until the error, $e_n = x_n - \hat{x}_n$, is minimized, ideally becoming zero. At that point, the adaptive filter will be the same as the unknown system, since both are driven by the same input and both are producing effectively the same output.

Assuming that the unknown system can be modeled as a (possibly long) FIR filter, then if the FIR adaptive filter has at least as many wights as the unknown system, the adaptive filter weights will converge to the system's weights. In particular, if the input signal v_n is chosen to be uncorrelated noise (i.e., no eigenvalue spread), then the convergence will be accelerated.



Adaptive Inverse Modeling

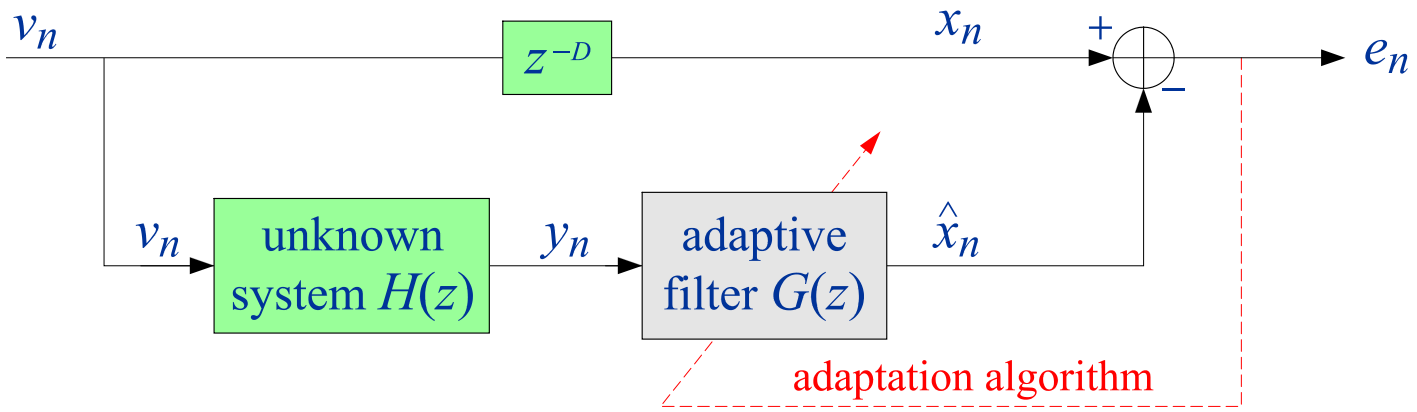
A variation of the adaptive system identification case is shown below that allows the identification of an inverse system, say, $H(z)$. The adaptive filter will converge to a transfer function, say, $G(z)$, which ideally is trying to drive the error output to zero,

$$e_n = x_n - \hat{x}_n \approx 0 \quad \Rightarrow \quad E(z) = z^{-D}V(z) - H(z)G(z)V(z) \approx 0$$

or,

$$\boxed{G(z) = \frac{z^{-D}}{H(z)}}$$

The delay z^{-D} is needed in case $H(z)$ has zeros outside the unit circle, which become poles of $1/H(z)$, and can be made stable and causal by a sufficient amount of delay.



Adaptive Noise Canceling

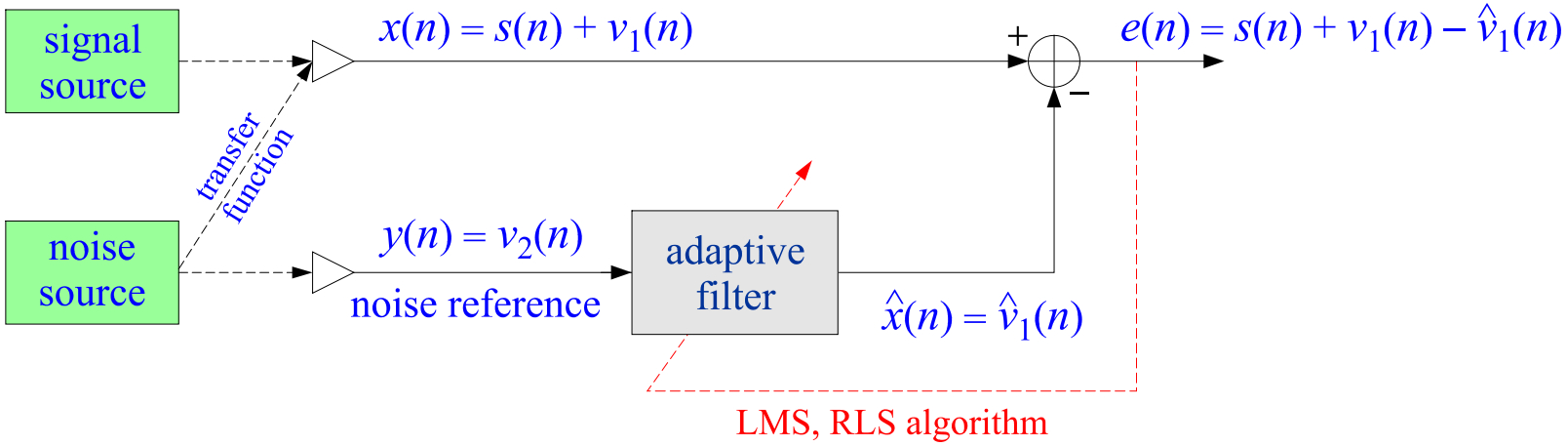
$$x(n) = s(n) + v_1(n) = \text{desired signal plus noise}$$

$$y(n) = v_2(n) = \text{noise reference}$$

$$\hat{x}(n) = \hat{v}_1(n) = \text{adaptive filter output}$$

$$e(n) = x(n) - \hat{x}(n) = s(n) + \underbrace{v_1(n) - \hat{v}_1(n)}_{\text{canceled noise}} \approx s(n)$$

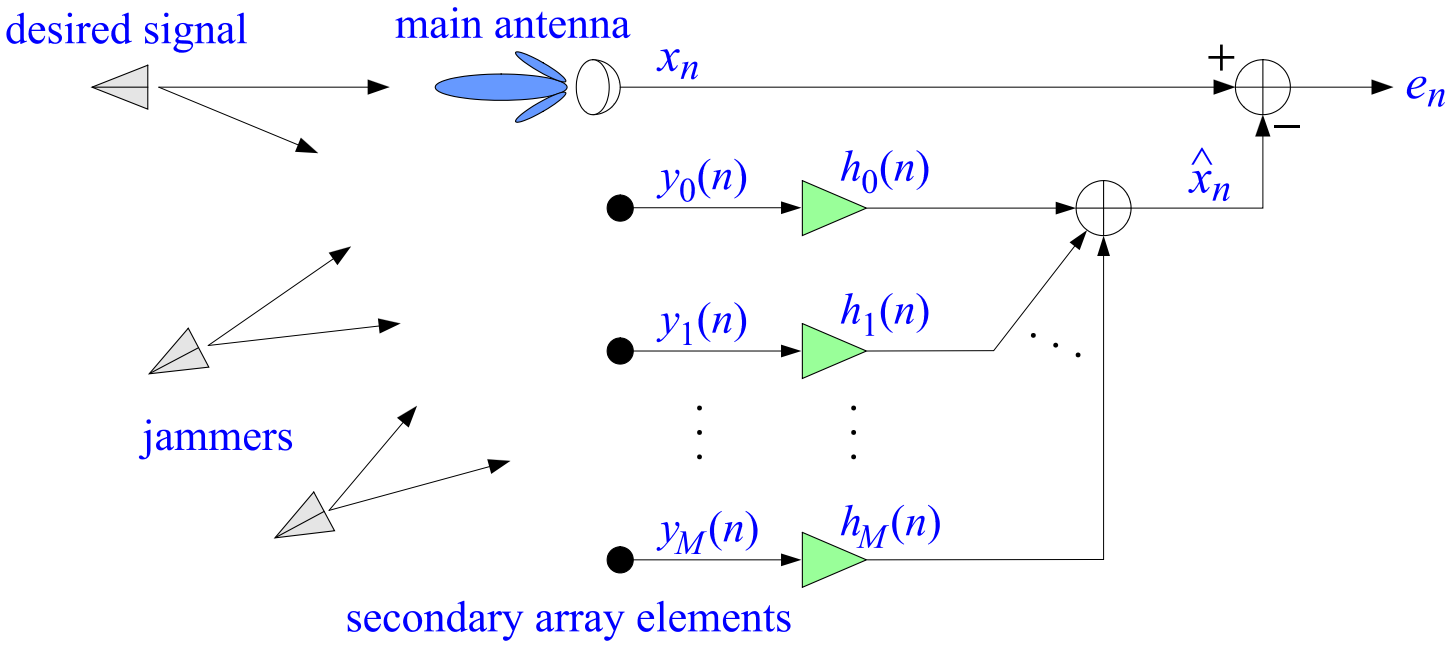
The adaptive filter processes an available noise reference signal $v_2(n)$ to make an estimate of the noise component $v_1(n)$ of the primary signal, and then proceeds to cancel it from the error output, thus, effectively resulting in an estimate of the desired signal. The basic assumption of the noise-canceling system is that $v_2(n)$ is correlated with $v_1(n)$, but not with $s(n)$.



Adaptive Sidelobe Canceler

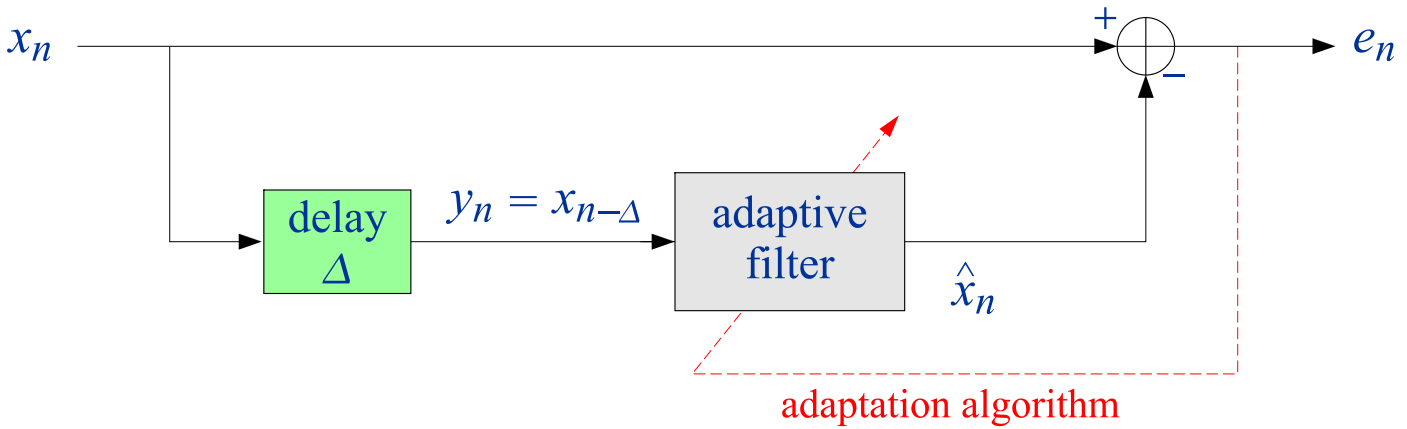
The transmitting antenna of the desired signal is assumed to be oriented along the mainlobe of the primary receiving antenna. Jamming signals are picked up by the primary antenna through its sidelobes, and also by the secondary antennas.

Since the jamming signals picked up by the secondary antennas are correlated with those picked up by the sidelobes, the adaptive linear combiner will act as a noise canceler that estimates the sidelobe signals and cancels them from the error output, leaving the desired signal.

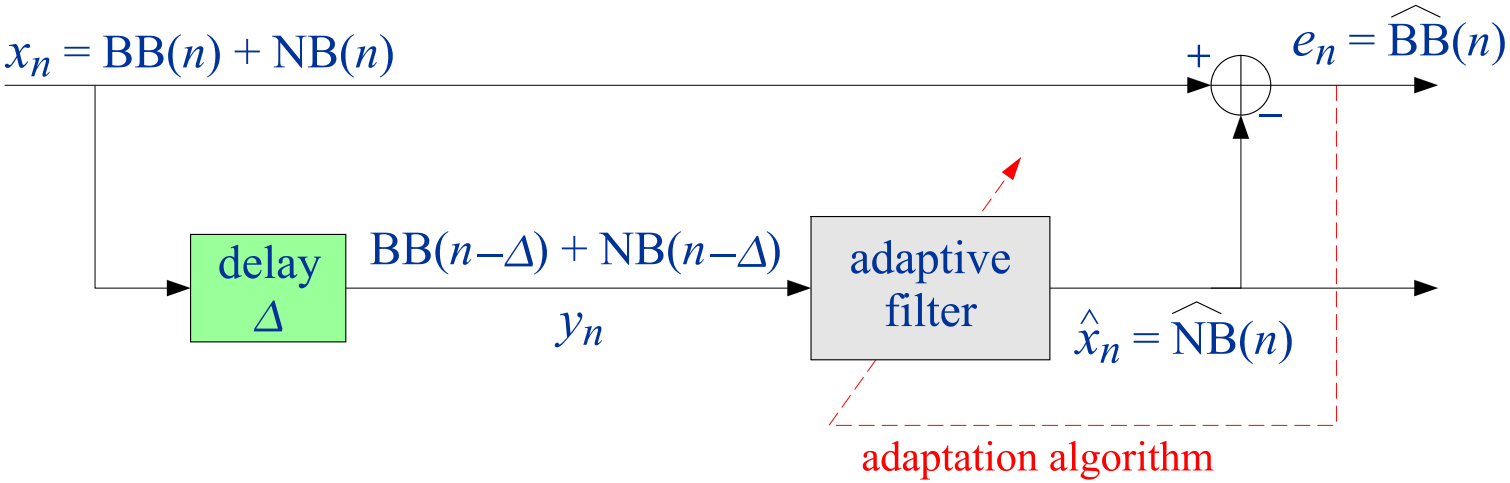


Adaptive Line Enhancer

A special case of adaptive noise canceling is when there is only one signal x_n available which is contaminated by noise. In such a case, the signal x_n provides its own reference signal y_n, which is taken to be a delayed replica of x_n, that is, $y_n = x_{n-\Delta}$, as shown below, and referred to as the adaptive line enhancer (ALE). The adaptive filter will respond by canceling any components of the main signal x_n that are in any way correlated with the secondary signal $y_n = x_{n-\Delta}$.



It can also be thought of as an adaptive signal separator. Suppose the signal x_n consists of two parts: a narrowband component that has long-range correlations such as a sinusoid, and a broadband component, such as white-noise, which will tend to have short-range correlations. One of these could represent the desired signal and the other an undesired interfering noise.



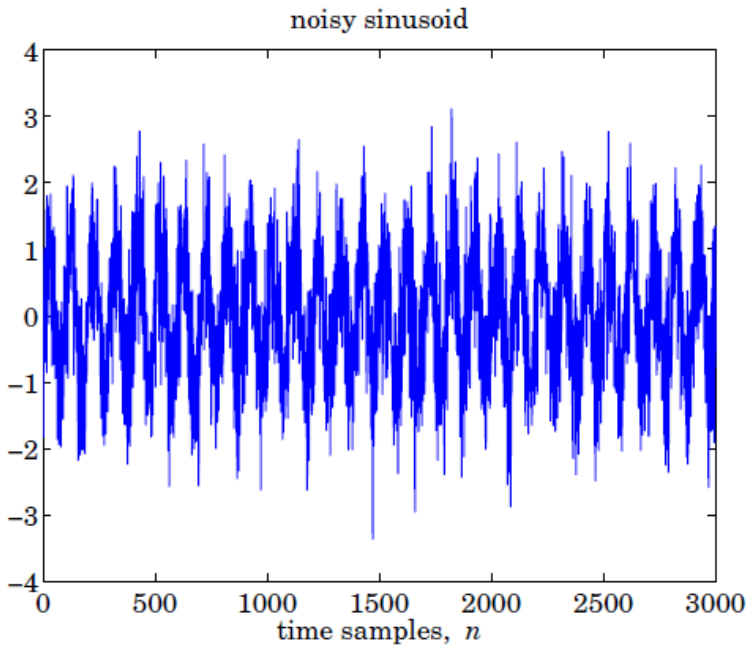
The ALE acts as an adaptive signal separator, provided Δ is properly selected. If Δ is longer than the effective correlation length of the BB component, the delayed replica $\mathrm{BB}(n - \Delta)$ will be entirely uncorrelated with the BB part of the main signal. The adaptive filter will not be able to respond to this component.

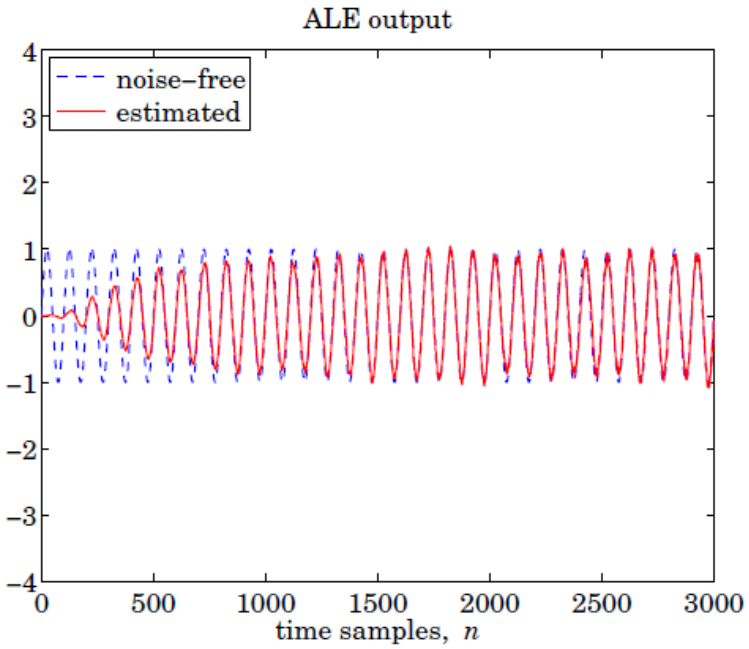
On the other hand, if Δ is shorter than the correlation length of the NB component, the delayed replica $\mathrm{NB}(n - \Delta)$ that appears in the secondary input will still be correlated with the NB part of the main signal, and the filter will respond to cancel it.

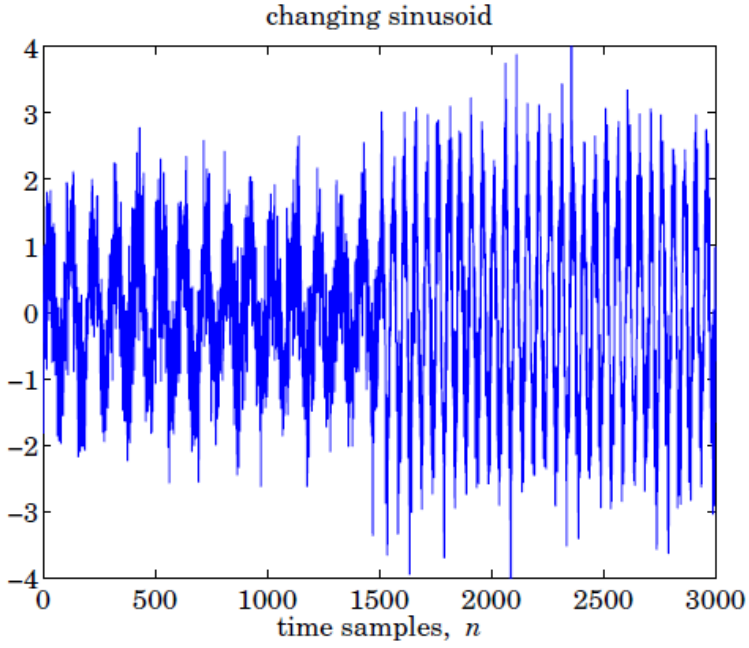
ALE simulation example

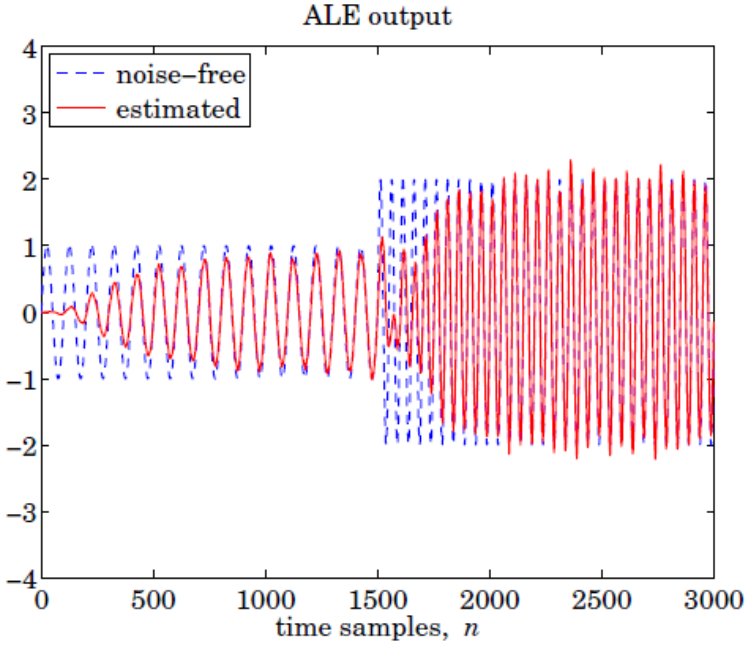
```
SNRdB = 0; SNRa = 10^(SNRdB/10);    % SNR
A = 1; w0 = 0.02*pi;                % amplitude & frequency
L = 3000;
n = 0:L-1;
s = A * sin(w0*n);                  % single sinusoid

% add sinusoid with doubled amplitude and frequency
% n0 = L/2;
% s = A * sin(w0*n).*(n<=n0) + 2*A * sin(2*w0*n).*(n>n0);

seed = 1000;                        % noise
randn('state',seed);
sigv = 1/sqrt(2*SNRa);              % noise var = sigv^2
v = sigv * randn(1,L);              % zero-mean gaussian

x = s + v;                          % noisy sinusoid

D = 10;                             % ALE delay

y = [zeros(1,D), x(1:end-D)];       % delayed input to ALE
```

```
mu =  2e-05;                        % adaptation parameter
M = 200;                            % filter order
h = zeros(M+1,1);                   % initialize weights
w = zeros(M+1,1);                   % initialize delay line

for n = 1:L                         % LMS algorithm
   w(1) = y(n);                     % current input to filter
   xhat(n) = h.'*w;                 % filter output
   e(n) = x(n) - xhat(n);           % estimation error
   h = h + 2*mu*e(n)*w;             % adapt filter weights
   w = [w(1); w(1:end-1)];          % update delay-line vector
end

n = 0:L-1;

figure; plot(n,x,'b-');
title('noisy sinusoid')

figure; plot(n,s,'b--', n,xhat,'r-');
title('ALE output');
```


noisy sinusoid
time samples, n

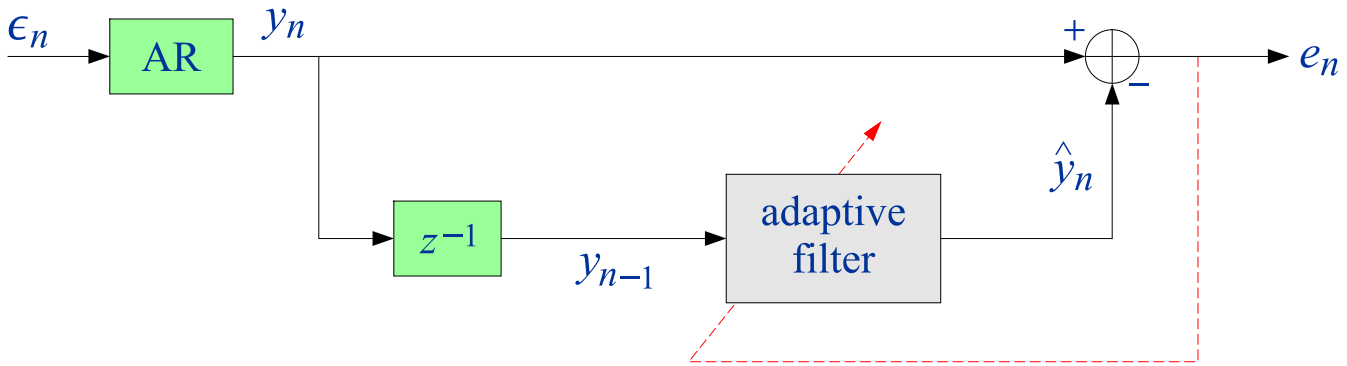

ALE output
noise–free
estimated
time samples, n

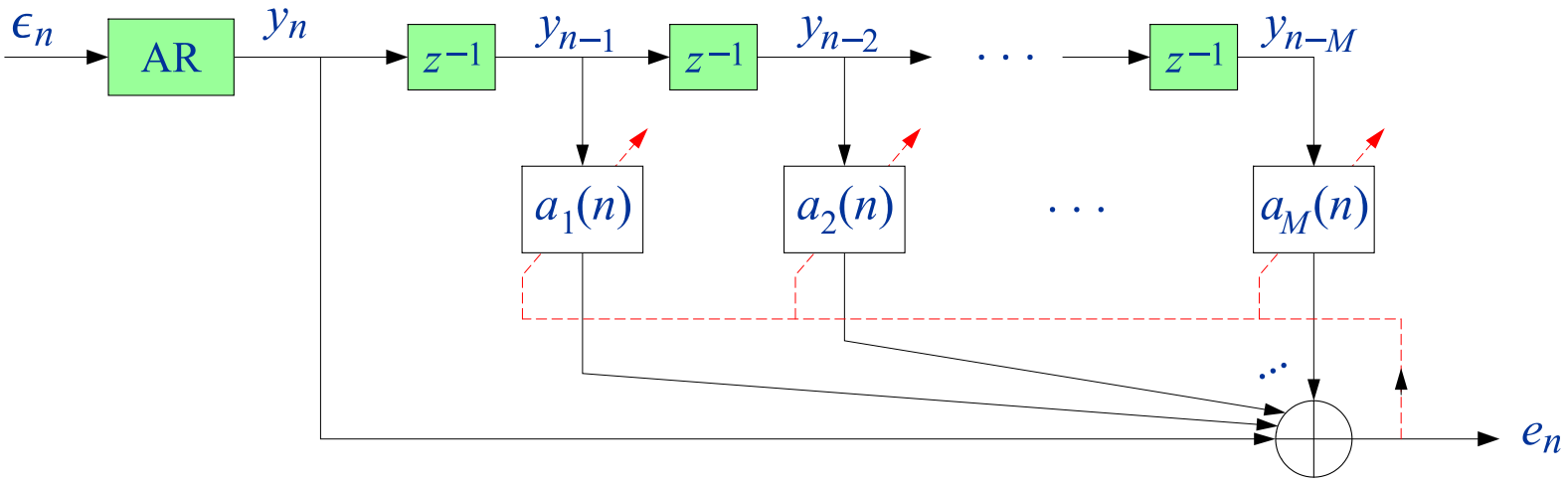

changing sinusoid
time samples, n


ALE output
noise–free
estimated
time samples, n

Adaptive Linear Prediction

A linear predictor is a special case of the ALE with the delay $\Delta = 1$. It is shown below, where we have denoted the main signal by y_n. The secondary signal, which is the input to the adaptive filter, is then y_{n-1}.





Due to the special sign convention used for linear predictors, the LMS adaptation algorithm now reads,

$$\hat{y}_n = -\big[a_1(n)y_{n-1} + a_2(n)y_{n-2} + \cdots + a_M(n)y_{n-M}\big]$$

$$e_n = y_n - \hat{y}_n = y_n + a_1(n)y_{n-1} + \cdots + a_M(n)y_{n-M}$$

$$a_m(n+1) = a_m(n) - 2\mu e_n y_{n-m}\,, \quad m = 1, 2 \ldots, M$$

Because of the importance of the adaptive predictor, we present a direct derivation of the LMS algorithm as it applies to this case. The weights a_m are chosen optimally to minimize the mean output power of the filter, that is, the mean-square prediction error:

$$\mathcal{E} = E[e_n^2] = \mathbf{a}^T R\, \mathbf{a} = \min$$

where $\mathbf{a} = [1, a_1, a_2, \ldots, a_M]^T$ is the prediction error filter.

The performance index $\mathcal{E}$ is minimized with respect to the M weights a_m. The gradient with respect to a_m is the mth component of the vector $2R\mathbf{a}$, namely,

$$\frac{\partial \mathcal{E}}{\partial a_m} = 2(R\,\mathbf{a})_m = 2\big(E[\mathbf{y}(n)\mathbf{y}(n)^T]\,\mathbf{a}\big)_m = 2\big(E[\mathbf{y}(n)\mathbf{y}(n)^T\mathbf{a}]\big)_m$$

$$= 2\big(E[\mathbf{y}(n)e_n]\big)_m = 2E[e_n\,y_{n-m}]$$

The instantaneous gradient is obtained by *ignoring* the expectation instruction, so that the LMS time-update of the mth weight becomes,

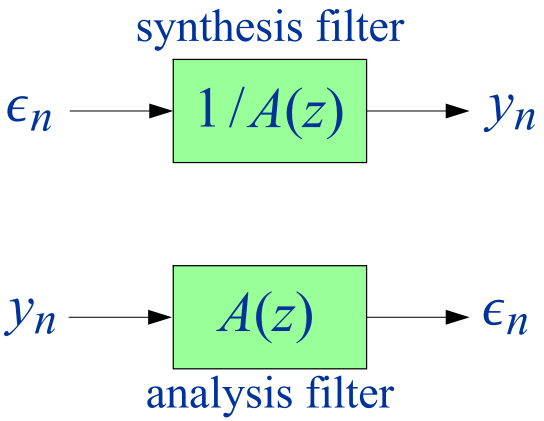
$$\Delta a_m(n) = -\mu\,\frac{\partial \mathcal{E}}{\partial a_m} = -2\mu e_n\,y_{n-m}\,, \quad m = 1, 2, \ldots, M$$

so that the adaptive weights are updated by,

$$a_m(n+1) = a_m(n) + \Delta a_m(n) = a_m(n) - 2\mu e_n y_{n-m}\,, \quad m = 1, 2 \ldots, M$$

Adaptive Spectrum Estimation

The adaptive predictor may be thought of as an *adaptive whitening filter*, or an analysis filter which determines the linear prediction model parameters adaptively. As processing of the signal y_n takes place, the autoregressive model parameters a_m are extracted *on-line*.



$$A(z) = 1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}$$

$$y_n = -\left[a_1 y_{n-1} + a_2 y_{n-2} + \cdots + a_M y_{n-M}\right] + \epsilon_n$$

$$\epsilon_n = y_n + a_1 y_{n-1} + a_2 y_{n-2} + \cdots + a_M y_{n-M}$$

The extracted model parameters may be used in any desired way—for example, to provide the *autoregressive spectrum estimate* of the signal y_n.

One of the advantages of the adaptive implementation is that it offers the possibility of tracking slow changes in the spectra of non-stationary signals. The only requirement for obtaining meaningful spectrum estimates is that the non-stationary changes of the spectrum be slow enough for the adaptive filter to have a chance to converge between changes.

Typical applications are the tracking of sinusoids in noise whose frequencies may be slowly changing, or tracking the time development of the spectra of non-stationary EEG signals.

At each time instant n, the adaptive weights $a_m(n)$, $m = 1, 2, \ldots, M$ may be used to obtain an instantaneous autoregressive estimate of the power spectrum of y_n in the form,

$$S_n(\omega) = \frac{1}{\left|1 + a_1(n)e^{-j\omega} + a_2(n)e^{-2j\omega} + \cdots + a_M(n)e^{-Mj\omega}\right|^2}$$

The same adaptive approach to LP spectrum estimation may also be used in the problem of *angle-of-arrival* estimation, or, *multiple source location* by adaptive sensor arrays.

The only difference in the algorithm is to replace y_{n-m} by $y_m(n)$, that is, by the signal recorded at the mth sensor at time n—and to use the complex-valued version of the LMS algorithm. For completeness, we summarize the computational steps in this case,

$$e(n) = y_0(n) + a_1(n)y_1(n) + a_2(n)y_2(n) + \cdots + a_M(n)y_M(n)$$

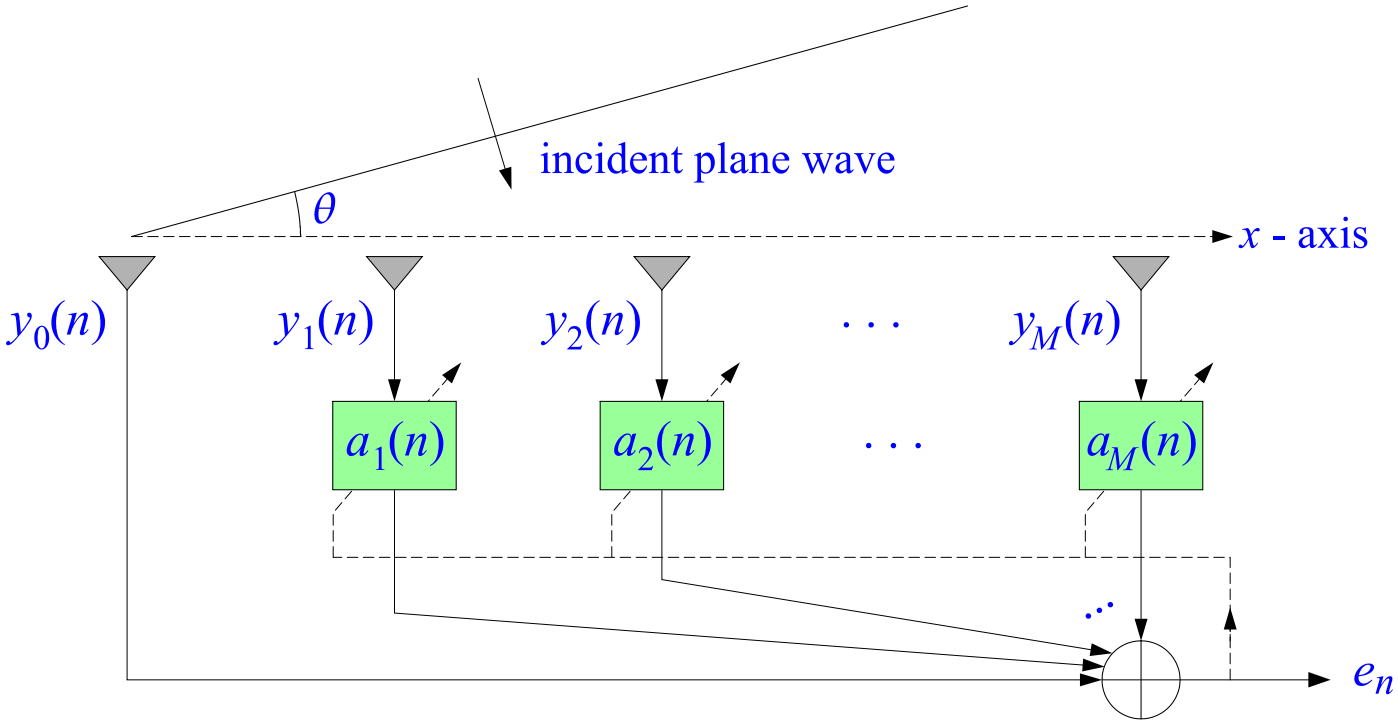
$$a_m(n+1) = a_m(n) - 2\mu e(n)y_m^*(n)\,, \quad m = 1, 2, \ldots, M$$

At each time instant n, the corresponding spatial spectrum estimate may be computed by,

$$S_n(k) = \frac{1}{\left|1 + a_1(n)e^{-jk} + a_2(n)e^{-2jk} + \cdots + a_M(n)e^{-Mjk}\right|^2}$$

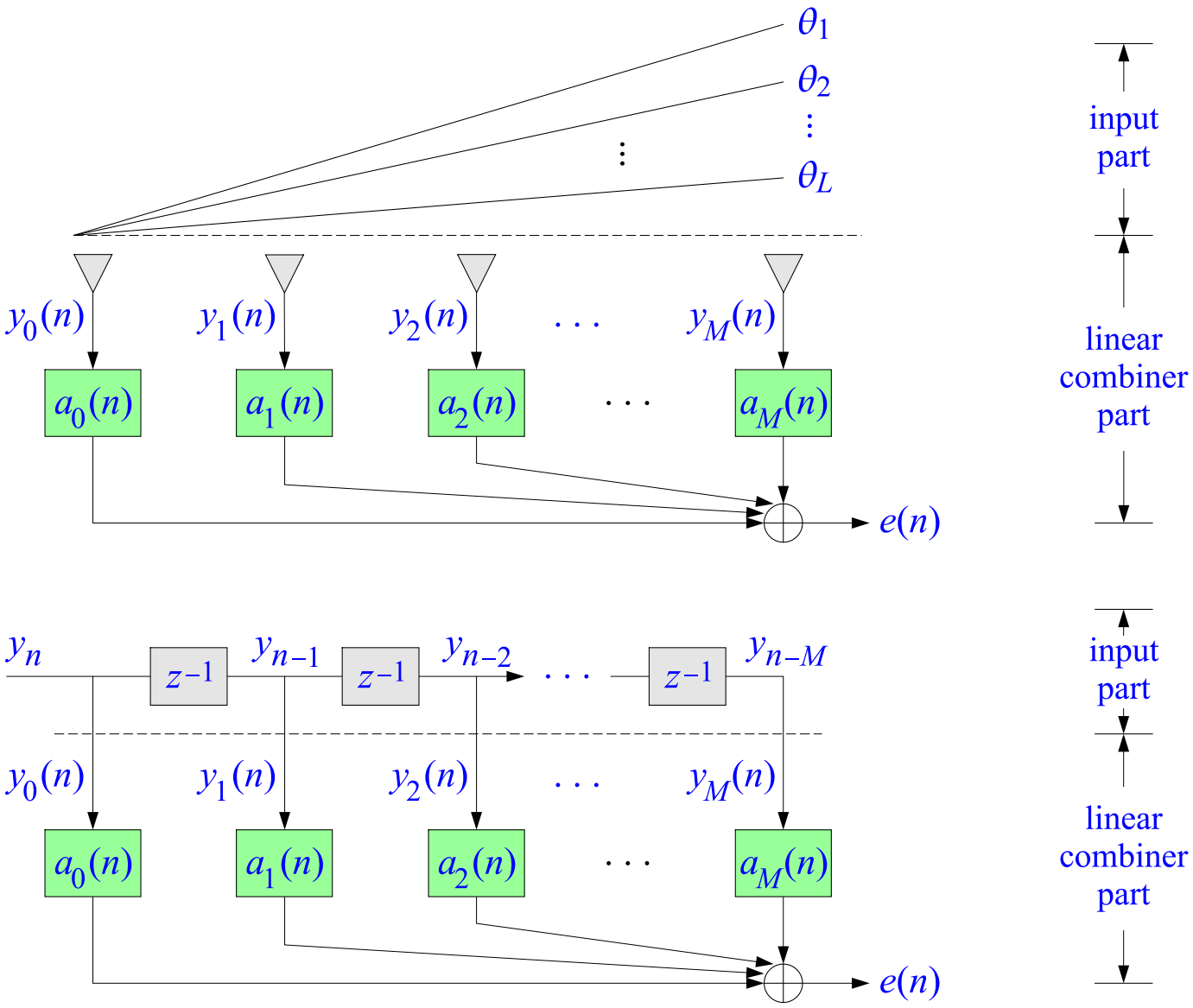
where k denotes the *normalized* wavenumber

$$k = \frac{\omega d}{c}\sin\theta = \frac{2\pi d}{\lambda}\sin\theta\,, \quad \lambda = \text{wavelength}$$



$$k = \frac{\omega d}{c} \sin\theta = \frac{2\pi d}{\lambda} \sin\theta$$

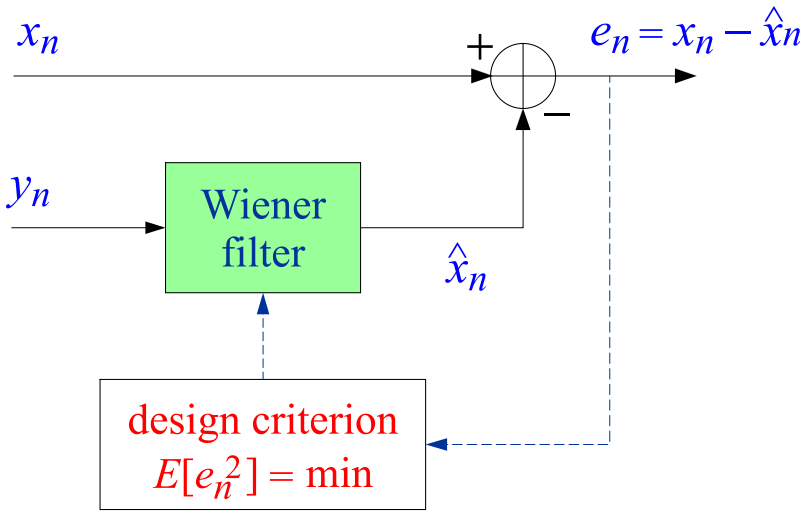
Duality between time-series and array problems



RLS Algorithm

All adaptive and block processing implementations of optimum Wiener filtering problems effectively replace the theoretical performance index $\mathcal{E}$, with a computable one. Examples are:

theoretical:	$\mathcal{E} = E[e_n^2] = \min$
block processing:	$\mathcal{E}_N = \frac{1}{N} \sum_{n=0}^{N-1+M} e_n^2 = \min$, exact
LMS adaptive:	$\mathcal{E}_n = e_n^2 = \min$, gradient-descent
RLS adaptive:	$\mathcal{E}_n = \sum_{k=0}^{n} \lambda^{n-k} e_k^2 = \min$, exact



The LMS adaptation algorithm, based on the steepest descent method, provides a gradual, iterative, minimization of the performance index. The adaptive weights are not optimal at each time instant, but become so only after convergence. By contrast, the RLS adaptation algorithm is based on the *exact minimization* of a least-squares error criterion, so that the filter weights are optimal at each time instant n.

To better track possible non-stationarities in the signals, the performance index includes exponential weighting,

$$\mathcal{E}_n = \sum_{k=0}^{n} \lambda^{n-k} e^2(k) \qquad \text{(RLS performance index)}$$

$$\mathcal{E}_n = e^2(n) + \lambda e^2(n-1) + \lambda^2 e^2(n-2) + \cdots + \lambda^n e^2(0)$$

where the *forgetting factor* λ is positive and less than one. The performance index emphasizes the most recent observations and exponentially ignores the older ones.

Setting the derivative with respect to $\mathbf{h}$ to zero, we find the least-square versions of the *orthogonality equations*,

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{h}} = -2 \sum_{k=0}^{n} \lambda^{n-k} e(k) \mathbf{y}(k) = 0$$

which may be cast in a *normal equation* form,

$$\sum_{k=0}^{n} \lambda^{n-k} \left[x(k) - \mathbf{h}^T \mathbf{y}(k) \right] \mathbf{y}(k) = 0 \,, \quad \text{or,}$$

$$\left[\sum_{k=0}^{n} \lambda^{n-k} \mathbf{y}(k) \mathbf{y}(k)^T \right] \mathbf{h} = \sum_{k=0}^{n} \lambda^{n-k} x(k) \mathbf{y}(k)$$

Define the quantities,

$$\begin{aligned} R(n) &= \sum_{k=0}^{n} \lambda^{n-k} \, \mathbf{y}(k) \, \mathbf{y}(k)^T \\ \mathbf{r}(n) &= \sum_{k=0}^{n} \lambda^{n-k} x(k) \, \mathbf{y}(k) \end{aligned} \tag{2}$$

Then, we may rewrite the normal equations as,

$$R(n)\mathbf{h} = \mathbf{r}(n)$$

with solution, $\mathbf{h} = R(n)^{-1}\mathbf{r}(n)$. Note that the n-dependence of $R(n)$ and $\mathbf{r}(n)$ makes $\mathbf{h}$ depend on n, and we shall write, therefore,

$$\boxed{\mathbf{h}(n) = R(n)^{-1}\mathbf{r}(n) = \text{optimum weights at time } n} \qquad (3)$$

and similarly,

$$\mathbf{h}(n-1) = R(n-1)^{-1}\mathbf{r}(n-1) = \text{optimum weights at time } n{-}1 \qquad (4)$$

These are the least-squares versions of the ordinary Wiener solution, with $R(n)$ and $\mathbf{r}(n)$ playing the role of the covariance matrix, $R = E[\mathbf{y}(n)\mathbf{y}^T(n)]$, and cross-correlation vector, $\mathbf{r} = E[x(n)\mathbf{y}(n)]$.

These quantities satisfy the exponential-movine-average (EMA) updating properties, derivable from (2),

$$\boxed{\begin{aligned} R(n) &= \lambda R(n-1) + \mathbf{y}(n)\mathbf{y}(n)^T \\ \mathbf{r}(n) &= \lambda\,\mathbf{r}(n-1) + x(n)\,\mathbf{y}(n) \end{aligned}} \qquad \begin{aligned} &(5) \\ &(6) \end{aligned}$$

Because $R(n), \mathbf{r}(n)$ satisfy the time recursions (5) and (6), one might expect that $\mathbf{h}(n)$ and $\mathbf{h}(n-1)$ can also be related recursively to each other, resulting in the RLS algorithm.

Assuming tentatively that $R(n)$ and $R(n-1)$ are invertible matrices (an issue to be clarified below), and multiplying Eq. (5) from the left by $R(n)^{-1}$ and from the right by $\lambda^{-1}R(n-1)^{-1}$, we obtain,

$$\lambda^{-1}R(n-1)^{-1} = R(n)^{-1} + R(n)^{-1}\mathbf{y}(n)\,\mathbf{y}(n)^T R(n-1)^{-1}\lambda^{-1} \qquad (7)$$

This leads us to define the so-called a priori and a posteriori "Kalman gains",

$$\boxed{\begin{aligned} \mathbf{k}(n) &= R(n)^{-1}\mathbf{y}(n) = \text{a posteriori Kalman gain} \\ \mathbf{k}(n/n-1) &= \lambda^{-1}R(n-1)^{-1}\mathbf{y}(n) = \text{a priori Kalman gain} \end{aligned}} \qquad (8)$$

as well as the a priori and a posteriori estimates and estimation errors,

$$\begin{aligned} \hat{x}(n) &= \mathbf{h}(n)^T \mathbf{y}(n) = \text{a posteriori estimate of } x(n) \\ e(n) &= x(n) - \hat{x}(n) = \text{a posteriori estimation error} \end{aligned} \tag{9}$$

$$\begin{aligned} \hat{x}(n/n-1) &= \mathbf{h}(n-1)^T \mathbf{y}(n) = \text{a priori estimate of } x(n) \\ e(n/n-1) &= x(n) - \hat{x}(n/n-1) = \text{a priori estimation error} \end{aligned} \tag{10}$$

Using the Kalman gains (8), we can re-express the estimates as follows,

$$\begin{aligned} \hat{x}(n) &= \mathbf{h}(n)^T \mathbf{y}(n) = \mathbf{r}(n)^T R(n)^{-1} \mathbf{y}(n) \\ \hat{x}(n/n-1) &= \mathbf{h}(n-1)^T \mathbf{y}(n) = \mathbf{r}(n-1)^T R(n-1)^{-1} \mathbf{y}(n) \end{aligned}, \quad \text{or,}$$

$$\boxed{\begin{aligned} \hat{x}(n) &= \mathbf{r}(n)^T \mathbf{k}(n) \\ \hat{x}(n/n-1) &= \lambda\, \mathbf{r}(n-1)^T \mathbf{k}(n/n-1) \end{aligned}} \tag{11}$$

We note that $\hat{x}(n)$ is the optimum estimate of $x(n)$ using the optimum weights at time n, whereas $\hat{x}(n/n-1)$ is a suboptimal estimate of $x(n)$, or a tentative estimate of $x(n)$, obtained by using the old optimum weights $\mathbf{h}(n-1)$ instead of the updated ones at time n, but acting on the same observation vector $\mathbf{y}(n)$ at time n.

Let us define also the so-called "likelihood" scalar variables,

$$\nu(n) = \mathbf{k}(n/n-1)^T \mathbf{y}(n) = \lambda^{-1}\mathbf{y}(n)^T R(n-1)^{-1}\mathbf{y}(n)$$
$$\mu(n) = \frac{1}{1+\nu(n)} \quad \Rightarrow \quad \mu(n)\nu(n) = 1-\mu(n) \tag{12}$$

With the definitions (8), we can write (7) in the form,

$$\boxed{\lambda^{-1}R(n-1)^{-1} = R(n)^{-1} + \mathbf{k}(n)\,\mathbf{k}(n/n-1)^T} \tag{13}$$

and if we act by both sides of (13) on $\mathbf{y}(n)$ and use (8) and (12), we find,

$$\lambda^{-1}R(n-1)^{-1}\mathbf{y}(n) = R(n)^{-1}\mathbf{y}(n) + \mathbf{k}(n)\,\mathbf{k}(n/n-1)^T\mathbf{y}(n)\,, \quad \text{or,}$$

$$\mathbf{k}(n/n-1) = \mathbf{k}(n) + \mathbf{k}(n)\,\nu(n) = \big[1+\nu(n)\big]\mathbf{k}(n)$$

or, solving for $\mathbf{k}(n)$, and using (12),

$$\boxed{\mathbf{k}(n) = \mu(n)\,\mathbf{k}(n/n-1)} \tag{14}$$

As a consequence of this, we note also that,

$$\mathbf{y}(n)^T R(n)^{-1}\mathbf{y}(n) = \mathbf{y}(n)^T\mathbf{k}(n) = \mathbf{y}(n)^T\mathbf{k}(n/n-1)\,\mu(n)\,, \quad \text{or,}$$

$$\mathbf{y}(n)^T R(n)^{-1}\mathbf{y}(n) = \mu(n)\nu(n) = 1-\mu(n)$$

Next, we derive a relationship between the a priori and a posteriori estimation errors, using (11) and the recursion (6), and Eq. (14),

$$\begin{aligned}
e(n) &= x(n) - \hat{x}(n) = x(n) - \mathbf{r}(n)^T\mathbf{k}(n) \\
&= x(n) - \big[\lambda\mathbf{r}(n-1) + x(n)\mathbf{y}(n)\big]^T\mu(n)\mathbf{k}(n/n-1) \\
&= x(n) - \lambda\,\mathbf{r}(n-1)^T\mathbf{k}(n/n-1)\mu(n) - x(n)\mu(n)\mathbf{y}(n)^T\mathbf{k}(n/n-1) \\
&= x(n) - \hat{x}(n/n-1)\mu(n) - x(n)\mu(n)\nu(n) \\
&= x(n) - \hat{x}(n/n-1)\mu(n) - \big[1-\mu(n)\big]x(n) \\
&= \mu(n)\big[x(n) - \hat{x}(n/n-1)\big]\,, \quad \text{or,}
\end{aligned}$$

$$\boxed{e(n) = \mu(n)e(n/n-1)} \qquad (15)$$

Next, we obtain the time-updates for the optimum weights, using Eqs. (3) and (4), and the recursions (5) and (6), and starting with,

$$\begin{aligned} R(n)\,\mathbf{h}(n-1) &= \left[\lambda R(n-1) + \mathbf{y}(n)\mathbf{y}(n)^T\right]\mathbf{h}(n-1) \\ &= \lambda R(n-1)\,\mathbf{h}(n-1) + \mathbf{y}(n)\,\mathbf{y}(n)^T\mathbf{h}(n-1) \\ &= \lambda\,\mathbf{r}(n-1) + \mathbf{y}(n)\hat{x}(n/n-1) \\ &= \mathbf{r}(n) - x(n)\,\mathbf{y}(n) + \mathbf{y}(n)\,\hat{x}(n/n-1) \\ &= \mathbf{r}(n) - \left[x(n) - \hat{x}(n/n-1)\right]\mathbf{y}(n) \\ &= \mathbf{r}(n) - e(n/n-1)\,\mathbf{y}(n) \end{aligned}$$

and multiplying both sides by $R(n)^{-1}$,

$$\mathbf{h}(n-1) = R(n)^{-1}\mathbf{r}(n) - e(n/n-1)\,R(n)^{-1}\mathbf{y}(n) = \mathbf{h}(n) - e(n/n-1)\,\mathbf{k}(n)$$

or, solving for $\mathbf{h}(n)$,

$$\boxed{\mathbf{h}(n) = \mathbf{h}(n-1) + e(n/n-1)\,\mathbf{k}(n)} \qquad (16)$$

Putting all the steps together, after defining the matrix inverses,

$$P(n) = R(n)^{-1}\,, \quad P(n-1) = R(n-1)^{-1}$$

and rearranging Eq. (13), we obtain the RLS algorithm,

for each time instant n, do,

$$\mathbf{k}(n/n-1) = \lambda^{-1} P(n-1)\mathbf{y}(n)$$

$$\nu(n) = \mathbf{k}(n/n-1)^T \mathbf{y}(n)\,, \quad \mu(n) = \frac{1}{1+\nu(n)}$$

$$\mathbf{k}(n) = \mu(n)\,\mathbf{k}(n/n-1)$$

$$P(n) = \lambda^{-1} P(n-1) - \mathbf{k}(n)\,\mathbf{k}(n/n-1)^T$$

$$\hat{x}(n/n-1) = \mathbf{h}(n-1)^T \mathbf{y}(n)$$

$$e(n/n-1) = x(n) - \hat{x}(n/n-1)$$

$$e(n) = \mu(n) e(n/n-1)$$

$$\hat{x}(n) = x(n) - e(n)$$

$$\mathbf{h}(n) = \mathbf{h}(n-1) + e(n/n-1)\,\mathbf{k}(n) \qquad (17)$$

```
x = ...                              % read array of primary input x(n)
y = ...                              % read array of secondary input y(n)

w = zeros(M+1,1);                    % initialize delay line
h = zeros(M+1,1);                    % initialize filter
delta = ...                          % e.g., delta = 0.001
lambda = ...                         % e.g., lambda = 0.999
P = eye(M+1)/delta;                  % initialize convariance inverse

% H = [];                            % collect h's for plotting, optional
for n=1:length(x)                    % RLS algorithm
   w(1) = y(n);                      % current input to filter
   k0 = P*w/lambda;                  % a priori Kalman gain vector
   nu = k0'*w;                       % likelihood variable
   mu = 1/(1+nu);                    % likelihood variable
   k1 = mu*k0;                       % a posteriori Kalman gain vector
   P = P/lambda - k1*k0';            % update P
   P = (P+P')/2;                     % symmetrize P, optional
   xhat0 = h'*w;                     % a priori estimate of x
   e0 = x(n) - xhat0;                % a priori estimation error
   e(n) = mu*e0;                     % a posteriori estimation error
   xhat(n) = x(n) - e(n);            % a posteriori estimate of x
   h = h + e0*k1;                    % update h
   w = [w(1); w(1:end-1)];           % update delay-line vector
   % H = [H, h];                     % collect h's for plotting, optional
end
```

We also consider the RLS version of the adaptive predictor,

for each time instant, $n = 0, 1, 2, \ldots,$ do,

$$\mathbf{k}(n/n-1) = \lambda^{-1} P(n-1)\, \tilde{\mathbf{y}}(n)$$

$$\nu_n = \mathbf{k}(n/n-1)^T\, \tilde{\mathbf{y}}(n)\,, \quad \mu_n = \frac{1}{1+\nu_n}$$

$$e(n/n-1) = y(n) + \mathbf{a}^T(n-1)\, \tilde{\mathbf{y}}(n) = \text{a priori error}$$

$$\mathbf{a}(n) = \mathbf{a}(n-1) - \mu_n\, e(n/n-1)\, \mathbf{k}(n/n-1)$$

$$\hat{y}(n/n-1) = -\mathbf{a}^T(n)\, \tilde{\mathbf{y}}(n) = \text{a posteriori prediction}$$

$$P(n) = \lambda^{-1} P(n-1) - \mu_n \mathbf{k}(n/n-1)\, \mathbf{k}(n/n-1)^T$$

$$\tilde{\mathbf{y}}(n+1) = \big[y(n);\ \tilde{\mathbf{y}}(1:\text{end}-1)\big] \tag{19}$$

and initialized at, $P(-1) = \delta^{-1} I_M$, $\mathbf{a}(-1) = 0$, $\tilde{\mathbf{y}}(0) = 0$. The last step expresses (in MATLAB notation) the delay updating operation that is necessary for the next iteration of the loop,

$$\mathbf{a}(n) = \begin{bmatrix} a_1(n) \\ a_2(n) \\ \vdots \\ a_M(n) \end{bmatrix}, \quad \tilde{\mathbf{y}}(n) = \begin{bmatrix} y_{n-1} \\ y_{n-2} \\ \vdots \\ y_{n-M} \end{bmatrix} \quad \Rightarrow \quad \tilde{\mathbf{y}}(n+1) = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-M+1} \end{bmatrix}$$

RLS adaptive predictor

```
y = ...                          % read array of input data y(n)

M = ...                          % e.g., M = 10
delta = ...                      % e.g., delta = 1e-4
lambda = ...                     % e.g., lambda = 0.999

w = zeros(M,1);                  % initialize delay line
a = zeros(M,1);                  % initialize prediction coefficients
P = eye(M)/delta;                % initialize covariance inverse

for n=1:length(y)
   k0 = P*w/la;                  % a priori Kalman gain
   nu = k0'*w;                   % likelihood variable
   mu = 1/(1+nu);                % likelihood variable
   e0 = y(n) + a'*w;             % a priori prediction error
   k1 = mu*k0;                   % a posteriori Kalman gain
   a = a - e0*k1;                % update prediction coefficients
   ypred(n) = -a'*w;             % a posteriori prediction of y(n)
   P = P/la - k1*k0';            % update covariance inverse P
   P = (P+P')/2;                 % symmetrize P, optional
   w = [y(n); w(1:end-1)];       % update delay line
end
```

Neural Networks – Overview

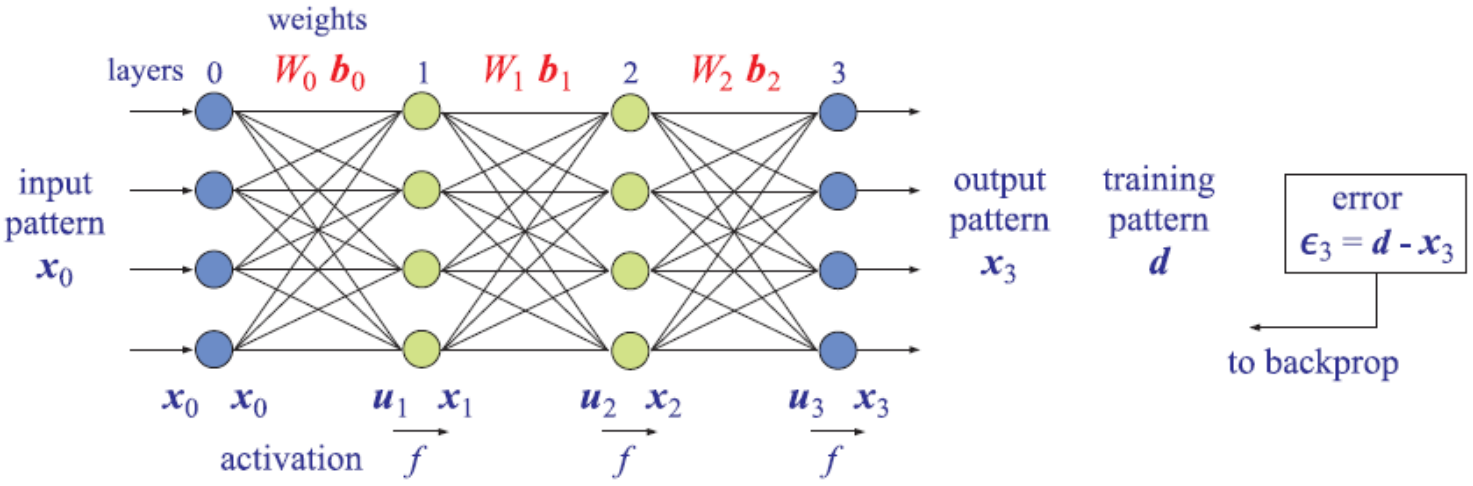
Neural networks are generalizations of adaptive processing systems that involve both linear and non-linear operations across multiple stages (hidden layers). They can be adapted by the LMS algorithm and its implementation via backpropagation. Some useful references are:

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in D. E. Rumelhart and J. L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, Cambridge, MA: MIT Press, 1986.

[2] M. T. Hagan, et al, *Neural Network Design*, available freely from, `http://hagan.okstate.edu/nnd.html`

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016, available from, `http://www.deeplearningbook.org`.

A typical neural network is depicted below. For clarity only two hidden layers are shown, but any additional layers can be added as necessary, as in deep learning networks.



Starting with an applied input pattern $\mathbf{x}_0$, the output pattern $\mathbf{x}_3$ is computed by the following sequence of operations defined by the *connection* matrices W_r and *bias* weights $\mathbf{b}_r$, $r = 0, 1, 2$, and activation function $f(u)$,

forward pass
$\mathbf{u}_1 = W_0\mathbf{x}_0 + \mathbf{b}_0$
$\mathbf{x}_1 = f(\mathbf{u}_1)$
$\mathbf{u}_2 = W_1\mathbf{x}_1 + \mathbf{b}_1$
$\mathbf{x}_2 = f(\mathbf{u}_2)$
$\mathbf{u}_3 = W_2\mathbf{x}_2 + \mathbf{b}_2$
$\mathbf{x}_3 = f(\mathbf{u}_3)$

(20)

last step, $\mathbf{x}_3 = f(\mathbf{u}_3)$, is optional →

The vector dimensions of the layers are,

$$\begin{array}{lll} M_0 \times 1, & \text{input layer}, & \mathbf{x}_0 \\ M_1 \times 1, & \text{hidden layer 1}, & \mathbf{u}_1,\ \mathbf{x}_1 = f(\mathbf{u}_1) \\ M_2 \times 1, & \text{hidden layer 2}, & \mathbf{u}_2,\ \mathbf{x}_2 = f(\mathbf{u}_2) \\ M_3 \times 1, & \text{output layer}, & \mathbf{u}_3,\ \mathbf{x}_3 = f(\mathbf{u}_3) \end{array}$$

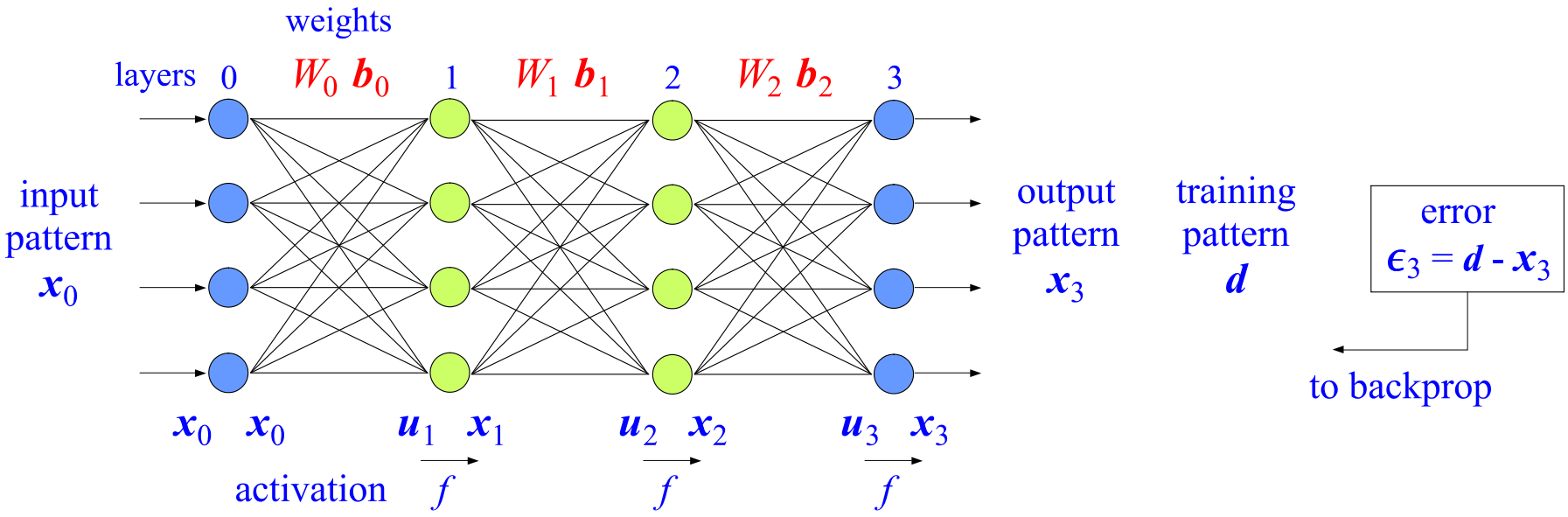
so that the dimensions of the connection matrices and bias weights will be,

$$\begin{array}{llllll} W_0, & M_1 \times M_0 & \text{and} & \mathbf{b}_0\,, & M_1 \times 1 \\ W_1, & M_2 \times M_1 & \text{and} & \mathbf{b}_1\,, & M_2 \times 1 \\ W_2, & M_3 \times M_2 & \text{and} & \mathbf{b}_2\,, & M_3 \times 1 \end{array}$$

and the activation functions are assumed to operate element-wise, that is, if u_i is the i-th component of a vector $\mathbf{u}$, then, the i-th component of the vector $f(\mathbf{u})$ is $f(u_i)$.

During the training phase, a set of training input/output patterns, $\{\mathbf{x}_0, \mathbf{d}\}$, are applied to the network, and the connection matrices and bias weights are adapted to minimize the square deviations of the actual outputs $\mathbf{x}_3$ from the desired outputs $\mathbf{d}$, that is, minimizing the performance index,

$$J = \sum_{\text{patterns}} \epsilon_3^T \epsilon_3 = \sum_{\text{patterns}} (\mathbf{d} - \mathbf{x}_3)^T (\mathbf{d} - \mathbf{x}_3) = \min \qquad (21)$$



Several types of activation functions $f(u)$ have been used, some of which are listed below together with their derivatives $f'(u)$, with the most common being the sigmoid logistic function, but the rectified linear unit (ReLU) is also quite popular,

logistic:	$f(u) = \dfrac{1}{1+e^{-u}}$
symmetric logistic:	$f(u) = \tanh\left(\dfrac{u}{2}\right)$
softplus:	$f(u) = \ln(1+e^{u})$
linear:	$f(u) = u$
rectified linear:	$f(u) = uU(u)$
sinusoid:	$f(u) = \sin u$

logistic:	$f'(u) = f(u)\big[1 - f(u)\big]$
symmetric logistic:	$f'(u) = \frac{1}{2}\big[1 - f^2(u)\big]$
softplus:	$f'(u) = \frac{1}{1 + e^{-u}}$
linear:	$f'(u) = 1$
rectified linear:	$f'(u) = U(u) =$ unit-step function
sinusoid:	$f'(u) = \cos u$

The adaptation of the weights is usually implemented with the gradient-descent, Widrow-Hoff LMS, algorithm, that is, the weight updates are computed by the following rule with a small positive adaptation constant μ, where W_{rij} denotes the ij matrix element of the r-th connection matrix W_r, and b_{ri} denotes the i-th component of the bias weight $\mathbf{b}_r$,

$$\boxed{\Delta W_{rij} = -\mu \frac{\partial J}{\partial W_{rij}}\,, \quad \Delta b_{ri} = -\mu \frac{\partial J}{\partial b_{ri}}}, \quad r = 0, 1, 2 \tag{22}$$

The updated weights are then,

$$\begin{aligned} W_r &= W_r + \Delta W_r\,, \quad r = 0, 1, 2 \\ \mathbf{b}_r &= \mathbf{b}_r + \Delta \mathbf{b}_r\,, \quad r = 0, 1, 2 \end{aligned} \tag{23}$$

The updates can be applied:

(i) either on a pattern-basis, that is, J arises only from one pattern and the updates are applied for that pattern, and the whole process is repeated over all the patterns till convergence,

(ii) or on an epoch-basis, that is, a whole series of input/output patterns is applied and the corrections $\Delta W_{rij}, \Delta b_{ri}$ are accumulated over all the patterns in the epoch before the updated weights are computed, and then, the whole epoch is repeated till convergence.

The *backpropagation algorithm* [1] is a convenient way of calculating the updates (22) based on each pattern $\{\mathbf{x}_0, \mathbf{d}\}$, and makes use of the following gradients of the performance index,

$$\mathbf{e}_r = -\frac{\partial J}{\partial \mathbf{u}_r}, \quad r = 1, 2, 3 \tag{24}$$

or, component-wise,

$$e_{ri} = -\frac{\partial J}{\partial u_{ri}}$$

To see how this works, start with the updates of the last connection weights, $W_2, \mathbf{b}_2$. Given an input pattern, $\mathbf{x}_0$, then after the forward pass of Eq. (20), we have computed all layer signals, $\mathbf{u}_r$, $\mathbf{x}_r$, $r = 1, 2, 3$. Because J depends on $W_2, \mathbf{b}_2$ only through the variable $\mathbf{u}_3$, we have for the partial derivatives of the updates,

$$\begin{aligned} \Delta W_{2ij} &= -\mu \frac{\partial J}{\partial W_{2ij}} = -\mu \frac{\partial J}{\partial u_{3i}} \frac{\partial u_{3i}}{\partial W_{2ij}} = \mu e_{3i} x_{2j} \\ \Delta b_{2i} &= -\mu \frac{\partial J}{\partial b_{2i}} = -\mu \frac{\partial J}{\partial u_{3i}} \frac{\partial u_{3i}}{\partial b_{2i}} = \mu e_{3i} \end{aligned} \tag{25}$$

where we used the definitions (24) and the partial derivatives of the connection formula,

$$\mathbf{u}_3 = W_2 \mathbf{x}_2 + \mathbf{b}_2 \quad \Rightarrow \quad u_{3i} = \sum_j W_{2ij} x_{2j} + b_{2i}$$

$$\frac{\partial u_{3i}}{\partial W_{2ij}} = x_{2j} \,, \quad \frac{\partial u_{3i}}{\partial b_{2i}} = 1$$

Eqs. (25) can be written in the compact matrix forms,

$$\Delta W_2 = \mu\, \mathbf{e}_3 \mathbf{x}_2^T\,, \quad \Delta \mathbf{b}_2 = \mu\, \mathbf{e}_3 \tag{26}$$

The other updates can similarly be expressed in terms of the gradients (24). In summary, we have,

$$\begin{aligned} \Delta W_2 &= \mu\, \mathbf{e}_3 \mathbf{x}_2^T\,, \quad \Delta \mathbf{b}_2 = \mu\, \mathbf{e}_3 \\ \Delta W_1 &= \mu\, \mathbf{e}_2 \mathbf{x}_1^T\,, \quad \Delta \mathbf{b}_1 = \mu\, \mathbf{e}_2 \\ \Delta W_0 &= \mu\, \mathbf{e}_1 \mathbf{x}_0^T\,, \quad \Delta \mathbf{b}_0 = \mu\, \mathbf{e}_1 \end{aligned} \tag{27}$$

The backpropagation algorithm efficiently calculates the quantities, $\mathbf{e}_3, \mathbf{e}_2, \mathbf{e}_1$, starting from the output layer and proceeding backwards to the input layer.

The operations involve the diagonal matrix of the derivatives of the activation function, defined as follows,

$$D(\mathbf{u}) = \text{diag}\big[f'(\mathbf{u})\big]\,, \quad \text{or, element-wise}\,, \quad D_{ij}(\mathbf{u}) = \delta_{ij}\, f'(u_i) \tag{28}$$

Given the output $\mathbf{x}_3$ of the network corresponding to the input pattern $\mathbf{x}_0$, the error relative to the training pattern, $\mathbf{d}$, will be, $\boldsymbol{\epsilon}_3 = \mathbf{d} - \mathbf{x}_3$. Its contribution to the performance index J of Eq. (21), will be, $J_{\text{patt}} = (\mathbf{d} - \mathbf{x}_3)^T(\mathbf{d} - \mathbf{x}_3) = \boldsymbol{\epsilon}_3^T \boldsymbol{\epsilon}_3$. Starting with $\mathbf{e}_3$, we have component-wise,

$$e_{3i} = -\frac{\partial J_{\text{patt}}}{\partial u_{3i}} = -\frac{\partial x_{3i}}{\partial u_{3i}} \frac{\partial J_{\text{patt}}}{\partial x_{3i}} = f'(u_{3i})(d_i - x_{3i}) \,, \quad \text{or,}$$

$$\mathbf{e}_3 = D(\mathbf{u}_3)(\mathbf{d} - \mathbf{x}_3) = D(\mathbf{u}_3)\boldsymbol{\epsilon}_3$$

Next, for $\mathbf{e}_2$, because J_{patt} depends on $\mathbf{x}_2$ through $\mathbf{u}_3$, and for $\mathbf{e}_1$, because J_{patt} depends on $\mathbf{x}_1$ through $\mathbf{u}_2$, we have,

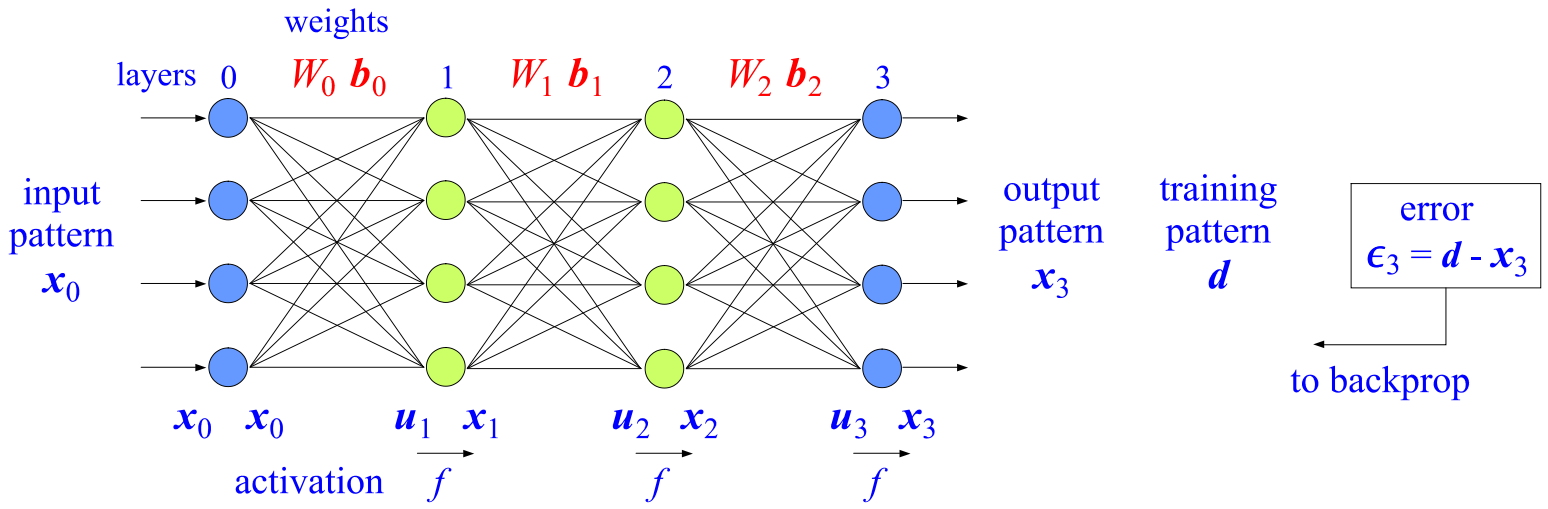
$$\begin{aligned} e_{2i} &= -\frac{\partial J_{\text{patt}}}{\partial u_{2i}} = -\frac{\partial x_{2i}}{\partial u_{2i}} \frac{\partial J_{\text{patt}}}{\partial x_{2i}} = -D(u_{2i}) \frac{\partial J_{\text{patt}}}{\partial x_{2i}} \\ &= -D(u_{2i}) \sum_j \frac{\partial J_{\text{patt}}}{\partial u_{3j}} \frac{\partial u_{3j}}{\partial x_{2i}} = D(u_{2i}) \sum_j W_{2ji} e_{3j} \\ e_{1i} &= -\frac{\partial J_{\text{patt}}}{\partial u_{1i}} = -\frac{\partial x_{1i}}{\partial u_{1i}} \frac{\partial J_{\text{patt}}}{\partial x_{1i}} = -D(u_{1i}) \frac{\partial J_{\text{patt}}}{\partial x_{1i}} \\ &= -D(u_{1i}) \sum_j \frac{\partial J_{\text{patt}}}{\partial u_{2j}} \frac{\partial u_{2j}}{\partial x_{1i}} = D(u_{1i}) \sum_j W_{1ji} e_{2j} \end{aligned}$$

These can be written in matrix forms using the transposed matrices,

$$\begin{aligned} \mathbf{e}_2 &= D(\mathbf{u}_2) W_2^T \mathbf{e}_3 \\ \mathbf{e}_1 &= D(\mathbf{u}_1) W_1^T \mathbf{e}_2 \end{aligned} \tag{29}$$

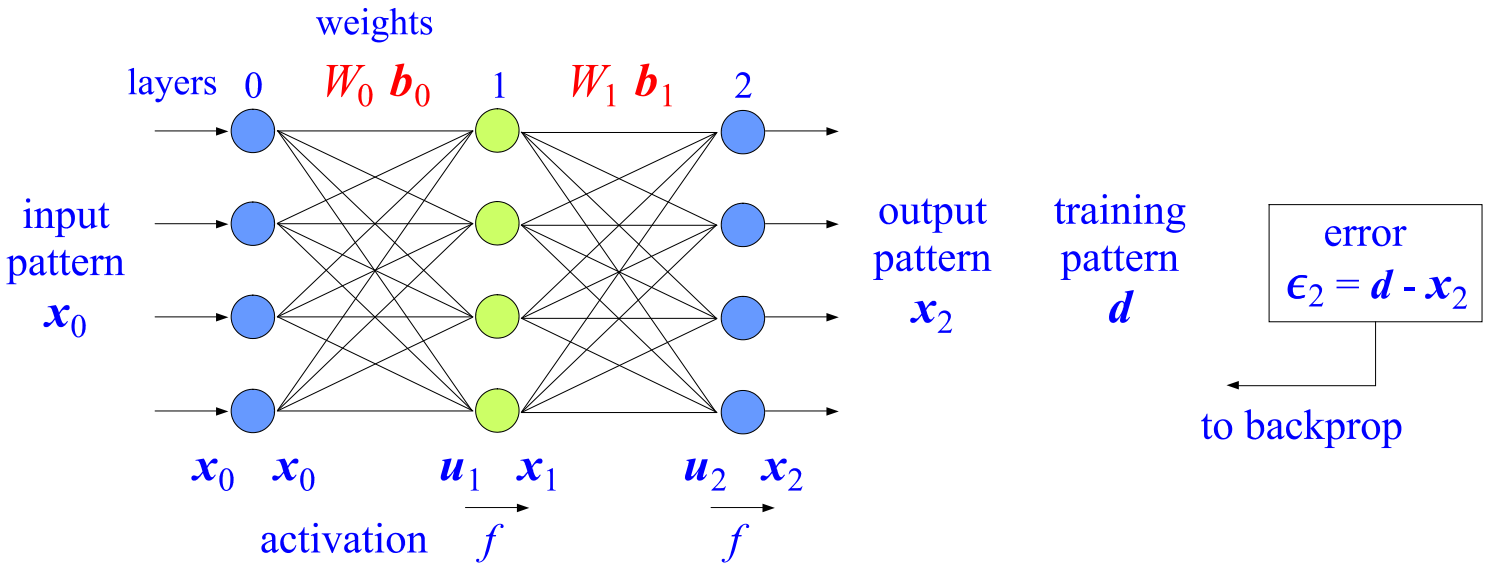
Finally, putting all the computational steps together, we have for the two-hidden-layer network,

forward pass	$\Rightarrow$	backpropagation	$\Rightarrow$	weight updates
$\mathbf{u}_1 = W_0 \mathbf{x}_0 + \mathbf{b}_0$		$\epsilon_3 = \mathbf{d} - \mathbf{x}_3$		$\Delta W_2 = \mu \mathbf{e}_3 \mathbf{x}_2^T$
$\mathbf{x}_1 = f(\mathbf{u}_1)$		$\mathbf{e}_3 = D(\mathbf{u}_3)\epsilon_3$		$\Delta \mathbf{b}_2 = \mu \mathbf{e}_3$
$\mathbf{u}_2 = W_1 \mathbf{x}_1 + \mathbf{b}_1$		$\epsilon_2 = W_2^T \mathbf{e}_3$		$\Delta W_1 = \mu \mathbf{e}_2 \mathbf{x}_1^T$
$\mathbf{x}_2 = f(\mathbf{u}_2)$		$\mathbf{e}_2 = D(\mathbf{u}_2)\epsilon_2$		$\Delta \mathbf{b}_1 = \mu \mathbf{e}_2$
$\mathbf{u}_3 = W_2 \mathbf{x}_2 + \mathbf{b}_2$		$\epsilon_1 = W_1^T \mathbf{e}_2$		$\Delta W_0 = \mu \mathbf{e}_1 \mathbf{x}_0^T$
$\mathbf{x}_3 = f(\mathbf{u}_3)$		$\mathbf{e}_1 = D(\mathbf{u}_1)\epsilon_1$		$\Delta \mathbf{b}_0 = \mu \mathbf{e}_1$



For a single-hidden-layer network depicted below, we have the simpler version,

forward pass	backpropagation	weight updates
$\mathbf{u}_1 = W_0\mathbf{x}_0 + \mathbf{b}_0$	$\boldsymbol{\epsilon}_2 = \mathbf{d} - \mathbf{x}_2$	$\Delta W_1 = \mu \mathbf{e}_2 \mathbf{x}_1^T$
$\mathbf{x}_1 = f(\mathbf{u}_1)$	$\mathbf{e}_2 = D(\mathbf{u}_2)\boldsymbol{\epsilon}_2$	$\Delta \mathbf{b}_1 = \mu \mathbf{e}_2$
$\mathbf{u}_2 = W_1\mathbf{x}_1 + \mathbf{b}_1$	$\boldsymbol{\epsilon}_1 = W_1^T\mathbf{e}_2$	$\Delta W_0 = \mu \mathbf{e}_1 \mathbf{x}_0^T$
$\mathbf{x}_2 = f(\mathbf{u}_2)$	$\mathbf{e}_1 = D(\mathbf{u}_1)\boldsymbol{\epsilon}_1$	$\Delta \mathbf{b}_0 = \mu \mathbf{e}_1$



In practice, the weight updates, ΔW_r, $\Delta \mathbf{b}_r$, are smoothed using a simple EMA with a forgetting factor λ (referred to as "momentum" updating) before the new weights are computed, that is, instead of using the corrections of Eq. (27),

$$\Delta W_r = \mu\, \mathbf{e}_{r+1} \mathbf{x}_r^T\,, \quad r = 0, 1, 2$$
$$\Delta \mathbf{b}_r = \mu\, \mathbf{e}_{r+1}$$

we use their EMA-smoothed versions,

$$\Delta W_r = \lambda \Delta W_r + \mu\, \mathbf{e}_{r+1} \mathbf{x}_r^T\,, \quad r = 0, 1, 2$$
$$\Delta \mathbf{b}_r = \lambda \Delta \mathbf{b}_r + \mu\, \mathbf{e}_{r+1}$$

Depending on whether one uses pattern-updating or epoch-updating, the iterative computational algorithm for minimizing the performace index J may be summarized as follows.

for each epoch, do,

  for each pattern, do,

    forward pass, calculate, $\mathbf{x}_r$

    backpropagation pass, calculate, $\mathbf{e}_{r+1}$

    $\Delta W_r = \lambda \Delta W_r + \mu \mathbf{e}_{r+1} \mathbf{x}_r^T , \quad r = 0, 1, 2$

    $\Delta \mathbf{b}_r = \lambda \Delta \mathbf{b}_r + \mu \mathbf{e}_{r+1}$

    if pattern-updating, update now

      $W_r = W_r + \Delta W_r , \quad r = 0, 1, 2$

      $\mathbf{b}_r = \mathbf{b}_r + \Delta \mathbf{b}_r$

    end

  end pattern loop

  if epoch-updating, update after all epoch patterns

    $W_r = W_r + \Delta W_r , \quad r = 0, 1, 2$

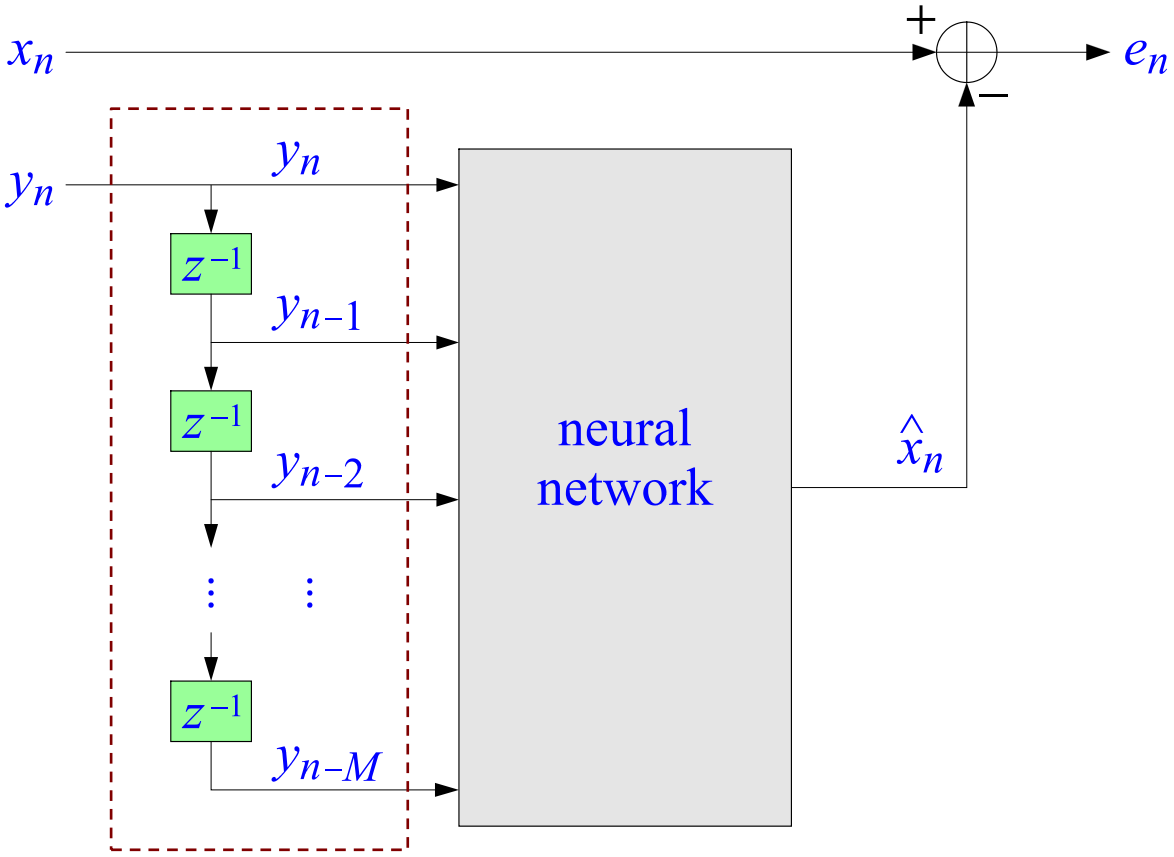
    $\mathbf{b}_r = \mathbf{b}_r + \Delta \mathbf{b}_r$

  end

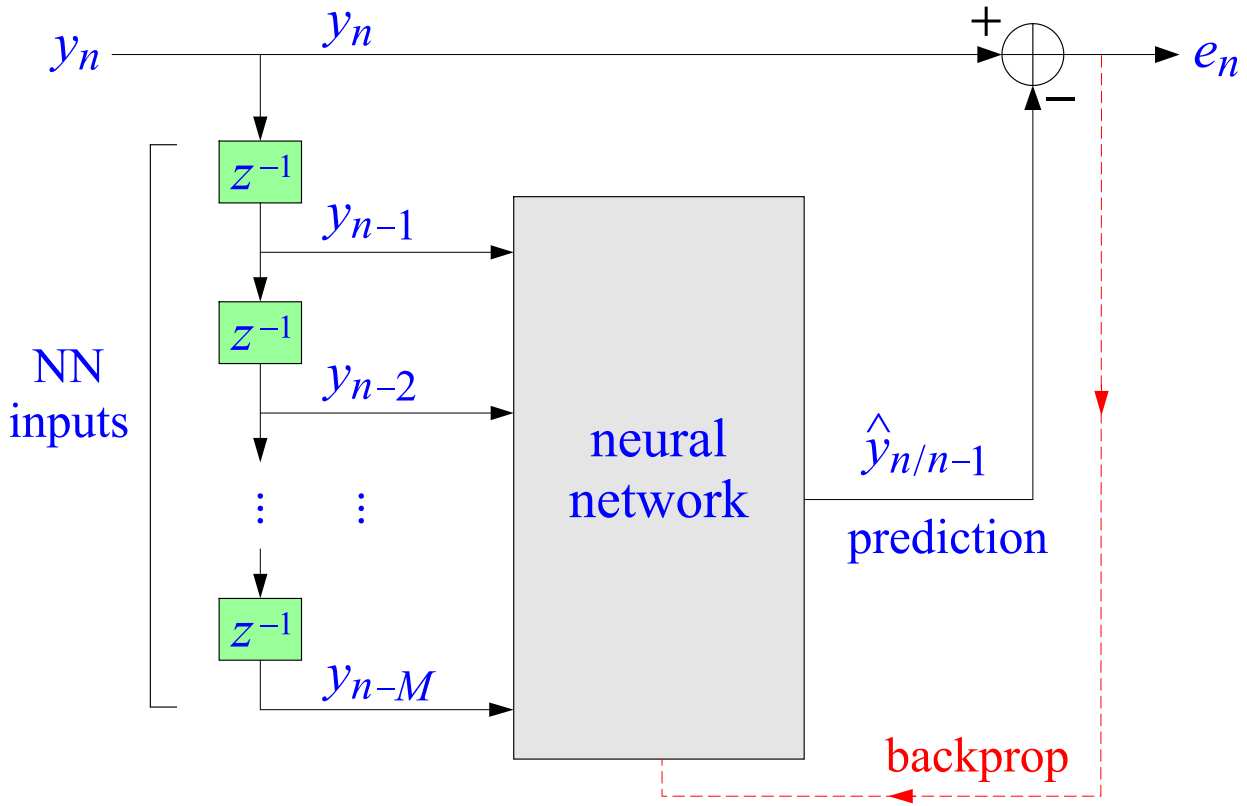
end epoch loop

In the pattern-updating case, the smoothed weight corrections are applied to update the weights after each pattern presentation, whereas in the epoch-updating case, the weight corrections are accumulated over the patterns and applied to update the weights after all patterns have been presented. The epoch loop is then repeated several times (typically, thousands of times) until J has been minimized.

Neural Networks for Estimation and Prediction



Neural Networks for Estimation and Prediction

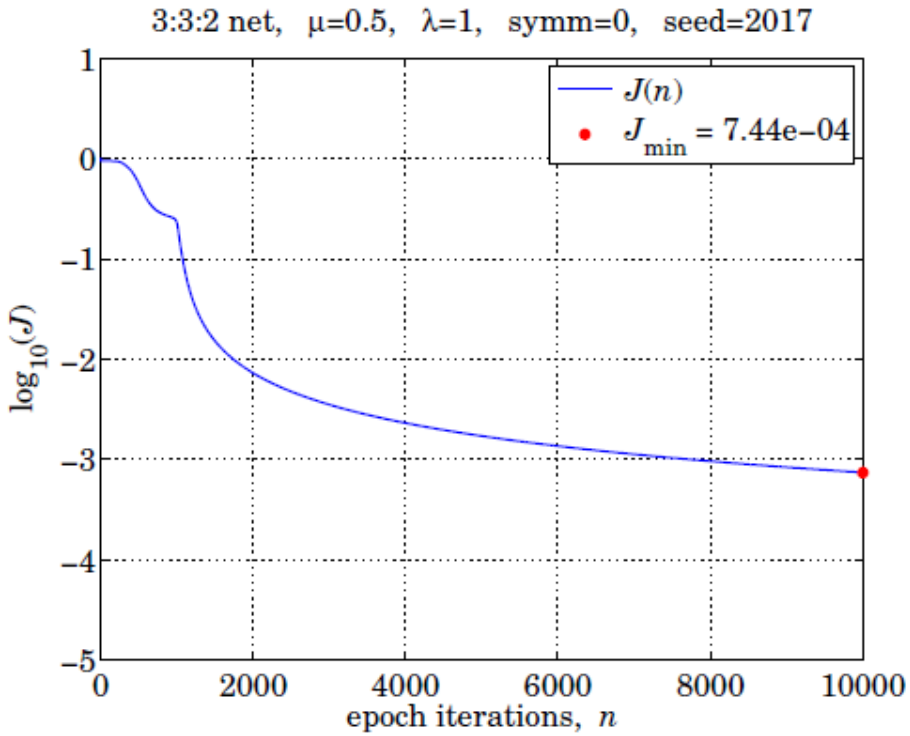


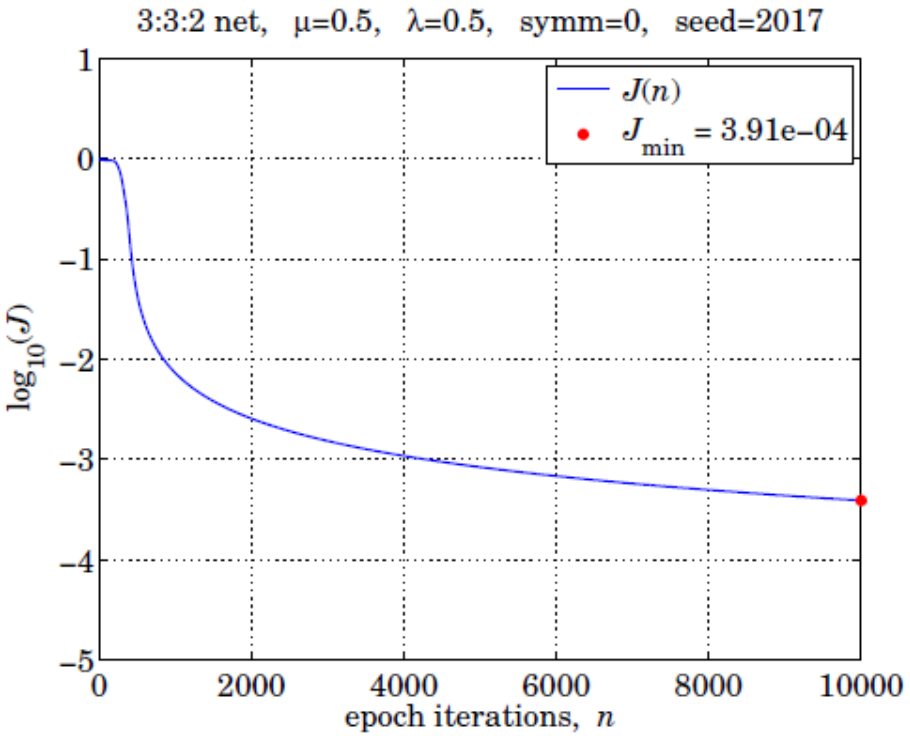
NN Experiments

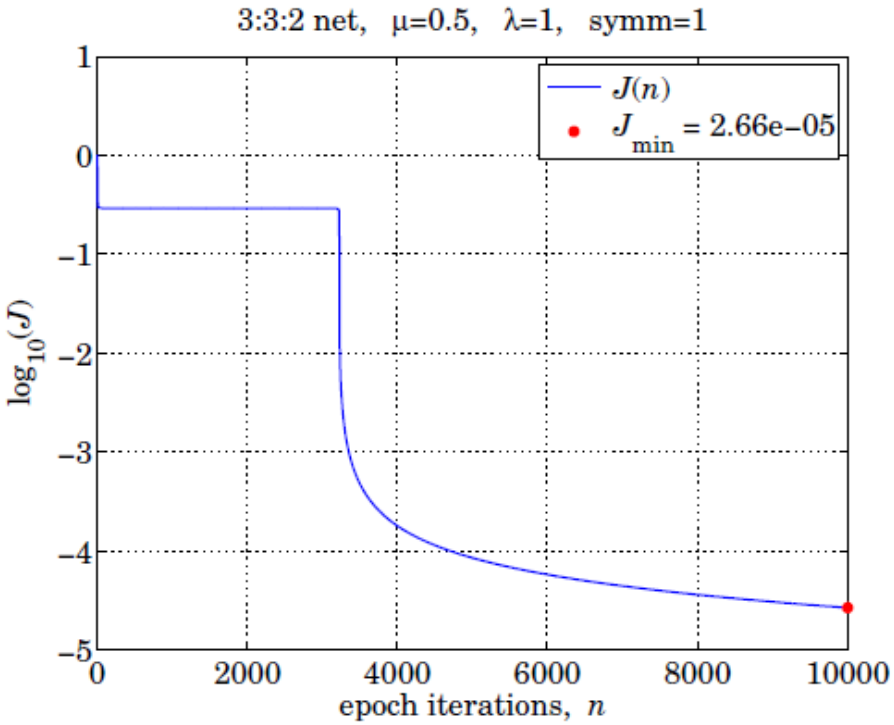
XOR problem with 3:3:2 network

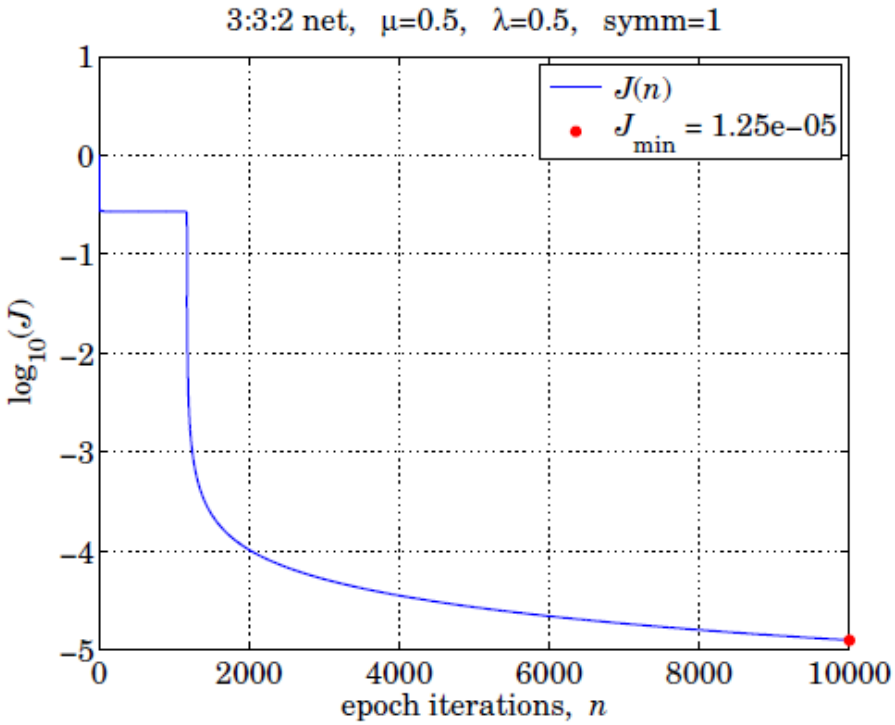
```
%      input          | XOR output
% ------------------------------
%  0     0     0     |   0     1
%  0     0     1     |   1     0
%  0     1     0     |   1     0
%  0     1     1     |   0     1
%  1     0     0     |   1     0
%  1     0     1     |   0     1
%  1     1     0     |   0     1
%  1     1     1     |   1     0
```

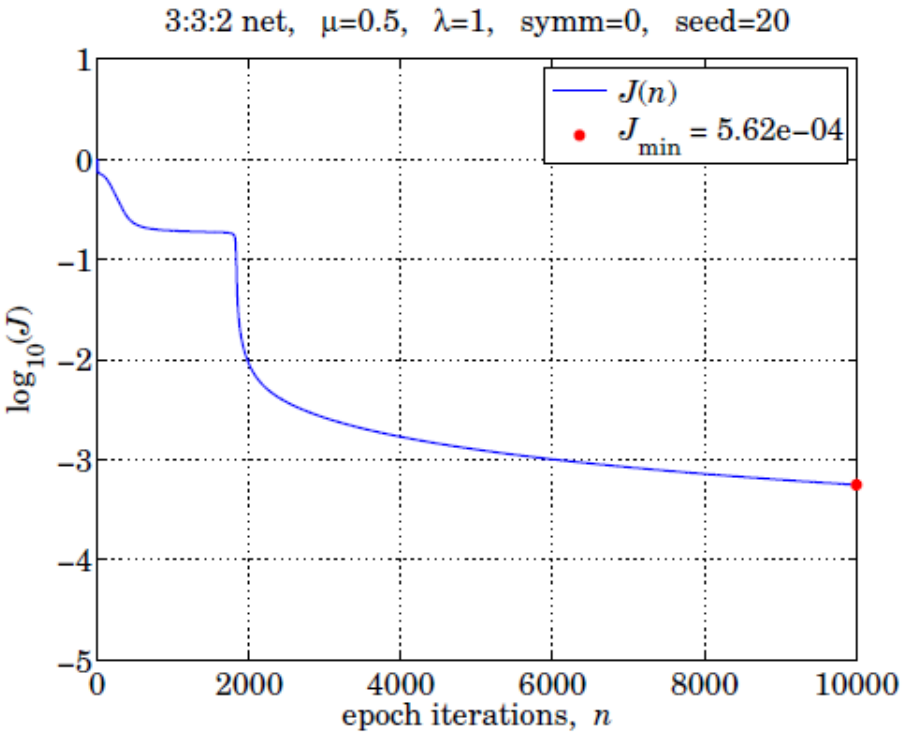
MATLAB code in, **s21nnexp.pdf**

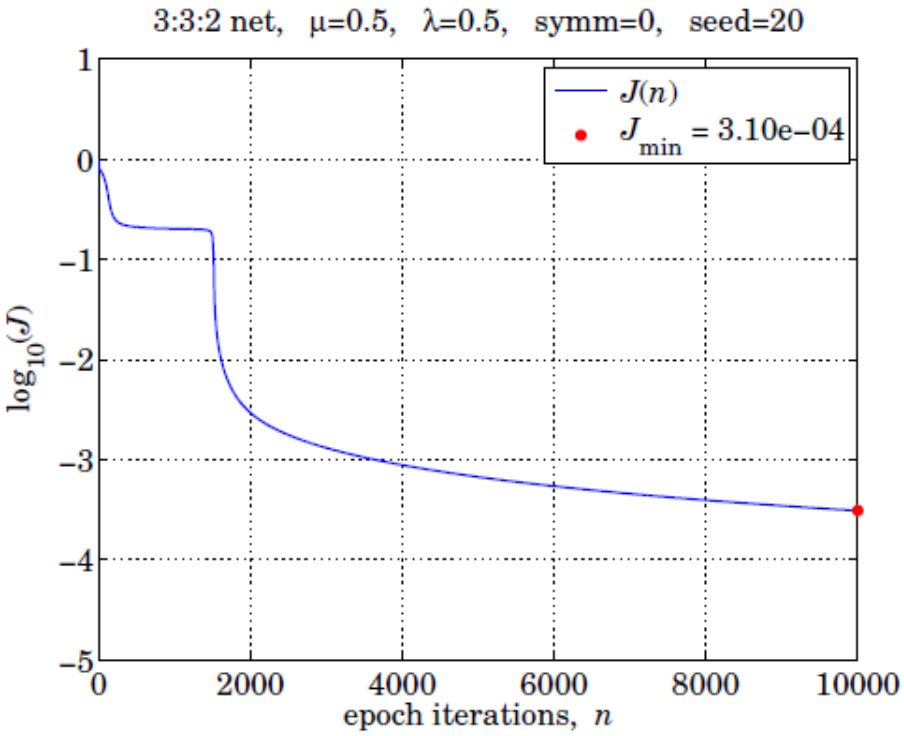

3:3:2 net, μ=0.5, λ=1, symm=0, seed=2017
J(n)
J min = 7.44e−04
log10(J)
epoch iterations, n
1
0
−1
−2
−3
−4
−5
0
2000
4000
6000
8000
10000


3:3:2 net, μ=0.5, λ=0.5, symm=0, seed=2017
J(n)
J min = 3.91e−04
log10(J)
epoch iterations, n
1
0
−1
−2
−3
−4
−5
0
2000
4000
6000
8000
10000


3:3:2 net, μ=0.5, λ=1, symm=1
J(n)
J min = 2.66e−05
log10(J)
epoch iterations, n
1
0
−1
−2
−3
−4
−5
0
2000
4000
6000
8000
10000


3:3:2 net, μ=0.5, λ=0.5, symm=1
J(n)
J min = 1.25e−05
log10(J)
epoch iterations, n
1
0
−1
−2
−3
−4
−5
0
2000
4000
6000
8000
10000


3:3:2 net, μ=0.5, λ=1, symm=0, seed=20
J(n)
J min = 5.62e−04
log10(J)
epoch iterations, n
1
0
−1
−2
−3
−4
−5
0
2000
4000
6000
8000
10000

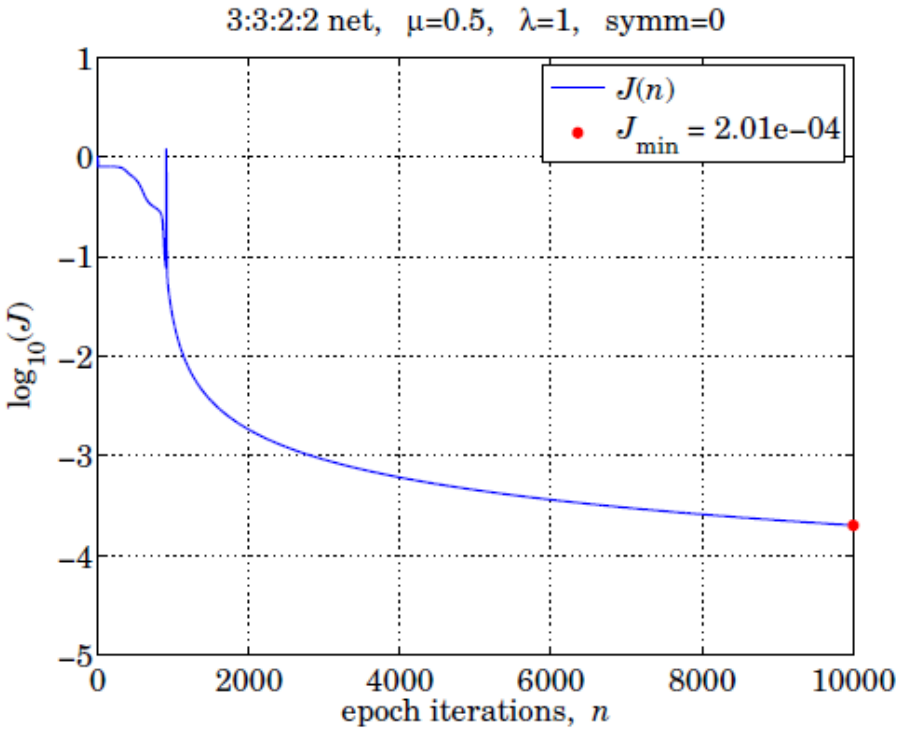

3:3:2 net, μ=0.5, λ=0.5, symm=0, seed=20
J(n)
J min = 3.10e−04
log10(J)
epoch iterations, n
1
0
−1
−2
−3
−4
−5
0
2000
4000
6000
8000
10000

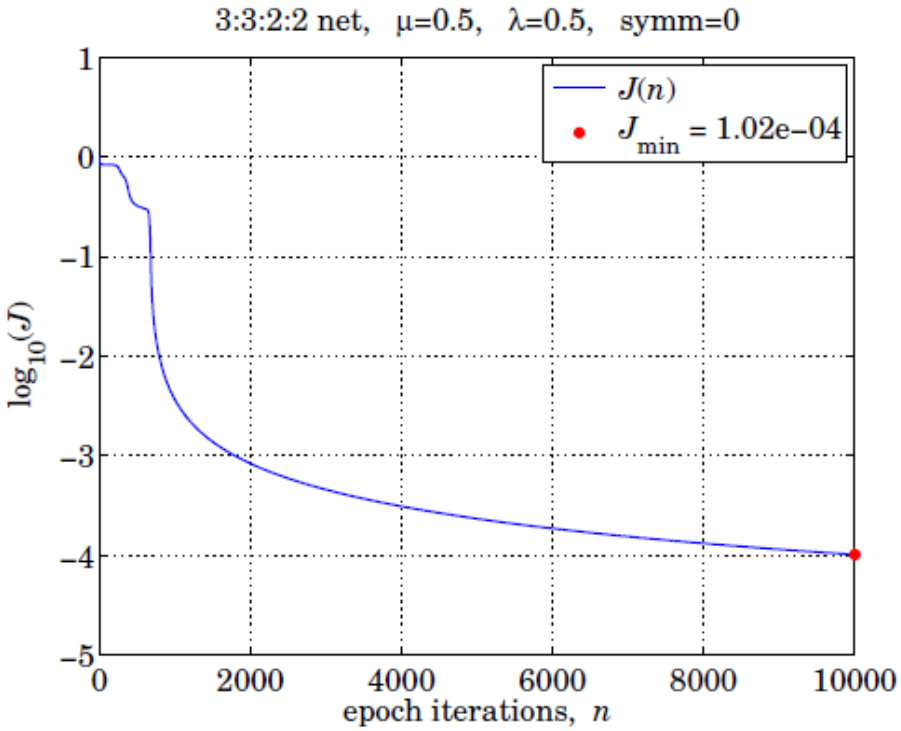
NN Experiments

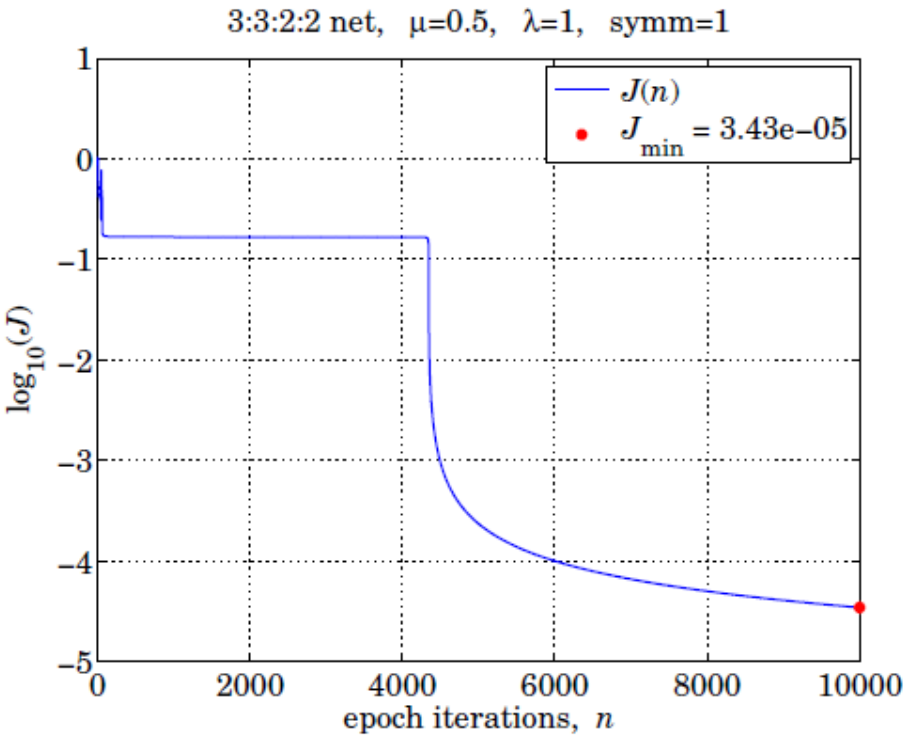
XOR problem with 3:3:2:2 network

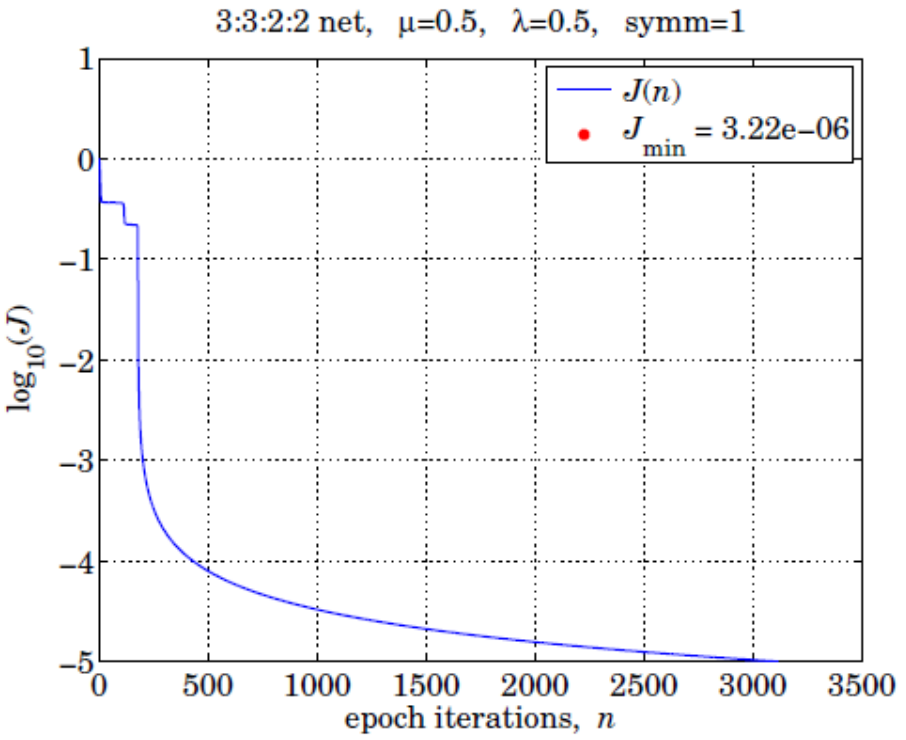
```
%      input          | XOR output
% ------------------------------
%   0     0     0    |    0     1
%   0     0     1    |    1     0
%   0     1     0    |    1     0
%   0     1     1    |    0     1
%   1     0     0    |    1     0
%   1     0     1    |    0     1
%   1     1     0    |    0     1
%   1     1     1    |    1     0
```

MATLAB code in, **s21nnexp.pdf**


3:3:2:2 net, μ=0.5, λ=1, symm=0
J(n)
J min = 2.01e−04
log10(J)
epoch iterations, n
1
0
−1
−2
−3
−4
−5
0
2000
4000
6000
8000
10000

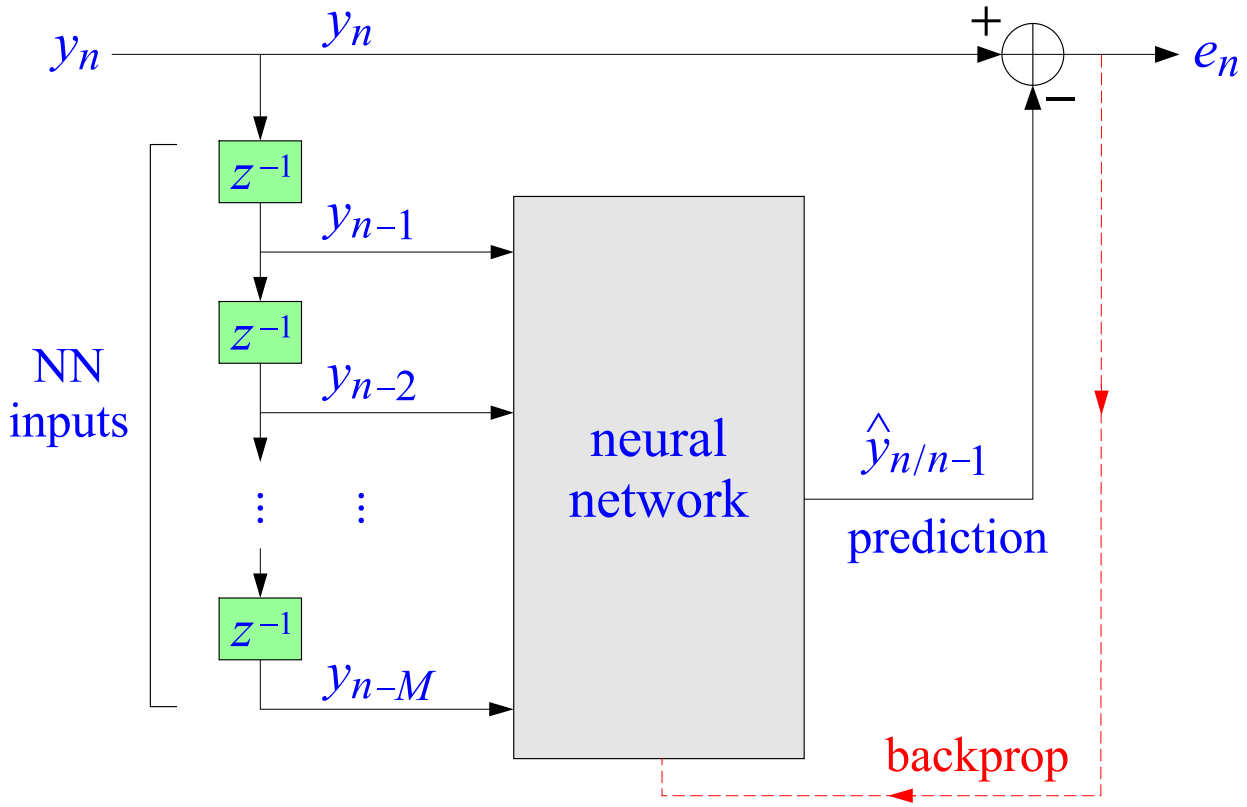

3:3:2:2 net, μ=0.5, λ=0.5, symm=0
J(n)
J min = 1.02e−04
log10(J)
epoch iterations, n
1
0
−1
−2
−3
−4
−5
0
2000
4000
6000
8000
10000

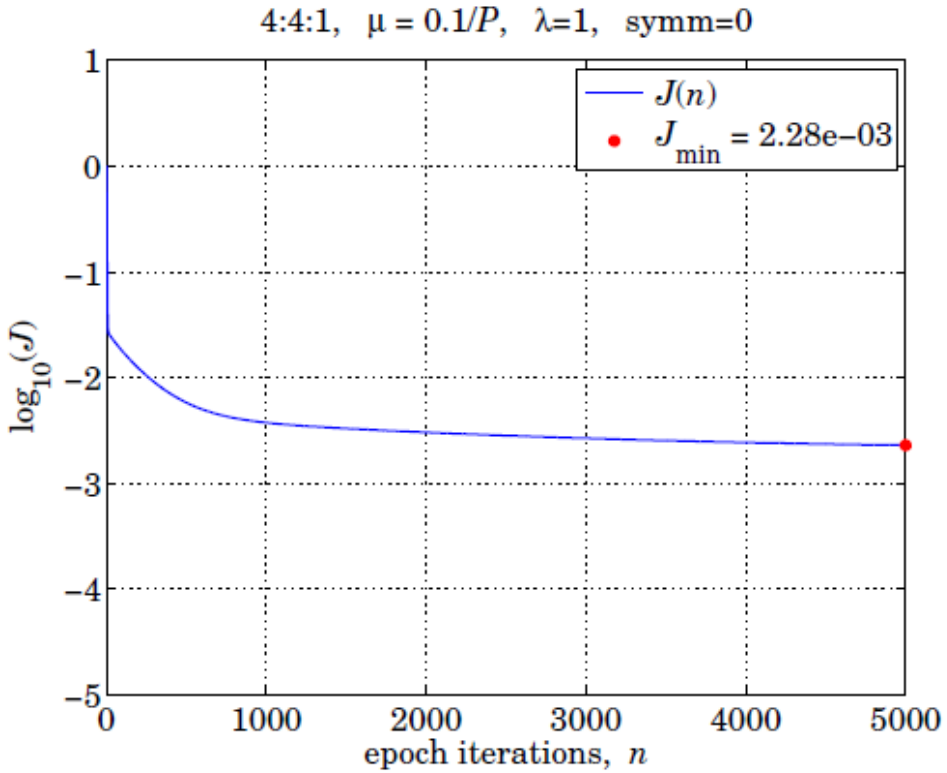

3:3:2:2 net, μ=0.5, λ=1, symm=1
J(n)
J min = 3.43e−05
log10(J)
epoch iterations, n
1
0
−1
−2
−3
−4
−5
0
2000
4000
6000
8000
10000

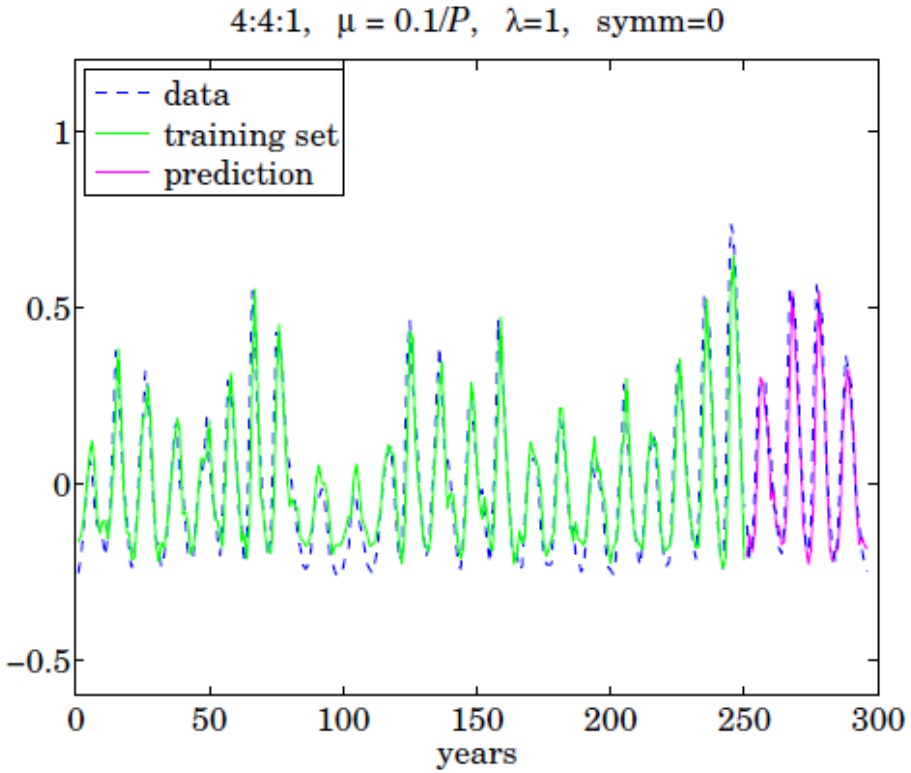

3:3:2:2 net, μ=0.5, λ=0.5, symm=1
J(n)
J min = 3.22e−06
log10(J)
epoch iterations, n
1
0
−1
−2
−3
−4
−5
0
500
1000
1500
2000
2500
3000
3500

NN Experiments

NN prediction of sunspot data with 4:4:1 network



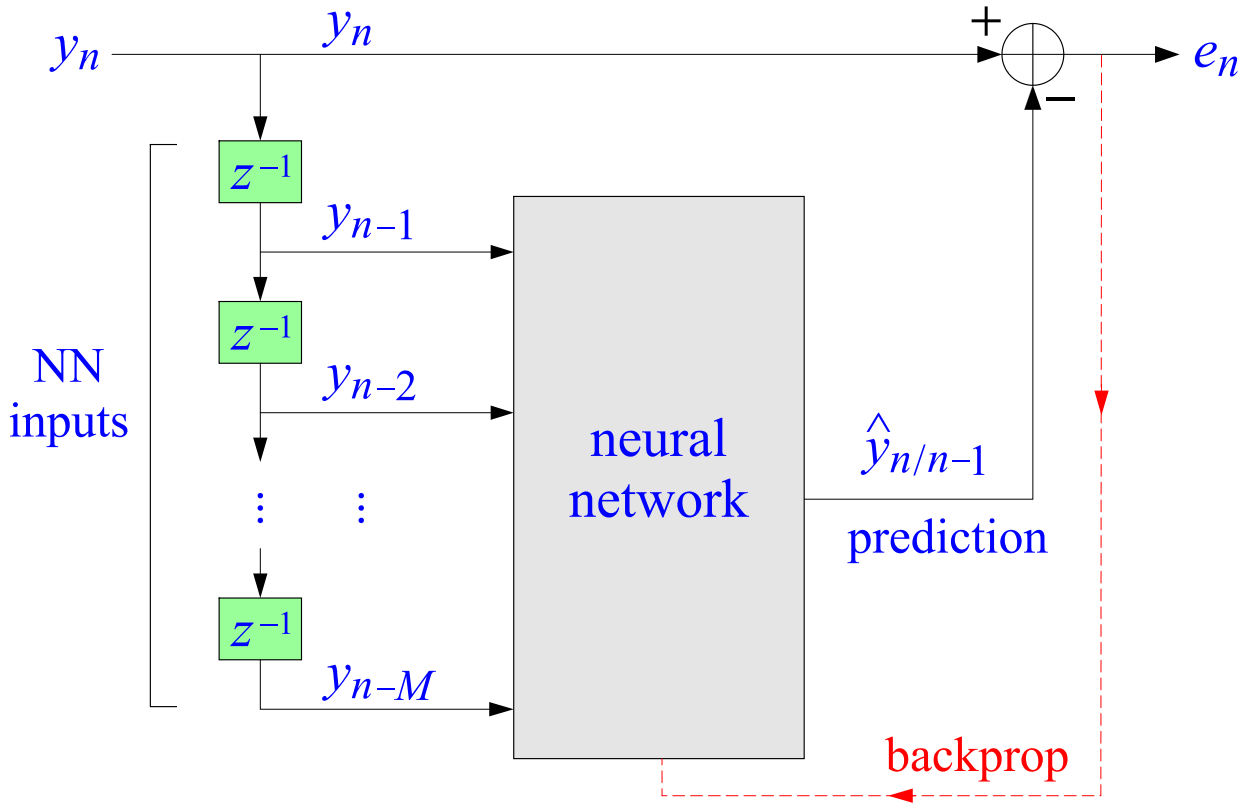


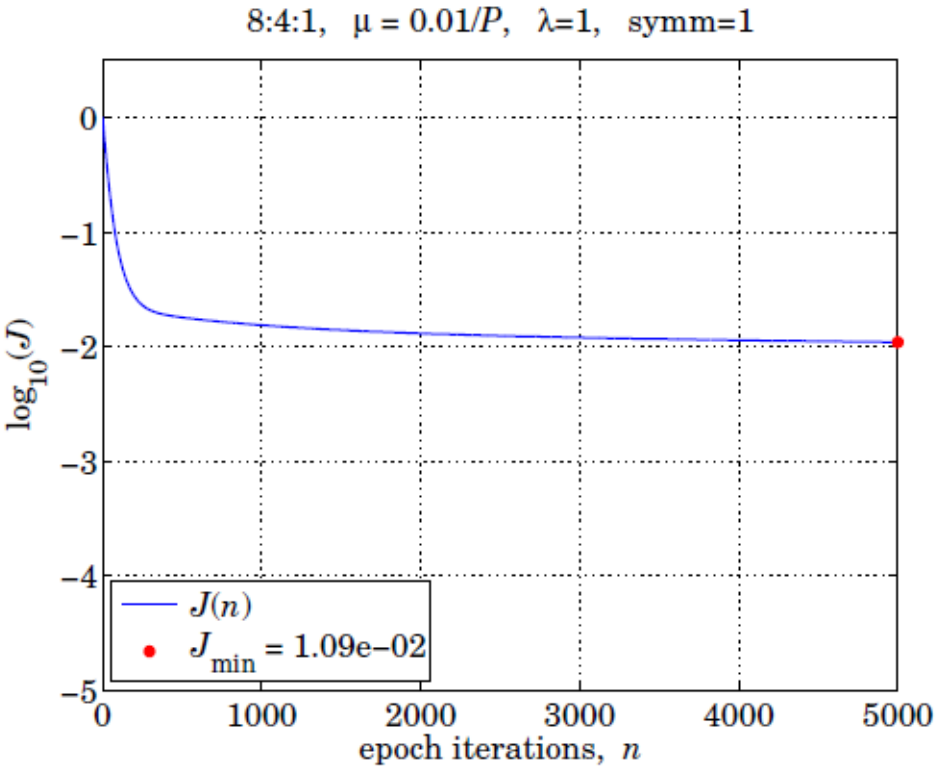


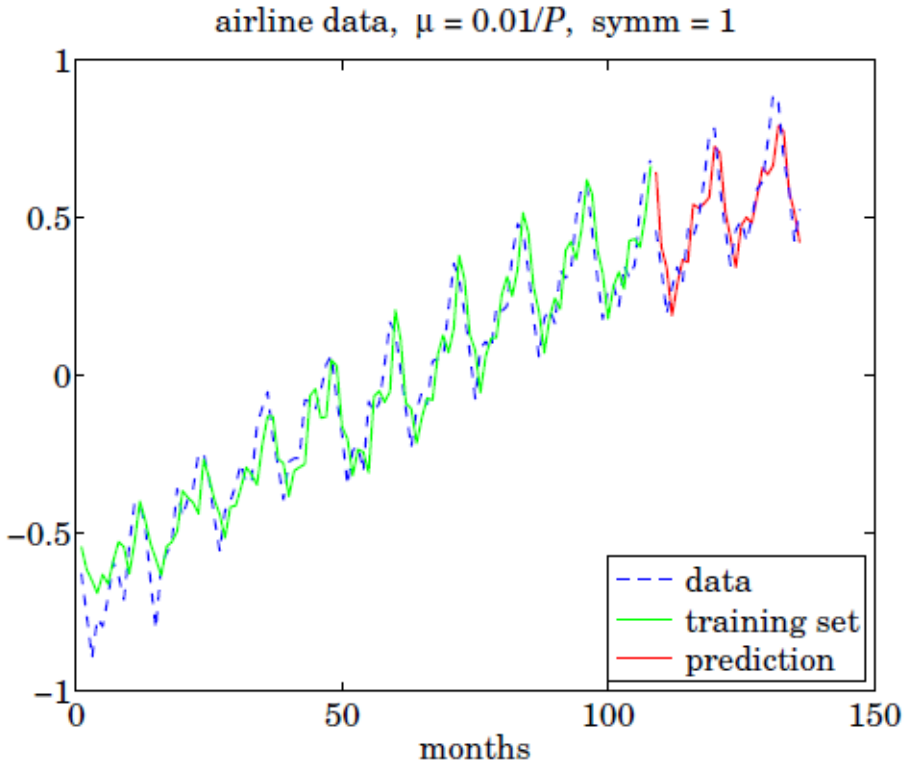
MATLAB code in, **s21nnexp.pdf**

NN Experiments

NN prediction of airline data with 8:4:1 network







MATLAB code in, **s21nnexp.pdf**