Table of Contents

Chapter IX- Adaptive Filters

Version 2.0
This Chapter is Part of:

Neural and Adaptive Systems: Fundamentals Through Simulation© by

Jose C. Principe
Neil R. Euliano
W. Curt Lefebvre

The goal of this chapter is to introduce the concepts of adaptive linear filtering. The linear combiner, which is the extension to time of the linear regressor of Chapter I, will be presented along with the extensions to time of the PCA networks of Chapter VI. Many useful applications will also be explored in the chapter.

Go to next section

1. Introduction

In this chapter we will study a different way of designing filters. Instead of synthesizing filters according to user specifications as we did in Chapter VIII, we will assume that the desired response is available. Hence we can use the ideas of adaptation to automatically produce the required functions without ever synthesizing the filter transfer function. For obvious reasons this methodology is called adaptive filters.

Adaptive filters are a special type of neural networks with two main characteristics. First, the adaptive systems are restricted to be linear which means that can be studied in linear vector spaces. Our present knowledge of linear adaptive systems can be framed in relatively simple mathematics, when compared to the one required for general (nonlinear) neural networks. Excellent textbooks are (Widrow) and (Haykin). We will see that adaptive filters are nothing but regressors in the signal space. The output of each tap of the delay line is considered a component of the input signal, and the goal is to regress it onto the desired signal. Normally we do not use bias in linear filters to preserve the properties of linearity as discussed in Chapter VIII. So what we will see below is a recapitulation of Chapter I.

We also extend to time the theory of Hebbian networks covered in Chapter VI. We will see that very interesting functions can be created with temporal PCA networks.

One important part of this chapter is to show how adaptive filters (the linear combiner and temporal PCA) can be applied to various signal processing functions. Applications that we cover are: prediction, model based spectral analysis, system identification, inverse modeling, interference cancellation, echo cancellation, adaptive inverse control, maximum eigenfilters, PCA in time, spectral analysis by eigendecomposition, noise cancellation and bind source separation. As you can see this is a very long list of applications which will help you solve problems using adaptive concepts.

Go to the next section

2. The Adaptive Linear Combiner and linear regression

The linear combiner was briefly discussed in Section 3 of Chapter VIII. It is built from a delay line of N-1 delays (N taps) and an adder, with equation

$$y(n) = \sum_{i=0}^{N-1} w_i x(n-i)$$

Equation 1

The impulse response of this system is specified by the weights w_i, and its region of support is N samples long. Hence, this system has a finite impulse response (FIR). The output of the linear combiner is a linear combination of the present and past N-1 inputs. Depending on the filter weights one can create any of the filter types discussed in Chapter VIII (lowpass, highpass, bandpass, bandstop). If the filter order is large enough any transfer function can be constructed with the linear combiner. The filter design described above is applicable when the signal (and the noise) are very well characterized, and the signal and noise spectra do not overlap. In such cases the design procedure is very well suited to solve the filtering problem.

Unfortunately, not all the problems of interest fall in this category. Sometimes the signal and the noise spectra overlap, and/or the signal has characteristics that change slowly over time. In such cases predefining the filter coefficients is not appropriate. More generally we may ask different questions such as signal modeling, prediction, etc. This is the reason alternatives to the filter design have been developed. Here we will study methods to adaptively determine the filter coefficients from the data and the existence of a cost function.

The *adaptive linear combiner* is a linear system with adaptive weights. So instead of fixing its weights according to some design criterion (i.e filter synthesis) we are back to the framework of adaptive theory. A desired response is presented, and the filter weights are modified such that a minimum of a cost function is achieved (Figure 1).

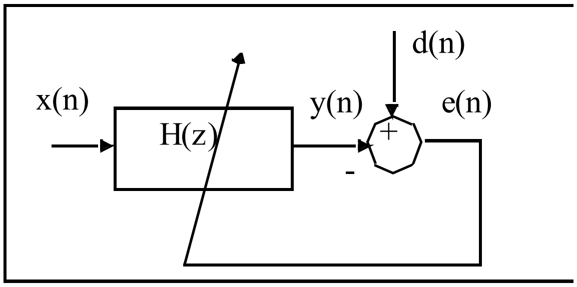


Figure 1. Paradigm to adapt the weights of the linear combiner

This block diagram is identical to the one presented for regression or classification. But it is more similar to regression than classification because normally the desired response is also a discrete signal with many amplitude values instead of {1, -1 (or 0)} as in classification. *So we can interpret the function of the adaptive linear combiner as a linear regressor (without bias) from the input time series to the desired time series.* Note that the order of the regressor is established by the number of taps in the delay line. In spite of the similarity with regression which warrants simply a review of Chapter I, we will formulate the problem of finding the optimal filter coefficients as done in adaptive filter theory to enhance our understanding and specifically address time series analysis.

Go to the next section

3. Optimum filter weights

The pioneering work in optimum filtering theory was done by Wiener and Kolmogorov . They solved the following problem in continuous time (i.e. in infinite vector spaces called Hilbert spaces): given a signal x(t) eventually contaminated by noise n(t), find the best linear system that is able to approximate the signal d(t). d(t) could be any signal, including the signal x(t) advanced by t seconds. In this most general case the linear system would work as a predictor and filter. Here we will only cover the discrete time case (finite vector spaces).

The problem is set up exactly like the regression problem. Let us define the error signal as the difference between the desired response d(n) and the system output y(n)

$$\varepsilon(n) = d(n) - y(n) \qquad \textbf{Equation 2}$$

when the input to the linear combiner with N weights is the signal x(n). We assume that all these signals are statistically stationary, i.e. they are random vectors, but their statistical properties do not change over time. We define the mean square error (MSE) as the expected value of the error square, i.e.

$$MSE = J = \frac{1}{2}E\left\{\varepsilon^2(n)\right\} \qquad \textbf{Equation 3}$$

This means that the optimization criterion is the power of the error, or the L2 norm of the error. We want to choose the filter weights to minimize J. In order to find the optimal weight values we derive the cost with respect to the unknowns (the filter weights) and set it to zero.

$$\frac{\partial J}{\partial w_k} = 0 \qquad\qquad k = 0,\ldots,N-1 \qquad \textbf{Equation 4}$$

Substituting Eq. 2 in Eq. 3 with y given by Eq.1 we obtain a set of N equations in N unknowns called the *Wiener-Hopf equations*

$$\frac{\partial J}{\partial w_k} = -E\left(x(n)d(n) - \sum_{i=0}^{N-1} w_i x(n-k)x(n-i)\right) = 0 \qquad \textbf{Equation 5}$$

The solution, using vector notation is the well known equation

$$\mathbf{p} = \mathbf{Rw}^* \qquad \textbf{Equation 6}$$

which has the same form as the solution of the regression problem. So the structure of the solution is the same, however **R** and **p** have here a slightly different meaning when compared with the regression case. They represent the *time autocorrelation* of the input and *time crosscorrelation* functions between the input and the desired response.

We define the autocorrelation function in time as

$$\mathbf{R} = E\left[\mathbf{x}_n\mathbf{x}_n^T\right] = E\begin{bmatrix} x^2(n) & x(n)x(n-1) & \dots & x(n)x(n-N+1) \\ x(n-1)x(n) & x^2(n-1) & \dots & x(n-1)x(n-N+1) \\ x(n-N+1)x(n) & x(n-N+1)x(n-1) & \dots & x^2(n-N+1) \end{bmatrix}$$

Equation 7

and the crosscorrelation function as

$$\mathbf{p} = E\left[\mathbf{d}_n\mathbf{x}_n\right] = E\left[d(n)x(n) \quad \dots \quad d(n)x(n-N+1)\right]$$

Equation 8

These definitions use the statistical operator E[.]. In practice the statistical operator is substituted by the temporal operator A[.].

$$A = \lim_{M\to\infty}\frac{1}{M}\sum_{i=1}^{M}x(i)$$

Equation 9

If some conditions hold (ergodicity), the autocorrelation and crosscorrelation functions computed with temporal operators equal their statistical counterparts. Commonly the limiting operation is also dropped, which means that the estimation is done within a window of M samples. The autocorrelation matrix becomes

$$\mathbf{R} = A\left[\mathbf{x}_n\mathbf{x}_n^T\right] = \begin{bmatrix} r(0,0) & \dots & r(0,N-1) \\ \dots & \dots & \dots \\ r(N-1,0) & \dots & r(N-1,N-1) \end{bmatrix}$$

Equation 10

The autocorrelation function for lags k1, k2 is obtained by cross multiplying the time series x(n-k1) with a shifted version of itself x(n-k2)

$$r_n(k_1,k_2) = \sum_{i=0}^{M-1}x_n(n-k_1-i)x_n(n-k_2-i) \qquad k_1,k_2 = 0,\dots,N-1$$

Equation 11

This is repeated for every lag up to N-1 to obtain all the entries of Eq. 10. The sub-index n means that we are doing this for the time series around sample n. The cross correlation vector P is obtained basically in the same way but now by cross multiplying the desired signal with shifted versions of the input,

$$p_n(k) = \sum_{i=0}^{M-1} d(n+i)x(n-k+i) \qquad k = 0,\ldots,N-1$$

Equation 11b

M should be at least ten times larger than the lag N of the autocorrelation function to obtain a reasonable estimate for the longest lag.

We can show Widrow that with the optimal weights given by Eq. 6 , the error e(n) becomes decorrelated with the output, hence it is also decorrelated with respect to the inputs, i.e.

$$E\left[e(n)x(n-i)\right] = 0 \qquad i = 0,\ldots,N-1$$

Equation 12

This gives a very powerful interpretation to the Wiener solution: the Wiener filter finds the weights such that the system output (which has to exist in the space spanned by the input since it is an inner product with x(n-i)) is the orthogonal projection of the desired signal onto the input space (Figure 2).

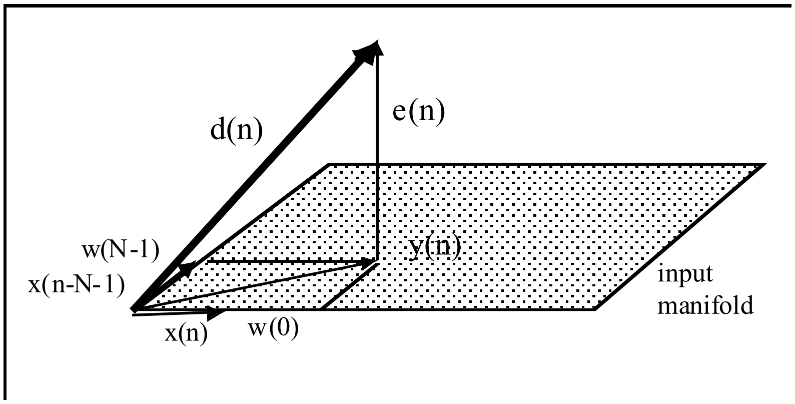


Figure 2. The Wiener solution in vector spaces

The orthogonal projection minimizes the error (the distance) between the desired response and the system output. Kolmogorov developed independently Eq. 12 to solve the minimization of MSE and its geometric interpretation.

Instead of analytically computing the optimum solution, one can search for the optimum set of weights as discussed in Chapter I. It turns out that the cost in the space of the weights is a paraboloid of equation

$$J = \frac{1}{2}\sum_i d^2(i) + \frac{1}{2}\mathbf{w}^T\mathbf{R}\mathbf{w} - \mathbf{p}^T\mathbf{w}$$

Equation 13

where **R** and **p** are the estimates of the autocorrelation and crosscorrelation function respectively, W is the weight vector, and d(n) is the desired time signal. An efficient search procedure to find the single minimum cost is to use gradient information. Under this paradigm, the weights are modified proportionally to the negative of the gradient estimated at each point

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta\nabla\mathbf{J}_k$$

Equation 14

where η is a step size parameter, and k is the iteration index. The most useful estimator of the gradient was proposed by Widrow . Instead of estimating the gradient through a statistical average, Widrow proposed the direct use of the instantaneous value of the gradient at each sample, i.e.

$$\nabla\tilde{\mathbf{J}}_k = -\varepsilon_k\mathbf{x}_k$$

Equation 15

This gives rise to the famous LMS algorithm to adapt the weights

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \eta\varepsilon_k\mathbf{x}_k$$

Equation 16

where $\mathbf{w}_{k+1}$ is the vector of weights at iteration k+1, ε_k is the error at iteration k and $\mathbf{x}_k$ is the input vector at iteration k. There are better algorithms to adapt the linear combiner but the LMS is still widely utilized due to its excellent compromise of simplicity and robustness versus computational complexity. RLS algorithm

NeuroSolutions 1
9.1 Design of filters using adaptation

In this example we will explore the power of adaptive filters. A very good example is to create filters that pick the harmonics of a square wave. As we know the square wave has a Fourier spectrum characterized by the odd harmonics of the period of the wave. We can see this easily by placing a FFT on top of the Axon

being fed by a square wave.

The goal of the adaptive filter is to individually pick the harmonics. The desired response is a sinewave of the particular harmonic. The adaptive filter is in fact implementing a bandpass function, but instead of using one of the filter synthesis programs to come up with the coefficients, we will use the desired response and the LMS algorithm to automatically design the filter for us. This is a good alternative in some cases.

Change the frequency of the desired signal to see how fast the weights adapt to the change. We can stop the simulation at any time and freeze the weights to compute the frequency response.

With adaptive filters, we are not confined to create the conventional filters we studied so far. We can smoothly design the frequency response. For instance we can transform a square wave to a triangular wave of the same period. Find out what is the frequency response of such system using NeuroSolutions.

NeuroSolutions Example

3.1. Wiener filter as a predictor

One application of the Wiener filter that raised a lot of interest is when the desired response d(n) is the input advanced in time, i.e. $x(n+n_0)$. This special case was something magical, because it was able to “predict the future” of a time series. But notice that the filter is simply computing the best approximation for a future value of the time series given the past samples. Since time signals have a time structure, i.e. they do not change radically from sample to sample, it is possible to extrapolate locally what the next sample is from the past N samples. Wiener was able to formulate mathematically prediction, which was a landmark.

NeuroSolutions 2

9.2 Adaptive filters for prediction

Prediction is just a special type of filtering. In fact prediction implements some type of lowpass filter which depends on the short-term time structure of the time series. The desired response in prediction is the signal advanced some samples (let us start with one). So the predictor has to look into the past of the signal and linearly combine the samples to obtain a good approximation to the position of the next sample. In a sense we are linearizing the trajectory around the present sample (the tangent space) and providing one point in this space.

Let us use a smooth signal first (sum of two sinewaves). If you let the system run, you quickly find the error becomes very small if the FIR has at least 8 taps (and a delay of 2 between taps, you will need 16 taps and a delay of 1 to get comparable results, but notice that the system has twice as many coefficients). Notice that the frequency response is in fact a lowpass type of filters. Prediction is in fact a kind of extrapolation.

If you now use a square wave with period 32, you will find out that the system does not work well at all. Can you discover why? The square wave is discontinuous, so there is no way the predictor can anticipate the abrupt change in amplitude by just looking at 8 samples of the recent past of the signal. In order to predict the waveform, a full cycle must be present in the delay line. In this case the predictor works very well. Note once again that is sufficient to increase the delay between taps to 4, and keep the number of coefficients at 8 (there is no need to have 32 weights….).

Another case where the predictor works badly is when there is white noise in the data. Since the samples of white noise are uncorrelated from sample to sample, there is no information in the past of the signal to help predict the future values. In general signals that have a very complex time structure are difficult to predict with small models. We can get an idea of the predictability of the signal if we visualize it in state space and find abrupt jumps in the plot.

You are ready now to bring in your pet data set to predict it (We do not take responsibility if you try to predict the stock market….).

NeuroSolutions Example

3.2. The Wiener filter as a function approximator

Chapter V was devoted to present an unifying view of regression and classification as special cases of a broader problem called function approximation. But remember that at the time only static problems were being addressed. Now we know how to handle time functions with filters, and in particular for the linear combiner we know how to optimally set its parameters to approximate the desired time series. Can we extend the views of function approximation to the linear combiner?

The answer is affirmative, and it is rather important for framing the application domains of linear combiners. If we compare Figure 1 with Figure 1 of Chapter V we can see immediately the similarity. Here we assume that the desired signal d(n) is a fixed but unknown function of the input x(n), i.e. d(n)=f(x(n)). The role of the Wiener filter is exactly to discover the unknown function f(.).

The noticeable difference here is the use of time functions, while in Chapter V the data was multidimensional, but static. However, we provided in section 2 of Chapter VIII a mapping from time to a vector space called the signal space, which essentially stated that a segment of a time series can be always mapped to a point of a vector space of appropriate dimensionality. So the difference between time functions and static data washes away if we think that the approximation is being performed in the signal space. Another minor difference is that we do not use a bias in the linear combiner (to comply with the definition of linear system).

We can then conclude that *the linear combiner is doing regression in the signal space*. Looking at the Wiener filter operation from the point of view of function approximation, we see that it is a linear function approximator. The basis of the signal space are the input signal and its delayed versions, so the linear combiner is finding the best projection of the desired signal onto the input signal space. The best projection must lie in the input signal space since it is a linear combination of the basis.

In engineering terms this function approximation property can be put to good use. We already saw the application of Wiener filters to prediction. In the application section we will discuss system identification and noise cancellation. system identification fundamentals

Go to the next section

4. Properties of the iterative solution

The properties of the iterative solution to find the minimum of J have been extensively studied in Chapter I, and they apply also to the adaptation of the linear combiner. The fundamental difference is the nature of the autocorrelation and crosscorrelation functions that now are related to time information. Here we will only summarize the most important ideas.

4.1. Largest step size

The largest step size η_{max} to guarantee convergence to the optimum value is

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

Equation 17

where λ_{max} is the largest eigenvalue of the input autocorrelation function R. In practice we choose

$$\eta = \frac{\eta_0}{tr[\mathbf{R}]}$$

Equation 18

where η_0 has to be set according to the problem (0.01~0.5) and the trace of **R** can be estimated by the signal power at the taps $\sum_{i=0}^{N} x^2(n-i)$. The trace of **R** is the sum of the eigenvalues, so tr[**R**] is an upper bound for λ_{max}. Notice that if the input power changes the step size will change automatically. This is called the *normalized step size*.

NeuroSolutions 3

9.3 Choosing the stepsize

Please check the convergence/divergence of the linear combiner as a function of the stepsize. We provide here the breadboard for the linear combiner for your convenience, but since these exercises are duplications of the ones in Chapter I we will not elaborate the details.

NeuroSolutions Example

4.2. Speed of Adaptation

In steepest descent, each weight will converge to the minimum with a different time constant given by the inverse of the respective eigenvalue. So the mode with the longest time constant is

$$\tau_{\max} = \frac{1}{\eta\lambda_{\min}}$$ **Equation 19**

So the search is limited by the adaptation of the slowest mode. The existence of many time constants is readily apparent in the learning curve. This means that the largest step size is limited by the largest eigenvalue (Eq.17), and the speed of adaptation is limited by the smallest eigenvalue (Eq. 19). Hence, the *eigenvalue spread* of the input autocorrelation function defines the performance of the steepest descent algorithms (LMS in particular).

For time signals, the speed of adaptation is coupled with the actual time it takes to adapt the linear combiner, since when we multiply the number of samples by the sampling period we obtain time in seconds.

NeuroSolutions 4

9.4 Speed of convergence

Please check the speed of convergence of the linear combiner as a function of the stepsize. We provide here the breadboard for the linear combiner for your convenience, but since these exercises are duplications of the ones in Chapter I we will not elaborate the details.

NeuroSolutions Example

4.3. Rattling

The choice of the step size in the LMS is a compromise between speed of adaptation and misadjustment. The faster the algorithm converges, the higher is the misadjustment, i.e. the excess error with respect to the theoretical minimum error. The misadjustment can be approximated by

$$M = \eta tr\left[R\right] \qquad \text{Equation 20}$$

If we assume that the eigen-spread of R is small we can say that the algorithm converges in

$$settling \quad time = 4\tau = \frac{N}{M} \qquad \text{Equation 21}$$

This gives the following rule of thumb: for a misadjustment of 10%, the algorithm will converge in a number of samples equal to approximately 10 times the number of weights. So we can really think of translating speed of adaptation with actual external time (in seconds) by multiplying the number of samples by the sampling period. In a lot of applications, the adaptation time of the linear combiner is reasonable, yielding systems that can track changes in the signal dynamics.

NeuroSolutions 5

9.5 Misadjustment

Please check the rattling of the linear combiner as a function of the stepsize. We provide here the breadboard for the linear combiner for your convenience, but since these exercises are duplications of the ones in Chapter I we will not

elaborate the details.

NeuroSolutions Example

4.4. Neural versus adaptive systems

There is a major difference between the neural network models covered so far and the adaptation of the linear combiner. In ANNs we always work with a training set, and test the performance in an independent test set. We may have to re-use the training set samples many times until the ANN converges. But notice that once the ANN is trained its *weights are fixed* for the test set. So, although the ANN has trainable weights, they are frozen during the testing. Therefore, the *ANN is no longer adaptable after training*.

Adaptive filters are utilized in a different way. They are *continuously adapted*, i.e. the weights are always being modified according to the LMS rule. There is no training set /test set division when working with adaptive filters. When the data is first sent through an adaptive filter, the performance of the system is far from optimal, but quickly the system parameters find reasonable values (convergence in ~10N samples). This is fast compared to the backpropagation convergence.

An advantage of this procedure is that the system *is able to track changes* in the structure of the time series, what does not occur with the training procedure applied to ANNs. However, the system forgets old data and can also be taken from its optimal parameter setting if outliers (noise spikes) occur. It will recover rather quickly as we have seen above due to the good tracking properties of LMS in a linear topology.

There is clearly a trade-off between trainable versus adaptive weights, or between neural and adaptive systems. A neural system seeks to model the data as long-term memory in its weights, while the linear combiner is basically a tracker, that is, it only has short-term memory. For general use the autonomous system needs both. This is not easy to do. Within the linear model, the Kalman filter solves this problem by using the state information. The state is estimated from the data, and it corresponds to the long-term memory of the system. When the system can not predict the input it decides to update the output for tracking. But we still do not have a general framework to accomplish this for the nonlinear model.

NeuroSolutions 6

9.6 Tracking ability

This example shows the tracking ability of the linear combiner. Tracking is a new concept that only makes sense in time series. The goal is to be able to change the filter weights such that a change in the desired response or the relation input/desired signal is followed. This is a very important characteristics in real world problems where the data may slowly change (and sometimes not that slowly).

Tracking is intrinsically a compromise. In one hand, we would like the linear combiner to reach the bottom of the bowl with low misadjustement which means a small stepsize. On the other hand, if the location of the minimum changes with time, we would like to have a sufficiently large stepsize to follow it to the new location. The trade-off can only be solved experimentally.

NeuroSolutions Example

Go to the next section

5. Hebbian networks for time processing

The Hebbian networks discussed in Chapter VI can be extended to time processing if a delay line is incorporated at their input (Figure 3).

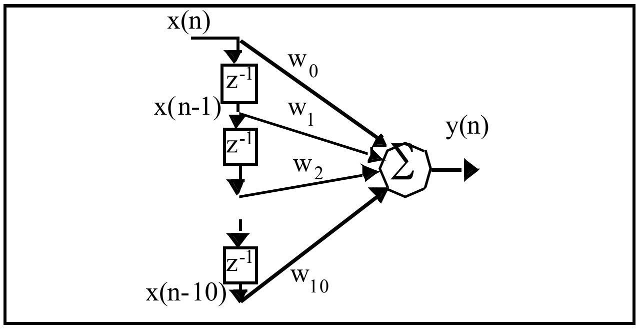


Figure 3. Hebbian PE extended to time

The single input signal flows into the delay line and the Hebbian PE receives not only the present input but also N-1 past samples. As in its static counterpart, the PE will estimate the eigenvectors and eigenvalues of the autocorrelation of the input, but now the input is a signal in time. Karhunen and Loeve did the original work in this area and temporal PCA is called the Karhunen Loeve transform (KLT) in their honor. The mathematics are very difficult for continuous time signals (the original formulation), but we already know all the mathematics needed to treat the case of discrete signals of finite length as we are interested here.

This discrete version of the KLT will be called here temporal PCA. The solution of temporal PCA is also an eigendecomposition given by

$$\sum_{k=0}^{N-1} \mathbf{r}(l,k)\mathbf{e}_i(k) = \lambda_i \mathbf{e}_i(l) \qquad i,l = 0,\ldots,N-1$$

Equation 22

or in matrix form:

$$\mathbf{R}\mathbf{e}_i = \lambda_i \mathbf{e_i}$$

Equation 23

where λ_i and $\mathbf{e}_i$ are the eigenvalues and eigenvectors, respectively, of the autocorrelation matrix, **R** given by Eq. 10 .

The Temporal PCA equations proceed analogously to the static PCA case. Since the eigenvectors are orthonormal,

$$\mathbf{e}_i^T \mathbf{e}_j = \delta_{ij}$$

Equation 24

they form a complete basis in the N dimensional space. Thus, the vector x(n) can be represented without error by an expansion over the N eigenvectors:

$$\mathbf{x}(n) = \sum_{i=0}^{N-1} u_i(n)\mathbf{e}_i = \mathbf{E}\mathbf{u}(n)$$

Equation 25

with $\mathbf{E} = [\mathbf{e}_0, \mathbf{e}_1, ..., \mathbf{e}_{N-1}]$. The matrix **E** is deterministic and full rank. The coefficients ui(n) of the expansion can be found from:

$$z_i(n) = \mathbf{e}_i^T \mathbf{x}(n) \rightarrow \mathbf{z}(n) = \mathbf{E}^T \mathbf{x}(n)$$

Equation 26

The ui(n) are called the principal components. We have seen in Chapter VI that Sanger's rule is an on-line algorithm to compute this decomposition, so we can adapt the parameters of the network of Figure 3 with Sanger's rule to compute temporal PCA. In this network the eigenvectors **e**i are the weight vectors **w**i. The outputs are the principal components ui(n). We can recover exactly the input **x**(n) from the ui(n) if we use Eq. 26, i.e. we have to take the transpose of the weight vector.

5.1. Properties of temporal PCA

When viewed as a set of eigenfilters, the eigenvectors of the autocorrelation matrix have very interesting properties. In the limit of long windows, the KLT expansion reduces to a Fourier basis, and the eigenvalues become the power spectrum (see Van Trees). For a finite number of delays, the relation between eigenvalues and the power spectrum is not exact but is a good starting point. Temporal PCA becomes a set of FIR bandpass filters with center frequencies attracted to the peaks of the spectrum.

Less well known is the relation between the ordered eigenvalues and spectral peaks. Assume that the eigenvalues of **R** are ordered from largest to smallest, $\lambda_0 \geq \lambda_1 \geq .... \geq \lambda_{N-1}$, as the number of delays approaches infinity, the maximum eigenvalue of **R** approaches the maximum value of the power spectrum of x(n), and the power spectrum of the corresponding maximum eigenfilter becomes an impulse centered at the corresponding maximum frequency of the power spectrum, i.e.

$$\lambda_0 = Max_w[X(w)]$$

where X(w) is the power spectrum of x(n). In fact, the 2nd eigenvalue and eigenvector will also exhibit the same properties. This can best be explained in terms of the orthogonality of sine and cosine waves. For any pure sinusoidal basis, a 2nd orthogonal basis of the same frequency can be found through a 90 degree phase shift of the original basis. For finite but sufficiently large L, the first two eigenvectors act as bandpass filters around the peak in the signal spectrum. The pass band becomes increasingly narrow as the number of taps is increased. Generalizing for the other components, we can say that eigenfilters can be approximately regarded as filters matched to peaks in the signal spectrum.

A multi-output Hebbian network trained with Sanger's rule will compute the PCA components on-line, i.e. in a sample by sample basis. If instead of the Hebbian learning we use the anti-Hebbian, then a new set of functions are possible as we briefly mentioned in Chapter VI (novelty filters).

Go to the next section

6. Applications of the Adaptive linear combiner.

It is very instructive to address the applications of the adaptive linear combiner. Here we will present simulations that utilize the adaptive linear combiner for prediction (time series identification), interference canceling, echo-cancellation, line enhancement, system identification, and adaptive controls. How can the same system be used for so many applications? We will see that the principle involved - *exploiting correlation between the desired response and the filter output*- is always the same, but the adaptive filter is used in different arrangements, producing very different end results.

Go to next section

6.1. Prediction

In prediction the desired signal is the input advanced by n_o samples. Normally n_o=1, which is called *single step prediction*. We will study here single step prediction, but multi-step prediction (n_o>1) is the same except that the task becomes increasingly more difficult for larger n_o.

Prediction is widely applied in forecasting and modeling. Forecasting is more easily understood and it is the one we will explain first. In forecasting problems, we know a time series until the present time n. We wish to know what is going to be the next sample n+1 of the time series. The idea is that if we can train a system to predict the former values of the time series (i.e. if the time series for sample k<n is fed to the system and it is requested to predict sample k+1), then the system will also predict accurately sample *n*+1.

It is obvious that we have to assume *that the structure of the time series does not change in time (i.e. the signal is stationary)*, otherwise the goal is illusory. In a lot of applications this model is reasonable, and even if the signal time structure changes, we assume it changes slowly, such the system can continuously track the changes.

Figure 4 shows the block diagram to implement single step prediction. The input signal is delayed by one sample before it is fed to the linear combiner. The desired signal is the current value of the input x(n). The input of the linear combiner is therefore delayed one sample with respect to the desired response.

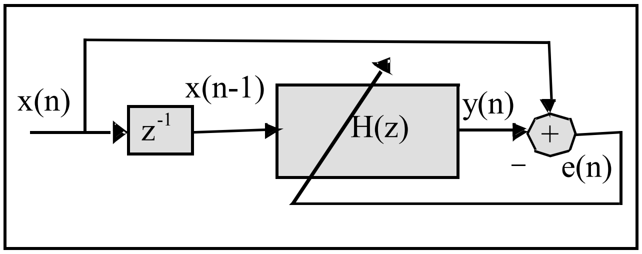


Figure 4. Block diagram to implement one step prediction

9.7 Prediction of time series

This breadboard shows the prediction of a complicated time series, the sun spot data. Sun spot activity is an example of a phenomena we can not control but affects us a lot (loss of radio communications, weather, etc.) so the only option is to look ahead and see what is going to happen. The sun spot data is very complex (very possibly chaotic) but the next step prediction with the linear combiner is not bad.

Let us select the size of the tap delay line at 16. This is the variable that controls the size of the model so it has to be experimentally determined. As we saw, larger models normally work better, but this depends on the type of data. Here for instance, larger models do not improve performance because the data is chaotic. Larger models simple utilize more information from the past to establish the value of the next sample. But if the time series changes rapidly and it is aperiodic this is of no help (remember the square wave?). You should experiment with this variable in the breadboard.

The accuracy of the prediction is not uniform along the training set. In order to see this, modify the learning from batch (which uses the full data) to custom with 500 samples in the backcontroller. You also have to modify the reporting on the L2 criterion from batch to weight update. The accuracy drops rapidly for multiplestep prediction. You can create multistep prediction by going to the desired response file and customize it (create a segment starting at samples 2, 3, etc.) . Watch the MSE for the different cases.

NeuroSolutions Example

When seen as the prediction of the future of the time series, this seems an impossible problem due to the causality inherent in any physical system. However, if we look at it from the point of view of signal space, prediction is really a simple problem of fitting the best hyperplane to the signal trajectory. The source of the error can be due to the fact that the hyperplane is not a good fit to the trajectory, or the characteristics of the signal have changed from the present to the future.

Go to next section

6.2. Prediction for modeling

It is important to interpret prediction as the identification of the linear system that produced the time series x(n). Let us analyze the block diagram of Figure 5. Here it is assumed that there exists an hidden linear model M(z) that generates the time series x(n) when its input is white noise (white noise is a random signal that contains equal amount of every possible frequency, i.e. its FFT has a flat spectrum).

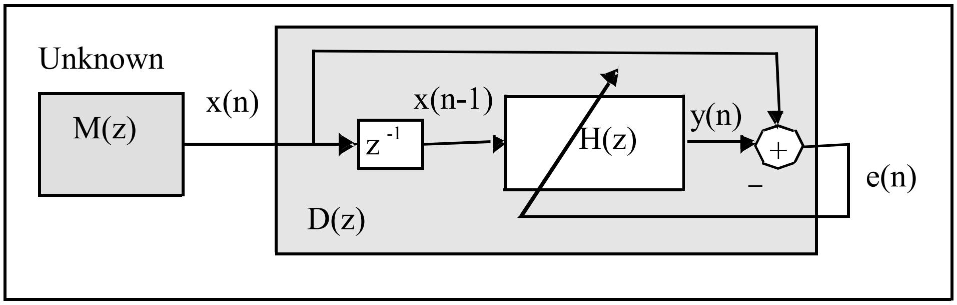


Figure 5. Prediction as modeling the system that produced the time series

When we use the linear adaptive filter H(z) to predict the next sample and the prediction is successful the output error is also white noise, i.e. E(z)= k. Under this condition, it is easy to show that the block D(z) has to be equal to k/M(z) since,

$$M(z)D(z) = k \qquad \rightarrow \qquad M(z) = \frac{k}{1 - z^{-1}H(z)}$$

Equation 27

which means that the zeros of H(z) correspond to the poles of the unknown system M(z). By using the adaptive linear combiner (which is a MA (moving average) model) we are modeling M(z) as an AR (autoregressive) model. This inversion of models is rather important and should be remembered. Hence, this scheme produces very good results when the time series has spectra with sharp peaks and broad valleys, which is indicative of an all pole model. When the time series spectrum has narrow valleys, a linear combiner of small order is not very effective.

The coefficients of the linear combiner concentrate the extracted information from the time series in the sense that different time series normally yield different model coefficients (note however that two time series can yield the same model coefficients as long as they have the same autocorrelation function). Therefore, the model coefficients can be used to classify the time series, as is done for instance in speech recognition where the model parameters are used as input to a classifier.

The order of the linear combiner is related to the complexity of the time series. Each spectral peak requires a pair of poles so it will require two weights in the linear combiner. The spectral "tilt", i.e. how the energy decays across frequency must be also modeled and requires several other (real) poles which also imply extra weights. By counting the number of peaks in the spectrum (and increase this number somewhat to model spectrum tilt) one can estimate the order of the linear combiner. If more accurate estimation is required, then Akaike or Rissanen criteria explained in Chapter V should be utilized.

NeuroSolutions 8
9.8 Prediction for modeling

One application of prediction is to derive a model for the signal generation model. We can apply this approach to extract parameters of real world data and use them to quantify the system that produced the data (called signal modeling) or for pattern classification (this effectively is done in speech recognition as we will see below).

For signal modeling, we will use exactly the previous breadboard. We will let the system train, and will look at the frequency response of the system which is obtained by sending every so often an impulse through the model and taking the FFT. After training, the filter weights and the parameters of the system that

generated our input signal should be very similar (as long as the signal model is linear, all pole and the order of the linear combiner matches its order.).

Next we illustrate the use of the linear combiner for feature extraction. We will start with a 6 tap delay line, and the data to be used is the sustained sound "a". We will let the system train and look at the coefficients. Then we will modify the file to the sustained sound "i" to see the difference in the coefficients. Notice also how different the two frequency responses are, but since they are FIR we can see broad peaks. If you look closely at the spectrum of the input and the frequency response they seem to be flipped over (this is not by chance, as we will see below).

Train the system several times and see that the weights are similar. Now change the training to batch and see how much smoother the overall spectrum gets. Increase the size of the delay line and observe the frequency response become more wavy, i. e. with more detail. The weights of the liner combiner can be used to classify the speech segment. They are in fact used in speech recognition and they are called the LPC (linear predictive coding) coefficients.

NeuroSolutions Example

The last comment we would like to make is to point out the ingenuity of the prediction formulation for system identification. Notice that we are using a supervised framework (a desired signal and an error criterion) but we just have a single time series, which is exactly what characterizes unsupervised (self-organizing) frameworks. Prediction is one of the cases (the other being autoassociation studied in Chapter VI) where a supervised framework is utilized to model the input data.

Go to next section

6.3. Model Based Spectral Analysis

The prediction framework also provides a very powerful method of doing spectral analysis, i.e. of computing the spectrum of a time series using and implicit model. Eq. 27 basically explains it all. Note that if the input to M(z) is white (and normalized power), then the spectrum of the time series is X(z)=M(z), so we can also say that the Z transform of the time series is

$$X(z) = \frac{k}{1 - z^{-1}H(z)} = \frac{k}{1 - \sum_{i=0}^{N-1} w_i z^{-i-1}}$$

Equation 28

when a linear combiner with N delays is used for the model. Notice that the spectrum of the time series can be computed by making $z = e^{j2\pi\frac{l}{L}}$ where L is the number of frequency samples we want to obtain in our spectral representation. This spectrum estimator assumes an AR model for the time series, and produces very peaky spectra (in fact distorting the spectrum for signals that have a different generation mechanism). It is frequently used in spectral analysis when one wants to quantify the presence of closely packed sinewaves and the data segments are of short duration.

When LMS is the learning algorithm to obtain the weights of the model the precision of the technique is poor due to the difficulties of gradient descent learning (rattling and coupled modes). Note that the weights are going to define the position of the poles of the inverse system. So even if all but one the coefficients are in their final position ALL the poles will be misplaced. This high sensitivity to the placement of the zeros in inverse modeling is well known. Therefore in AR based spectral analysis we normally prefer the RLS algorithm or the analytic solution.

NeuroSolutions 9
9.9 Prediction for spectral analysis

In the previous example we were a step away from doing model based spectral estimation. Conceptually what we need to do is to invert the MA model obtained during training to create a system given by Eq. 28. Note that this can be achieved easily if we create a TDNN Axon of order N in cascade with a FullSynapse and

feedback its output to the TDNNAxon. All the weights determined during training should be copied (entered by hand) into the FullSynapse.

The difficulty comes in the quality of the model obtained during LMS training. Notice that for model based spectral analysis we are creating a recurrent system, i.e. a system with poles which may be unstable. So any minor discrepancy in the linear combiner weights may leads to unstable spectral models. In particular for speech where the signal has a peaky spectrum, this almost always is the case unless the stepsize of the LMS is well controlled (try also batch learning which tends to be more accurate). Try to display the spectrum obtained during the training of the "a" and "i" sounds obtained in the previous example.

NeuroSolutions Example

Model based spectral analysis is one application that is much more demanding then prediction. In fact there are many similar set of coefficients that provide a small prediction error. However, only one of these will be the actual model for the system that generated the signal. Prediction uses an input-output model, while in signal modeling we are also interested in finding the parameters of the system. This is where other techniques (such s state space models) become more appealing.

Go to next section

6.4. System identification

In system identification the goal is to model the input-output transfer function of an unknown plant by injecting a common signal to the plant and the adaptive linear combiner. The output of the plant becomes the desired response (Figure 6).

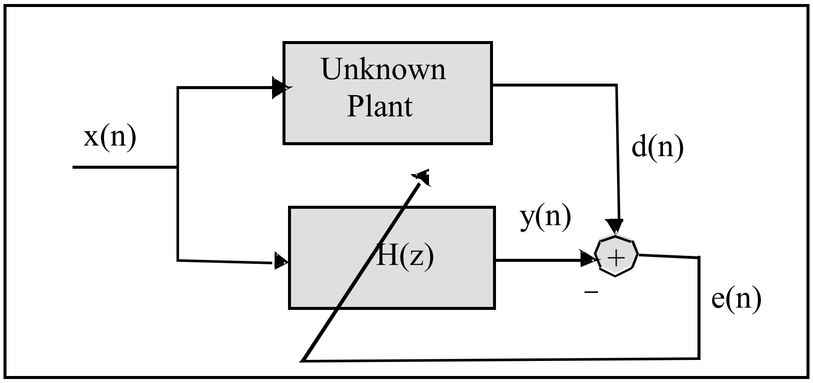


Figure 6. Adaptive filters for system identification

The goal of the task is to minimize the output error, and since the error is e(n)=d(n) -y(n) i.e., the difference between the plant output and the model, minimizing the error makes the model output as close as possible to the system output. Since both the model and the plant are excited by the same input, this scheme effectively makes H(z) very similar to the plant transfer function (assuming the plant linear and with constant coefficients*). The importance of input-output modeling is that it completely describes the external behavior of the unknown plant.*

When the adaptive linear combiner output approximates the desired response it produces an equivalent input-output model for the plant. The error can go to zero only if the plant is exactly modeled by the linear combiner. In this case, this means that only MA (moving average) plants can be exactly modeled (if the order of the linear combiner is large enough). In all the other cases, the modeling will be approximated. Normally the injected signal to the plant is white noise or a signal of broad spectrum to excite all the modes of the plant.

NeuroSolutions 10

9.10 System identification

The data for this example is created in the following way. In a different breadboard we stored both the input white noise and the output of a plant which is a very simple and known system (all weights equal to 1/N). The input is assigned to the input of our linear combiner, and the desired response file contains the output of

the plant. So effectively we are utilizing the block diagram of Fig 5.

Now the idea is just to train the system having as a variable the size of the model. We place both MegaScopes and FFTs at the inputs, outputs and desired responses such that we can have an idea of the system behavior during adaptation. We also have a probe that is showing the frequency response of the model. This is very similar to the system that we constructed. Reading the weights we see that they are very similar, which proves that the identification was successful.

We have done this assuming we knew the order of the unknown system. Now we should change the number of taps to a smaller and large value and see the effect in the error and in the values of the weights. We should also try a more difficult "plant" with poles and zeros to see that in fact the error is higher. But increasing the order of our model is able to decrease the error somewhat.

NeuroSolutions Example

Note that the linear combiner and its parameters are simply an input-output model of the unknown system. This is the reason it is called "black box" modeling. When little is known about the plant the procedure is reasonable. Alternatively, *physical modeling* can yield better results, by providing the functional form of the plant, but requires a lot more knowledge about the real world system which may not be available.

Remember that *system identification is the choice of an element of a family of models.* The family is defined by the topology of the model. In this case the family is the moving average (MA) models. But we will see below (and in the next chapter) other choices. The choice of the element within the family is effected by the parameterization, i.e. the model order and the actual value of the coefficients. As you can expect from the description, this is not an unique procedure, i.e. there are many possible models for a given unknown plant. We should use the Occam's razor argument to pick the smallest model that meets our performance criterion. In Chapter V we presented several methods to find the best model order, but in linear systems Akaike's criterion is probably the most widely used.

Go to next section

6.5. Inverse Modeling

The idea of inverse modeling is very important in a lot of applications and complements the direct modeling approach that we just discussed. Sometimes we do not have the freedom (in fact luxury) of choosing the plant input as we must do in the case of direct modeling. In a lot of applications we have to use the plant with its normal input. In inverse adaptive modeling the model system is in series with the unknown plant and the input to the model is also the desired response of the cascade (Figure 7).

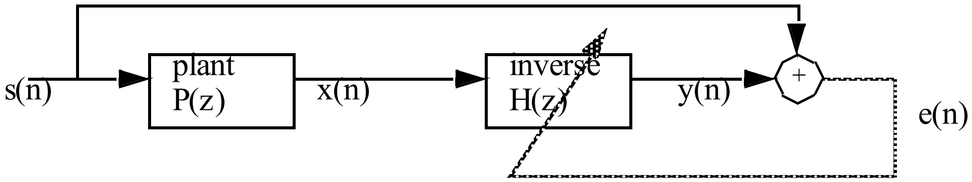


Figure 7. Inverse modeling.

Assuming that the output error is small, we can see that y(n) is quite similar to the input s(n), i.e. that the combination of the unknown plant and the model cancel each other. In terms of transfer functions this means

$$Y(z) \approx S(z) = H(z)P(z)S(z) \rightarrow \qquad H(z) = \frac{1}{P(z)}$$

Equation 29

Inverse modeling with a linear combiner produces an autoregressive model (AR) of the plant. So the modeling is perfect only if the plant is an all-pole system and the model has the same size as the plant. Otherwise, the overall plant input-output can only be approximated with the inverse model, and may require large model orders for reasonable results. Another concern is the type of input. In practice we would like to have an input that is able to excite all the important modes of the plant, otherwise they will not show up at the output which mean that they will not be modeled. A practical approach is to add small amounts of white noise to the plant input, which is called “dither” noise.

NeuroSolutions 11

9.11 Inverse modeling

Let us start with a very simple case of a first order recurrent system as our plant. This case can be perfectly identified by a two taps FIR filter (just remember the transfer function of the first order recurrent system and look at the denominator which is the target for modeling). Take note of the value of the recurrent parameter.

Use a square wave at the input to cover well the spectrum. Let the system adapt and notice that the error goes to zero, i.e. we obtained perfect modeling. Read back the coefficients and observe that they match exactly the recurrent system parameters.

Now let us use a more complex signal generation model. We will use an ARMA model, i.e. a system that has poles and zeros, here one pole and one zero. The pole can be exactly matched with the linear combiner, however, the zero of the model can only be modeled by a system with poles and so the linear combiner can only approximate the plant model. We know that in principle an infinite linear combiner order is needed, but in practice the error here is very small after an order of 7 or so (it depends upon the value of the feedback coefficient - try several and observe the dependency).

Now change the input (and desired signal) to a sinewave and verify that the error goes quickly to zero, however the coefficients are different from the previous values. Disable learning in the controller to fix the weights, and modify the input back to a squarewave. The waveform is distorted, i.e. the identification was not successful.

The reason can be found in the inappropriate excitation of the plant (the output contains only the response to a single frequency) which makes the inverse problem very difficult to solve (it is easy to find weights to match the amplitude

and phase of the sinewave, but this does not mean that the full plant response has been identified). Add noise and observe that the coefficients approach the ones of the plant.

One could think that the impulse is a good input signal since it has all the frequencies. But effectively this is not so when a sample by sample adaptation is used to modify the system parameters since in the time domain the waveform is impulsive. This means that the weights will be jumping up and down what is bad for a smooth convergence to the optimal values. White noise has also a flat spectrum and it is a much better choice for adaptive systems since the time waveform is a blend of all frequencies at all times. Try these two choices with this breadboard.

NeuroSolutions Example

In practical cases we have to handle two conditions. First, the plant being a physical system may include a delay and there may be plant noise (Figure 8).

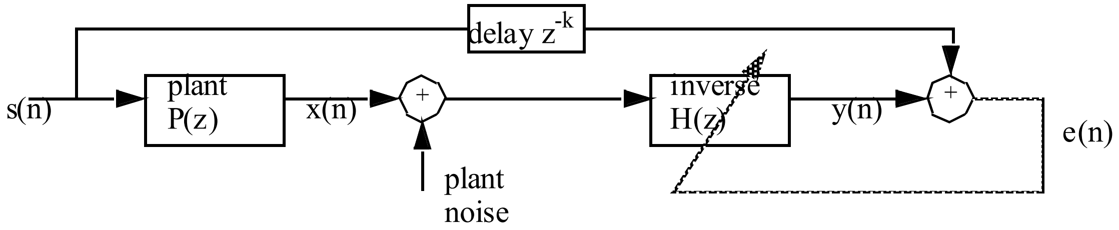


Figure 8. Practical inverse modeling for plants with noise and delays.

Nevertheless the same basic scheme can be utilized except that we include a delay in the forward path, i.e. the desired signal is delayed by k samples. A good rule of thumb to set k is one half of the model order. The inclusion of plant noise may affect seriously the determination of the inverse, but when the noise is white reasonable results can still be obtained.

NeuroSolutions 12

9.12 Inverse modeling of non-minimum phase plants

This is basically the same breadboard, but now the system is a little more complex. It is now an ARMA (autoregressive moving average) system with a transfer function

$$H(z) = \frac{a + bz^{-1}}{1 - cz^{-1}}$$

First make a=1, b=0.5 and c=0.5 as before. Let us input a square wave and create a model with two taps. As you can see the MSE is quite high and the output waveform is far from the square wave. Now let us increase the order of the model to 5. Run the example again and see that the error decreased quite a bit and the output is much closer to the square wave, even with the white noise added to the input. We can increase the model even further (order 10) but notice that the improvement is not as drastic. The output of the linear combiner has many ripples so although the MSE is slightly better in terms of waveform fitting the improvement is questionable.

Turn off the noise. Go back to 5 taps and now let us change the FIR coefficients to b=1.5. Note that the pole is the same and the zero is in the mirror location with respect to the unit circle, i.e. the frequency response of the previous and of this system are the same. Run the example and observe that the response is much worse than before. What happened? Looking closely at the input scopes you will notice that now there is an appreciable delay between the output and the input waveform, much larger than before. Effectively we have created a non-minimum phase plant. So the linear combiner is faced with a much harder problem since its desired response rises first than its input. So we are asking the system to be antecipatory, which is impossible for any causal system.

This is where a delay between the input to the cascade and the desired response as discussed in the text makes sense. Let us just delay the desired response by creating a phase of -18 (2 samples). Run the example and verify that the MSE is

again low and the output looks much closer to a square wave. Put the noise back on and experiment with the order of the model.

Notice how much harder inverse modeling is when compared to the direct modeling of the previous example.

NeuroSolutions Example

Important applications of inverse modeling are adaptive equalization of communication channels and adaptive inverse control.

Go to next section

6.6. Interference canceling

This application is very useful when the time series of interest is disturbed with an additive narrowband interference such as the power line 60 Hz frequency (50 Hz in Europe). In the modern world we are surrounded by power lines and so the 60 Hz interference is pervasive. Biological signals are very weak (from microvolts to millivolts) and the body works as an antenna at these frequencies, so data collection in medicine is very vulnerable to the 60 Hz noise contamination. The goal is to find out a way to filter out the 60 Hz from the collected signal. Traditional filtering ideas would develop an optimal filter with a single input which would allow the signal through and attenuate the noise. Interference canceling requires an extra collecting probe that will just receive the 60 Hz signal from the subject. The idea is to use an adaptive filter to subtract the 60 Hz from the collected physiologic signal (Figure 9).

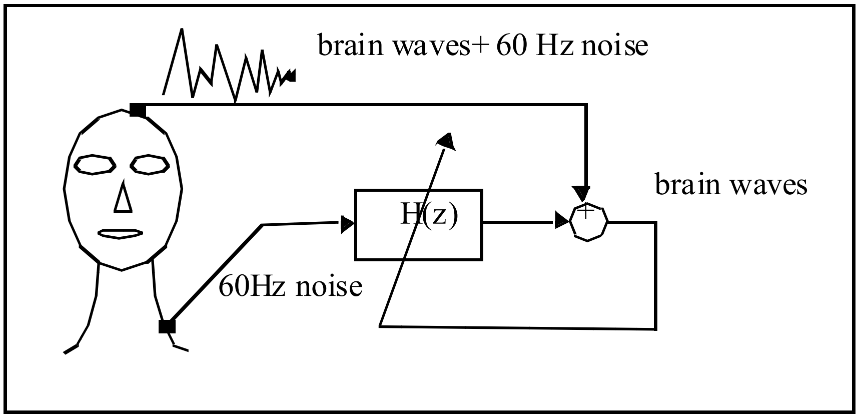


Figure 9. Removal of 60 Hz interference from brain waves

For narrowband interference the adaptive filter H(z) only requires two weights, because its job is to find the best amplitude and phase of the 60 Hz noise reference input to match the 60 Hz noise collected with the brain waves probe. Assuming that the brain waves are uncorrelated with the noise, the MSE will be minimized when the noise is removed from the brain waves.

Sometimes, no other probe is available to collect the 60 Hz signal. But we can artificially generate 60 Hz and assume that the line frequency is sufficiently close to 60 Hz (the line frequency varies with the load on the power system). It can be shown that in this case the adaptive filter (after adaptation) is simply a constant coefficient notch filter, but here we have not designing the coefficients explicitly, we used the power of adaptation to automatically find the weights.

NeuroSolutions 13

9.13 Interference canceling

We start with an example with synthetic data to explain the methodology. Let us assume that we have a square wave signal that got added with a sinewave of a known frequency but we do not know the phase nor the amplitude of the sine. How can we subtract it from the square wave?

One idea is to create a two tap adaptive filter (AF) that receives as the input a sine wave of the same frequency as the interference. This filter receives as its desired

response the mixed square + sinewave signal. What do you expect the adaptive filter will do? It will try to modify its weights such that it will produce at its output a sine wave of the same phase and amplitude as the interference. This is the best the AF will be able to do to decrease the difference between its output and the desired response. But notice that when the filter adapts to this solution, the error becomes very close to the square wave that we would like to recover. This is the principle of noise canceling with adaptive filters.

So let us create a simple breadboard with a two tap linear combiner that receives as input the sine and that gets as the desired response the square wave with the sine interference. We place a Megascope on the breadboard such that we can see the error, desired response (mixed signal), the input and the output of the adaptive filter (the signals will be displayed by this order). We also placed a FFT to show the spectrum of the error and of the output. Notice the system adapting. The error begins very close to the desired response, but when the system adapts the error approaches the square wave because the system output approaches the sine wave added to the square wave. This can also be seen in the spectrum of the error which starts with double peak in low frequency and approaches the spectrum of the square wave towards the end of adaptation.

This example is illustrative of the power of adaptive filters for noise canceling, but notice that this only works well when the interferences are narrowband signals that hopefully do not overlap with the signal spectrum. To see how crucial this is, just modify the frequency of the sinewave to coincide with that of the square wave (you have also to change the frequency of the input of the filter to match the expected new frequency). The filter now removes all the energy at the fundamental frequency of the square wave as can be seen in the spectrum of the error. Hence the error is no longer the expected square wave.

The EEG (electroencephalogram) is a signal collected with electrodes from the

scalp which monitor brain activity. The signal is rather tiny (in the range of a few to hundreds of microvolts) and has a spectrum between 0.5 to 50 Hz. Very high quality (low noise and high common mode rejection ratio) amplifiers are required to amplify the EEG, but even the best amplifiers are unable to successfully remove 60 Hz interference. The reason is that the body is an excellent antenna for 60 Hz created by the power lines. So either we collect the EEG in shielded rooms (which are very expensive) or we have to find a signal processing way to cancel the 60 Hz interference. This is where adaptive filters can be utilized as explained in the text.

Here we have a segment of the EEG with 60 Hz interference. The signal is sampled at 500 Hz so a 60 Hz wave will have approximately 8 samples per cycle. Since we did not collect any other channel, we will artificially generate a sinewave of 8 samples per cycle to input to the adaptive filter. We will use the previous breadboard. Run the system and observe how the EEG that is very irregular becomes cleaner. Observe the output of the system which basically only has a sinewave.

NeuroSolutions Example

The same basic idea can be used to filter for instance d.c. from a data set, i.e. create an highpass filter. For this case the input to the adaptive filter (a simple adaptive constant is sufficient) is a d.c. value such as the value 1. The value of the step size controls the cut off filter of the highpass filter.

NeuroSolutions 14

9.14 Adaptive dc removal

This example is just another application of the cancellation idea that may be very useful in practice for preprocessing. Many times the data we collect from the outside world has a DC bias due to instrumentation problems, and we would like to use just the AC component. This is particularly true for filtering since a lot of

signal processing algorithms assume zero dc data (such as PCA).

If we use a single tap adaptive filter with a constant value at the input, and our data presented as the desired response, we will effectively create a dc remover, i.e. a highpass filter.

Notice that the selection of the stepsize will also define the cutoff frequency of the filter. This is clear from the visualization of the spectrum of the filter output. Let us now modify the stepsize to see the effect on the operation. If we increase the stepsize, the output of the filter becomes more and more like the EEG! This is due to the fact that the stepsize effectively controls the bandwidth of the adaptive filter. We can understand this if we think that with a larger stepsize the system can change the weights faster, so it will be able to subtract more energy from the desired response. On the other hand, if we decrease the stepsize, the filter will take a little longer to converge, but only gets the DC value. We can see this easily in the spectrum of the filter output.

In neural networks, the stepsize was just controlling how fast the adaptation was happening, but in adaptive filtering the stepsize is also controlling the bandwidth of the system (since the adaptation of the adaptive filter happens in time and so there is a physical meaning to how fast the weights adapt).

NeuroSolutions Example

Another interesting application of interference canceling is the removal of noise from speech in high noise environments such as factories, plane cabins, etc. The idea is basically the same. The noisy speech is collected with a microphone providing the primary signal source, and an extra microphone captures only the noise.

In these more general cases the adaptive filter has to be of larger size since the noise source is normally far from a narrowband signal. For each two taps the filter is able to suppress an harmonic of the interference. The method is effective when the noise conditions are regular (stationary) or vary slowly and they have marked harmonic content (such as noise produced by rotating machinery). In cases of broadband noise, only the low frequencies are reasonably canceled.

NeuroSolutions 15

9.15 Real example of noise canceling

This example shows the power of interference canceling applied to more complex signals, speech and a washing machine noise. Our speaker is Homer Simpson, and his voice mixed with the washing machine is presented to the desired response of the system (s+n). At the input of the linear combiner we present just the noise (n). The linear combiner is a specified of size 5 since the washing machine noise has a broader spectrum, but still has a salient peak. The response of the linear combiner will try to cancel the noise in the desired response leaving only Homer's voice. We present the signals and their spectra on scopes for visualization of the progression of learning.

After the presentation of the speech segment we are going to listen to the error which will contain basically Homer's voice. We attached a DLL to the error output of the criterion which calls the Windows 95 routine that plays sound files. Notice how in the beginning Homer's voice is mixed with lots of noise, but after a few iterations it becomes intelligible. Can you understand what he is saying?

NeuroSolutions Example

The same principle can be applied to different problems, for instance, the enhancement of low-level sinewaves in high noise backgrounds. Only one signal source is utilized (Figure 10). The signal is split into the primary input and the reference input which is delayed to decorrelate the noise but keeps the correlation of the signal since we assume it periodic. One can take the DFT of the filter coefficients to clearly see the sinewave and its frequency.

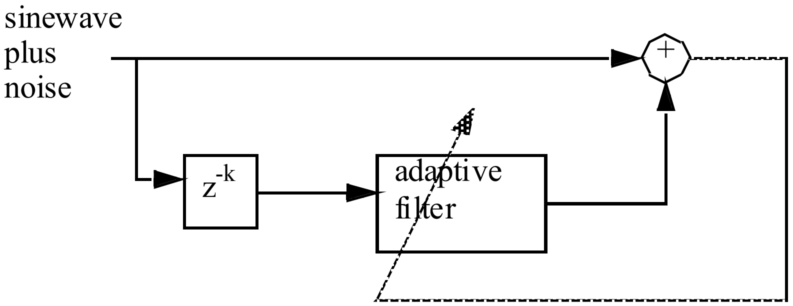


Figure 10. Line enhancer with the LMS

NeuroSolutions 16
9.16 Finding periodic signals in noise

This example explores the same basic principles of noise cancellation but the application is different. Here we would like to find weak sinewaves in high noise backgrounds which may be correlated. We will be delaying the input from the desired response to decorrelate the noise. However this will not decorrelate the signal since the sine is still correlated from period to period (it is periodic). So the adaptive filter will try to correlate the desired response with its input. Although we are searching for a sinewave (2 weights should suffice) we recommend to use a slightly larger model (4 to 6 weights) since the samples will be very noisy.

Experiment with the delay, the learning rate and the filter size and observe the differences in the spectrum.

NeuroSolutions Example

Go to next section

6.7. Echo-cancellation

Echo cancellation with adaptive equalization are probably two of the most widely applied interference canceling schemes due to its application in communications. We are going to explain echo cancellation separately. It turns out that our telephone channels are not ideally terminated, i.e. signals when they travel from the source to the destination are reflected back to the source. This is the familiar echo that we can hear in some long distance phone calls. The echo is annoying and decreases the intelligibility of speech. So the phone companies built special devices called echo cancellers to attenuate the echo.

The culprit of this problem is a transformer (called the hybrid) that receives a two wire input from the phone and creates a four wire output (the long distance trunk) to send our conversation multiplexed with others (and also to amplify them). The net effect is that when the caller A speaks, the caller B receives the voice, but there is a return path from the hybrids of the callers A and B hybrids that inject caller A voice in the outgoing port and it is bounced back to caller A phone, appearing delayed in time (Figure 11). For satellite communications this delay for the long echo path reaches 400 milliseconds and is very annoying (the local echo is not disturbing because our ear filters it automatically - the precedence effect). When caller B talks the same thing occurs but in the reverse direction.

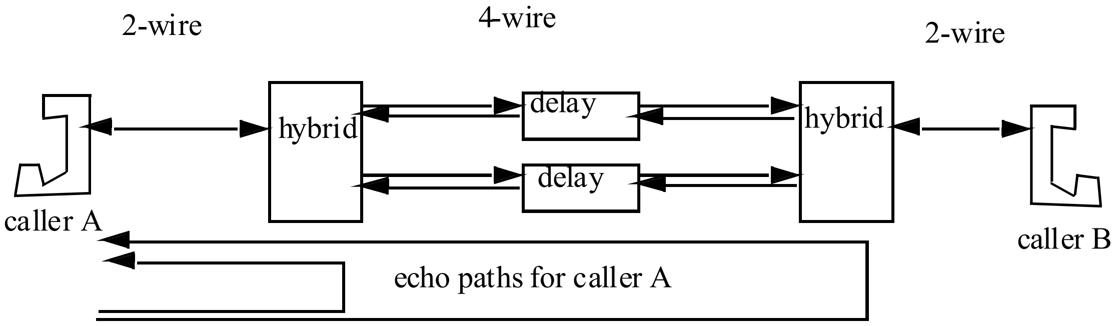


Figure 11. The source of the echo in the phone lines.

A possible implementation of an echo canceller is to use an adaptive linear combiner in parallel with each hybrid as shown in Figure 12 to subtract the echo from the incoming signal. The adaptive filter at the left filters the echo for caller A, and it is effectively identifying the system composed by the channels and the hybrid of caller B. For a sampling rate of 8 KHz a linear combiner with at least 128 taps is required.

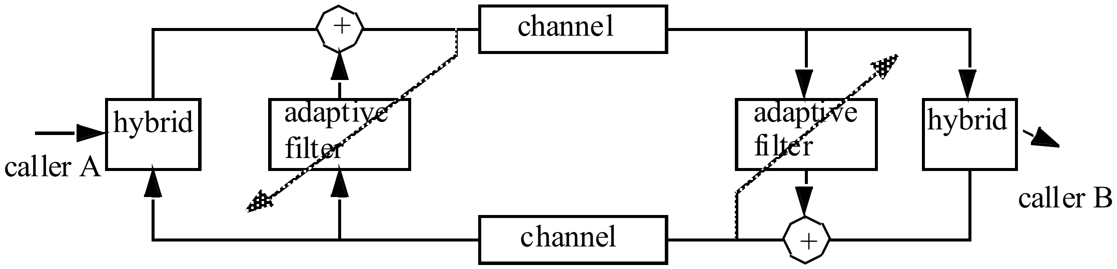


Figure 12. Echo cancellers using adaptive filters.

More recently, with the widespread use of the modems in our homes and offices for data transmission a similar problem occurs, but now the disturbing echo is from reflections in the local subscriber loop (the short path echo). The echo is much more troublesome in data transmission because the data rates are very high which means that any modification of the shape of the waveforms (as produced by adding a zero with a one) can be interpreted as the wrong symbol and so the message must be repeated.

Adaptive filters can also be used to attenuate the digital echo quite effectively and any modem that operates above 4800 Baud has a built in adaptive filter (Figure 13). The idea is to have the filter subtract the echo by learning the echo path transfer function. So the signal that goes back to the modem is basically echo free.

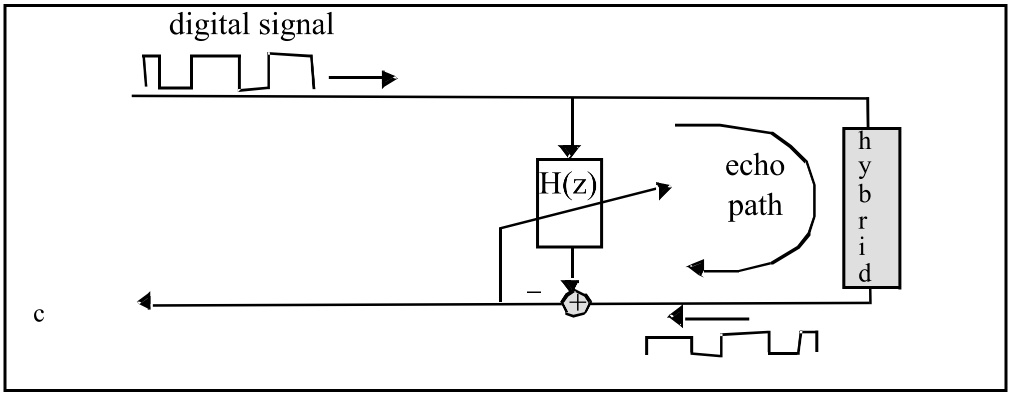


Figure 13. Echo-cancellation with adaptive filter

The efficiency of the echo canceller is measured in ERLE (echo return loss enhancement) which basically computes how much the echo was attenuated.

NeuroSolutions 17

9.17 Echo-cancellation

The data used in this example is fabricated, but is realistic of conditions found in the real world. We generated a pulse train to model a bit stream created by a modem. This signal is then passed through a transfer function that models the lossy path of a 2-4 wire transformer as

$$H(z) = \frac{0.1(1+z^{-1})(-1+z^{-1})^2}{(1+0.3z^{-1})^3}$$

The output signal is then added to the incoming signal to the modem which here is modeled by a simple square wave. So instead of receiving just the square wave the modem would see the addition of the square wave plus the echo of the outgoing bit stream.

The role of the adaptive filter is to subtract the echo from the square wave. In order to do this, the adaptive filter will receive as the input the bit stream send out by the modem, and will try to figure out the echo (identifying the hybrid lossy transfer function). We have placed Megascopes and FFT probes to observe the adaptation process.

Notice that the advantage of this technique is that it will always adapt to changes in the echo transfer function, which are known to change from line to line and with the load in the lines.

<u>NeuroSolutions Example</u>

Go to next section

6.8. Adaptive Inverse control

Another important application of adaptive filters is in control engineering. The idea of controlling an unknown plant is to design a system (controller) that will produce a prespecified plant output.

Probably the most widely used method of plant control is by feedback. The controller receives the command input and provides an input to the plant. The plant output is measured and fed back to the controller such that it can relate changes in the plant input with the changes produced at the plant output. Feedback control is useful but the applicability of the method is not always straightforward due to physical constraints. Other methods exist such as the one that uses inverse modeling (Figure 14).

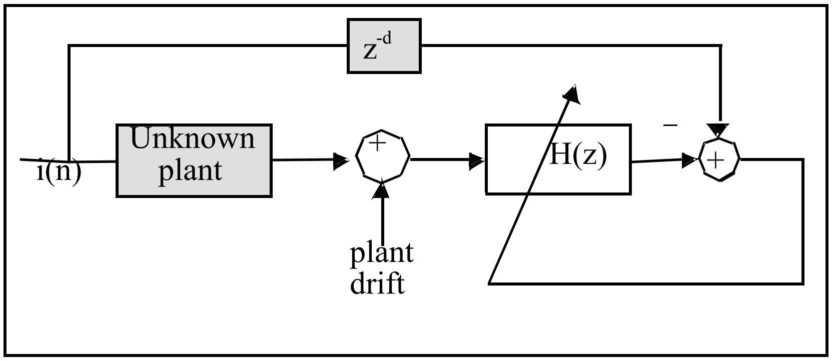


Figure 14. Inverse modeling with plant drift and a delay

In inverse modeling the adaptive filter H(z) is placed in cascade with the unknown plant, and the output of the combination seeks to match the input signal (or its delayed version). This block diagram is basically inverse modeling that we described in section 6.3. If we write the equations when the error is a small constant, we see that the adaptive filter transfer function is the inverse of the plant. The delay is necessary because the plant may either have delays or be non-minimum phase, i.e. may have zeros outside the unit circle which may cause the system model to be unstable (not in this case since we are using a linear combiner). The inclusion of the delay stabilizes the inverse model.

Now with the inverse model available, we can construct a controller between the command input and the plant just by duplicating the adaptive filter used in the inverse modeling and copying its weights as shown in Figure 15.

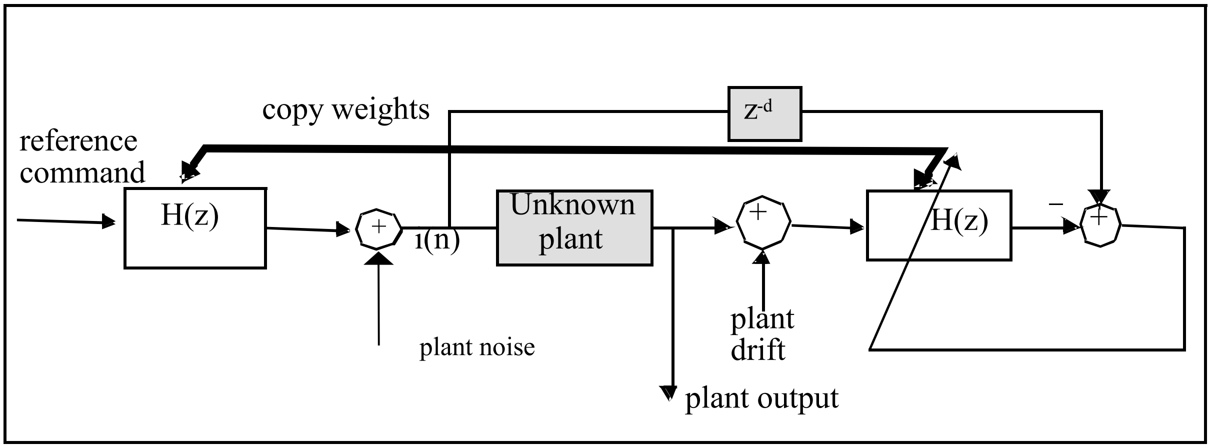


Figure 15. Adaptive inverse model control

With this arrangement the plant output will closely follow the command input applied to the controller since we “predistort” the command to the plant by the controller. This predistortion is exactly what the plant needs to follow the command input. This theoretical picture is complicated by two primary factors: non-minimum phase plants and plant drift, which can be handled, but will not be discussed here.

NeuroSolutions 18

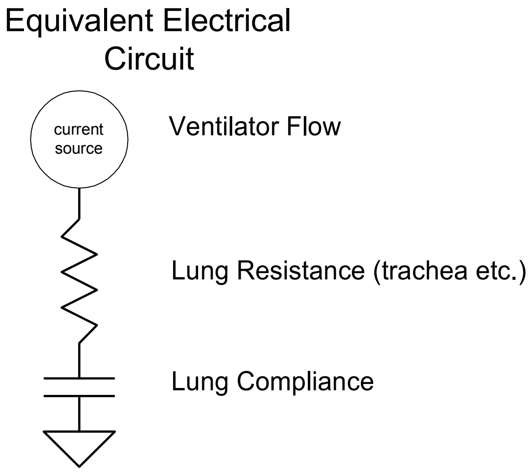
9.18 Adaptive inverse control

Here we will use as a plant a realistic model of the lung of a patient under anesthesia. The controller can be thought of as a mechanical ventilator which helps the patient breathe. We will use adaptive inverse control for this task.

Humans breathe with negative pressure by exerting their diaphragam and chest muscles to suck in air. Mechanical ventilation is positive pressure, actually blowing air into the lungs and it is almost exclusively controlled with a proportional flow control valve (PFCV) which controls the flow rate of the ventilator.

The model of the lung implemented here is a simple first order dynamical system modeling the lung compliance (i.e. relating air pressure to the lung volume). When the left and right compliances and resistances are equal (which is the what we are

using), the system can be modeled as follows:



where the goal is to control the pressure at the patient airway (at the end of the ventilator tubing). There is resistive pressure which is a function of the flow (when flow ceases there the resistive pressure goes to zero) and compliant pressure which is based on the amount of air in the lung (volume = flow times time) and the compliance ("stretchiness") of the lung which yields

$$V = \frac{1}{C}\int Idt + IR$$

The purpose of the ventilator is to apply a square wave of pressure to the lungs. However, due to the lung compliance and resistance, what should be the actual pressure waveform produced by the ventilator?

This is exactly what our implementation will figure out! The input to the controller will be the square wave of pressure (we will just model the rising slope here). The output of the controller will be the lung, here modeled as an Axon. The output of the "lung" is then fed to the adaptive part of the controller which will identify the inverse lung model. Its parameters will be copied to the controller which will pre-distort the command input such that the desired behavior is obtained.

This scheme can even be refined to include a reference model that will relax the condition that the plant input and the desired response of the inverse model are the same (eventually delayed) as explained in inverse modeling.

Now let us see how we can implement such a complex scheme in NeuroSolutions. First there is the controller and its function. We use a linear combiner followed by a logistic nonlinearity. The need for the nonlinearity arises because the controller can only produce positive signals (positive pressure of the ventilator). The weights of the controller are copied from the inverse model that follows the plant. A DLL transmits the coefficients. The input to the controller is a square wave added with white noise. The reason we use white noise is to provide always a driving force for the adaptation (the square wave is featureless except during the jumps, and just like the sinewave of our previous example will not be sufficient to drive all the modes of the plant).

The lung dynamics given by the above equation are here modeled in the Axon. The Axon is an input-output device, so through DLLs we can create in C arbitrary complex programs that model real systems. For each input to the Axon the C program will create the corresponding model output.

The inverse model is created by a linear combiner followed by the same logistic nonlinearity. The desired response to the inverse model is received from the input to the lung through a DLL. Let us see how this works.

NeuroSolutions Example

Go to the next section

7. Applications of temporal PCA networks

The ideas and learning algorithms covered in Chapter VI for Hebbian networks can also be extended to time signals with minor modifications in the topology. We will present here an example of an eigenfilter, the decomposition in principal components of a discrete signal, blind source separation of speech signals, and one example of a subspace method for sinusoid detection in broadband noise.

Go to next section

7.1. Maximum Eigenfilter

Suppose that we have a signal that is generated by a subtraction process where some frequencies are eliminated which happens for instance in the force signal collected from the cutting head of rotating machinery. Such signals have spectra with a large concentration of energy in several ranges of frequencies but have nulls which show their moving average (MA) generation. Instead of applying the spectral analysis method indicated above using prediction which implies an AR model, we will use here temporal PCA*. The PCA in time produces MA models that seek the peaks of the spectrum*. The single output system is called the maximum eigenfilter and will seek the largest peak in the spectrum, but assumes at the same time a moving average model. Basically the maximum eigenfilter finds a projection vector in the input space where the output power is maximized (see Chapter VI discussion).

We can train a temporal PCA network with a single output using the Oja's algorithm to solve this problem. The output will be the largest eigenvalue over time, while the filter weights correspond to the eigenvector. The larger the delay line the more coefficients the filter will be and hence the sharper will be the eigenfilter main lobe.

NeuroSolutions 19
9.19 Maximum eigenfilter

Let us create a synthetic signal to exemplify the largest eigenfilter model, and then apply it to real world data. We will create a signal through a subtraction process. Consider a pulse of 4 nonzero samples (-0.2, 1, -0.5, -0.3) which repeats itself every

15 samples. Let us now add 4 versions of the same signal but delayed in time by equal amounts, and add noise. We can consider the pulse produced by some rotating machine with 5 teeth, and what we measure is the overall waveform.

The spectrum of this signal is full of peaks, and the goal is to find what is the fundamental component. We can use the maximum eigenfilter to solve this problem. Let us create a delay line with 15 taps and use the Oja's rule to train the system (just one output). We will see that the output grabs the largest peak. Now we will freeze the weights (just by setting the learning rate to zero), and will send an impulse through just to compute the system frequency response. Since this is a FIR filter (implementing an MA model) we expect to see a frequency response with a large lobe and many sidelobes as sculptured by zeros. We can increase the number of taps in the delay line and see that the output spectrum gets sharper as we would expect from the theory.

The second example is to extract the fundamental component of the sunspot time series. As we have seen the time series is very noisy but it has a quasi periodic oscillation. Let us use a tap delay line of 20 and train the eigenfilter with Oja's rule. We will see the waveform progressive becoming cleaner showing that the high frequency information is being removed. If we freeze the weights and send a pulse through we will see the frequency response of the lowpass filter which was automatically obtained to preserve the most information about the largest mode in the spectrum. This type of processing can be considered pre-processing for the subsequent analysis, and it is one of the attributes of eigenfilters.

NeuroSolutions Example

Go to next section

7.2. PCA Decomposition in time

If the previous network is extended with more outputs (Figure 16) and the Sanger's rule is utilized in the training, then a PCA decomposition in time will be achieved.

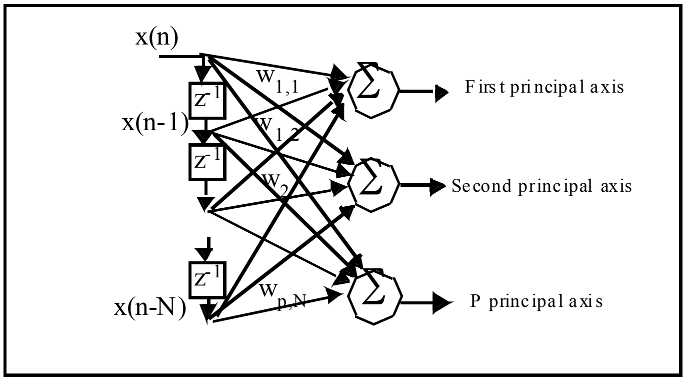


Figure 16. PCA in time network

The size of the time window limits the accuracy and the size of the autocorrelation function estimator. The number of output PEs establishes the number of principal components extracted. It is important to analyze the PCA in time as a bank of optimally derived filters.

When the input signal has a lowpass spectrum, i.e. the maximum power is concentrated at low frequencies, and there is a monotonic decay of power across frequency, the PCA in time defaults to a bank of optimally designed bandpass filters, which for very large windows approximate the spectral decomposition achieved with the FFT. In temporal PCA pairs of neighboring PEs will tune the same basic band since a given frequency band can be covered by two orthogonal components of the same basic frequency (as the sine and cosine). As we can expect the bandpass filters are FIR, i.e. PCA in time is implementing a moving average (MA) model.

We can easily observe the frequency response of each eigenfilter if we stop training and input a delta function to the system. The analysis of the decomposition is not straightforward except for the limit of long windows or for signals that have a line spectra such as the square wave. Closely analyzing the solution for this case we see that the zeros of lower order eigenfilters are placed over the top of the peaks of higher order eigenfilters, which can be expected from the orthogonality condition of the PCA components.

When the input signal spectrum is not lowpass or/and the limit of long windows is not verified, then this simple analysis does not hold anymore, since the principal component will grab the direction corresponding to the largest spectral mode (which are broad and may be anywhere in the spectrum). The lower order eigenfilters have to compete to the other spectral modes, but now there is no guarantee that the eigenfilters will be unimodal and will be ordered. Nevertheless, the system will still find the best (ordered by variance) orthogonal projection of the input, and effectively the complication is just one of interpretation, and it is not reflected in the learning.

NeuroSolutions 20
9.20 PCA decomposition in time

Let us use the square wave as the example of a signal that has a lowpass, monotonically decreasing spectrum. If the PCA in time decomposition is implemented with Sanger's rule, we will see that the system self-organizes in a fashion that pairs of eigenfilters grab one harmonic of the square wave. If we stop the training and estimate the frequency response, we see that the eigenfilters effectively are organized in pairs, and of increasing frequency. Note also that the zeros of the higher order eigenfilters are placed on top of the preceding eigenfilters bandpass frequencies.

Now let us change the input to a speech segment of the sound "a" as in "hot". The first three formants should appear at f1= 730 Hz, f2=1090 Hz, and f3=2440 Hz. We will use a PCA network with 20 input taps and 6 outputs. After training you will see the two first eigenfilters peak at the first formant, the second two at the second formant, and the last 2 eigenfilters picking up some low frequency activity but also some higher frequency activity. If we change the FFT display to log, we will be able to see more clearly this feature. Notice that this decomposition produces very interesting features in the time domain.

<u>NeuroSolutions Example</u>

Go to next section

7.3. Spectral analysis by eigen-decomposition

Spectral estimation based on autoregressive models looses accuracy rapidly when the noise in the data increases. The eigenvectors of the autocorrelation function can be used to find sinewaves in white noise. The idea is that in the eigen-decomposition of the autocorrelation function, the input signal energy is divided into orthogonal components. A common problem is to find a low level sinewave in noise. Since sinewaves are the natural modes of the PCA decomposition for long observation windows, one of the eigenvectors will be associated with the sinewave, while the noise will project everywhere.

NeuroSolutions 21

9.21 Noise removal by PCA

Let us utilize the previous breadboard but now change the input source to a single sinewave with much larger additive noise. We will decrease the number of outputs to 3 to show exactly what we said in the text. The number 3 comes from the fact that a sinewave will appear in two eigenfilters, while the third one will simply show the noise. Use a learning rate of 0.001.

We will start with the sinewave amplitude of 1, 10 samples per cycle, and add uniform noise of variance 2. The sinewave is only faintly visible in the time signal, but spectrum still shows a very clear peak. The network has no problem of providing a nice sinewave in the first two eigenfilters. If we now decrease the amplitude of the sine to 0.3, not even in the spectrum we see the sinewave, but the system still trains to the sine. For an amplitude of 0.1 the system is unable to find the sinewave, i.e. there is a limit to this technique. We can increase the window to 50 samples and decrease the learning rate. Most of the times we can see in the spectrum of the first eigenfilter a spectral line that can be traced to the sinewave.

NeuroSolutions Example

This same idea can be extended to multiple sinewaves in noise or man-made signals. The directions that will contain only the white noise components will generally correspond to smaller eigenvalues. So if we truncate the eigen-decomposition when there is a large drop on the magnitude of eigenvalues, we will retain the signal components (signal subspace) and attenuate the noise. The big problem is to determine accurately the frequency of the faint sinusoids as we can appreciate from the above example.

An alternative uses only the noise subspace to find the sinusoidal components. The noise subspace is orthogonal to the signal subspace. So if we use the minor component (last eigenvector) of the PCA as a filter, we are guaranteed that it will have zeros in the unit circle where the signal components are located. We can copy the weights of the minor component to a recurrent system implementing the inverse system, whose frequency response will have peaks where the sinusoids are located (since the zeros will give rise to poles of the inverse model). We already have done this when we discussed model based spectral analysis, except that here we are using as a filter the eigenfilter that corresponds to the noise subspace. It has been shown that this method is highly accurate for determining the frequency of sinusoids (the MUSIC algorithm). One of the problems of this method is that it requires knowledge of the number of sinusoids included in the signal.

NeuroSolutions 22
9.22 Spectral analysis by eigen-decomposition

We can use the previous breadboard to train the PCA in the sinewave plus noise case. We will use the anti-Oja which is simply anti-Hebbian normalized. Hence the network will have a single output and we will select a 5 tap network. In principle we only need a 3 tap filter but the lack of resolution will be noticeable. We will train the system using the scheduler for the stepsize. Once the PCA has converged, we will copy the weights of the weight matrix and call it H(z). These weights represent the eigenfilter that represents the noise subspace, and its frequency response is guaranteed to have zeros at the frequency of the sinusoid embedded in noise.

According to the theory of noise subspaces, the spectral estimator should be 1/H(z).

In order to create this transfer function we will create a recurrent system in NeuroSolutions by feeding back the output of a linear combiner to itself. The recurrent system transfer function is $\frac{1}{1-z^{-1}G(z)}$ **where G(z) is feedforward transfer function. In order to obtain the spectral estimator, with this breadboard, we have to make G(z)= 1- H(z). This can be accomplished easily if we exchange the signs of all the coefficients of H(z) and add 1 to the leading term (i.e. the coefficient for lag zero). Run the network and observe that the spectrum peaks at the desired frequency. With this method the frequency can be estimated very accurately.**

NeuroSolutions Example

Go to next section

7.4. Blind Source Separation

This is an important problem in science and in engineering. It can be formulated in the following way: Suppose that we have M sources assumed statistically independent (this is the only assumption we need to make) which are unknown and mixed by a linear but unknown system. Here we will only deal with the case that the mixing is instantaneous, i.e. the system is an unknown mixing matrix. We do not know the sources nor the mixing matrix, but we collect M signals from the output of the mixing system (i.e. we have as many mixed signals as sources). The goal is to find a system that is able to de-mix the signals. This is a difficult problem since we do not know the sources nor the mixing matrix.

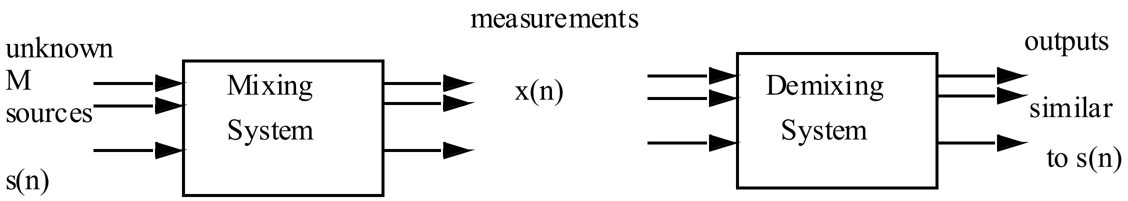


Figure 17. The problem of blind source separation

To relate to this problem, let us assume that we are in a room with many people talking at the same time (called the cocktail party effect). Our hear/brain is able to focus on a given speaker and individualize him/her voice from the others. Our senses are basically doing blind source separation because we do not the source (the speech is unknown to us) and the room acoustics and the placement of the subject with respect to the others and to us is unknown. Humans can even do it with just two ears…. But here we will assume that we have as many measurements x(n) as sources.

We will use the ideas of anti-Hebbian learning coupled with a tap delay line to create a system that will be able to de-mix the signals. How can this simple idea be used to perform blind source separation? Let us for the sake of simplicity use just two sources. We just have to use a recurrent system built from two Anti-Hebbian networks as the ones in the Figure 18.

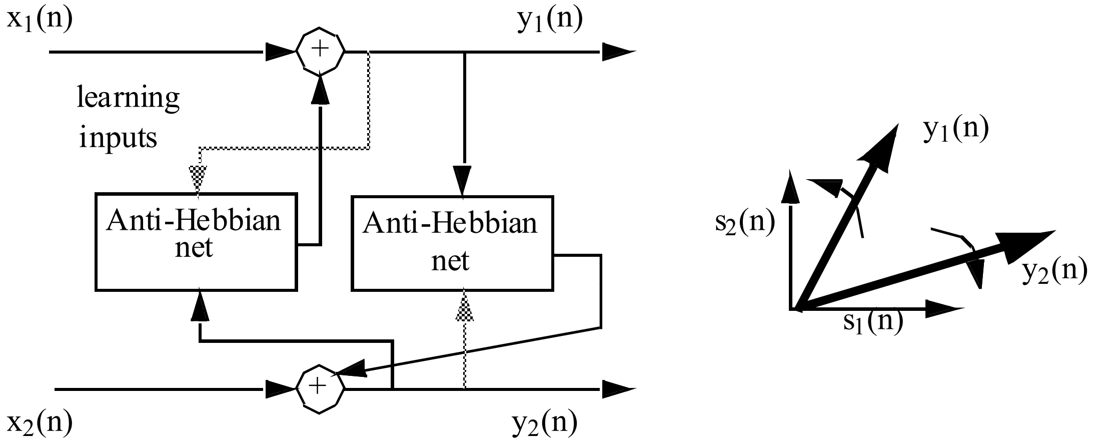


Figure 18. The solution based on novelty filters.

By assumption $s_1(n)$ and $s_2(n)$ are orthogonal (since we assumed they are independent).

When we start the adaptation using anti-Hebbian. $y_2(n)$ will push $y_1(n)$ away from it, while $y_1(n)$ will make $y_2(n)$ rotate away from its direction. Just by looking at the figure they will stop the rotation when $y_1(n)$ is orthogonal to $y_2(n)$, which coincides with the directions of $s_1(n)$ and $s_2(n)$ (there are other solutions to this problem which for nonstationary signals seem not to happen in practice too often). This is the principle that we will utilize to do blind source separation. Note however, that there is no guarantee that the solution found are the original signals.

NeuroSolutions 23

9.23 Blind source separation with anti-Hebbian learning

This breadboard is complex not because of the task, but because of the implementation in NeuroSolutions. There are two identical parallel channels which are interconnected as shown in Figure 18. Each is a tap delay line followed by a set of weights which are adapted using anti-Hebbian with normalization. The normalization to unit length of the weight vector is necessary to avoid convergence to the trivial solution (zero weights). This is accomplished by writing a DLL for the adaptation.

The first two Axons in each channel are there just for synchronization purposes. The two mixed signals are applied to the input of the network and the output is plotted in scopes. We also create a sound file from the top channel such that we can appreciate the evolution of the demixing while the system is training. In the beginning is impossible to understand the sentence. After 15 iterations over the data set, this system is able to demix the two speech signals.

Notice that filtering would be impossible since the two speech waveforms share the same frequency band. The demising are using the statistical properties of the two waveforms to achieve the separation, without knowing the sources nor the mixing matrix.

NeuroSolutions Example

Go to next section

8. Conclusions

In this Chapter we covered adaptive filters. We showed that an adaptive filter is a linear regressor in signal space, i.e. it tries to fit a line to a signal trajectory. We showed that we can adapt linear filters with our well known LMS procedure.

Another class of linear systems that we covered are the temporal PCA networks. Borrowing from Chapter VI the ideas of PCA we can bring them to time by including a tap delay line in the PCA architecture. Temporal PCA basically computes eigendecompositions now in time. We can therefore design maximum eigenfilters, novelty filters, and KLTs.

Probably the most important part of Chapter IX is contained in the application section. Here we tried to show many possible uses of adaptive filters and their usefulness in digital signal processing. We cover a wide range of applications from prediction, system identification, noise cancellation, to inverse adaptive controls. We also showed how the temporal PCA networks can be used for eigenfilters, in spectral analysis, and in blind source separation.

NeuroSolutions Examples

9.1 Design of filters using adaptation
9.2 Adaptive filters for prediction

9.3 Chosing the stepsize
9.4 Speed of convergence

9.5 Misadjustment

9.6 Tracking ability

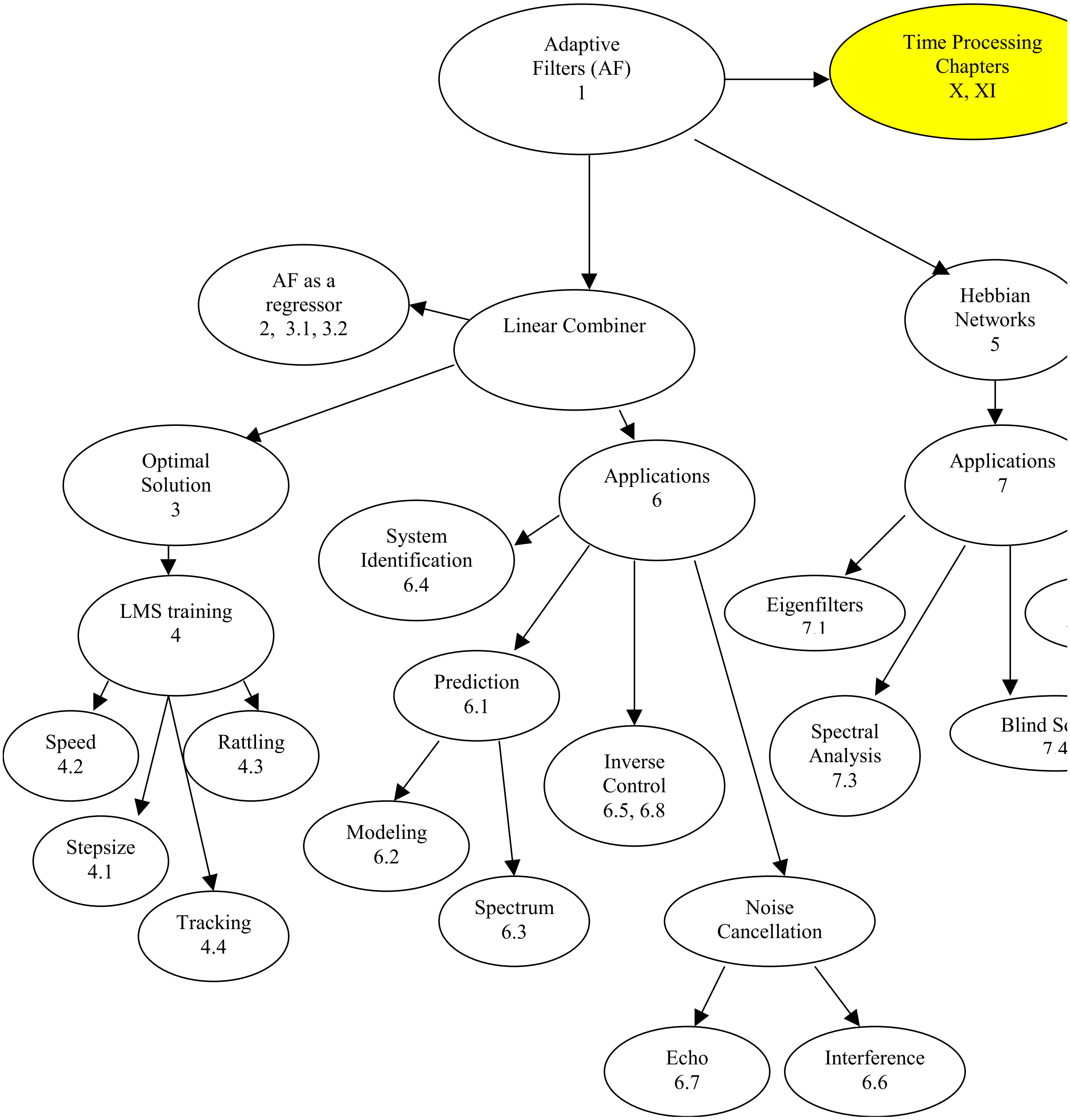
9.7 Prediction of time series

9.8 Prediction for modeling

9.9 Prediction for spectral analysis

Concept Map for Chapter IX

Chapter IX


Adaptive Filters (AF) 1
Time Processing Chapters X, XI
AF as a regressor 2, 3.1, 3.2
Linear Combiner
Hebbian Networks 5
Optimal Solution 3
Applications 6
Applications 7
System Identification 6.4
LMS training 4
Eigenfilters 7.1
Prediction 6.1
Speed 4.2
Rattling 4.3
Spectral Analysis 7.3
Blind Sc 7.4
Inverse Control 6.5, 6.8
Modeling 6.2
Stepsize 4.1
Spectrum 6.3
Tracking 4.4
Noise Cancellation
Echo 6.7
Interference 6.6

Go to Next Chapter

Go to Table of Contents

RLS algorithm

The recursive least square algorithm is an on-line implementation of the Newton's method that we discussed in Chapter I. The RLS algorithm utilizes a set of weights that is always the optimum given the data, unlike the LMS algorithm. If we use the analytic approach to solve the Wiener -Hopf equation, the RLS algorithm at the end of the data sequence, although iterative, provides exactly the same coefficient values as the analytic solution (for the same initial conditions). The RLS algorithm is also appealing since there is no stepsize. The problem of the RLS algorithm is that it is computationally more demanding than the LMS (complexity of $O(N^2)$). See Haykin, S., Adaptive Filter Theory, Prentice Hall, 1996. Here we will only provide a very brief explanation of the algorithm.

Let us assume that the input of the linear combiner is **x**(n) and the weights are **w**(n). The estimation of the autocorrelation function is **R**(n). The RLS algorithm specifies that the weights should be adapted according to

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{k}(n)[d(n) - \mathbf{x}^T(n)\mathbf{w}(n-1)]$$

where

$$\mathbf{k}(n) = \frac{\mathbf{P}(n-1)\mathbf{x}(n)}{1 + \mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}(n)}$$

and

$$\mathbf{P}(n) = \mathbf{P}(n-1) - \mathbf{k}(n)\mathbf{x}^T(n)\mathbf{P}(n-1)$$

where $\mathbf{P}(n) = \mathbf{R}^{-1}(n)$

We start by computing the gain vector **k**(n) by using the previous estimate of the inverse of **R** (**P**(n-1)) and the present input vector **x**(n). Once we have **k**(n) we can update the weights with the present input **x**(n) and also re-estimate the new **R** inverse (**P**(n)).

Notice that these formulas are recursive using always the previous available values for the estimates. The initialization is normally done with **w**(0)= 0 and **P**(0)= α**I** where $\alpha=1/100\sigma^2$ (power of the input).

The computational cost of the algorithm is proportional to $O(N^2)$, but notice that we are using second order information to update the weights (correlation function information). So the convergence is much faster and the final values more precise. The RLS is an on-line implementation of Newton’s method.

The RLS has problems in nonstationary environments since the gain **k**(n) tends to zero when the algorithm converges. If the properties of the data change the algorithm can not update the weights since **k**(n) is zero (or very small). This is the reason why we normally apply RLS with an exponential decaying window on the estimate of **R**(n).

Return to Text

system identification fundamentals

System identification is such an important problem that we have to spend some time analyzing it. Perhaps the best view point is to look at system identification from the point of view of function approximation. Being a form of function approximation, the three important problems in system identification are:

- What are the bases
- How to compute the parameters
- How to select the order of the approximation

As we have seen with the linear combiner we want to work with model systems and adapt their parameters. Hence the topology utilized in the model system will dictate the type of bases we will be using. The parameters of the model have to be adapted and we have to make decisions regarding the order of the model.

When a linear combiner is used the basis are the signal and its past samples as it is clearly seen from Eq. 1 . Since the model is a weighted sum of the input alone, this type of models are called moving average (MA). Once the topology is set, the only difference between the models are their parameters and their order. The order can be determined as we did in Chapter V by the Akaike or Rissanen (MDL) criteria. The parameters can be adapted with least squares, or if we prefer with LMS.

Experience has shown that if the plant belongs to the same family of models (MA) results can be very good with very small orders. This can be understood if we think in terms of eigendecompositions. If the plant is also a MA, then our model will be performing an eigendecompositon which is guaranteed to be very efficient. Unfortunately, not all linear systems are MA models. When the order is very large we can show that MA models can still approximate any other linear system at the expense of a large number of coefficients.

The most general linear system is given by

$$y(n) = \sum_{p=1}^{P} b_p y(n-p) + \sum_{q=0}^{Q} a_q x(n-q)$$

, which has an output given by a weighted sum of previous and present input and previous outputs. Such a system is called ARMA for autoregressive moving average. ARMA models are very powerful but very difficult to work with in practice due to three major reasons: they may get unstable during adaptation, the algorithms to update their parameters are more computationally demanding, and search can be caught in local minima (the performance surface is no longer convex as in the MA). So, they are less utilized then the MA models, but the appeal is that ARMA modeling is an eigendecompositon for linear systems. Note that the ARMA model is decomposing the output in a space created by the previous inputs and its own past states.

When we discussed linear systems, we presented systems which have a response dependent only upon the system output

$$y(n) = \sum_{i=1}^{N} y(n-i) + x(n)$$

which will be called autoregressive (AR) models. Note that the AR model is an extension of the first order recurrent system that we studied. AR models have very nice characteristics: first, they are very flexible being able to approximate well any other linear system. Second, they correspond to many physical processes, so they have a large chance of providing eigendecompositions. Third, we can often use a trick to adapt an AR model that avoids the problem of possible instability. We can use the prediction framework and adapt an MA model. Once this model is adapted we simply work with its inverse. It is easy to show that the inverse of a MA model is an AR model as we will do shortly. So the workhorse of linear system identification is the AR model.

Return to Text

Eq.25

$$y(n) = \sum_{i=0}^{N-1} w_i x(n-i)$$

Eq.30

$$P = W * R$$

Eq.36

$$E\left[e(n)x(n-i)\right] = 0 \qquad\qquad i = 0,\ldots,N-1$$

kalman

Rudolf Kalman is one of the most influencial electrical engineers alive who invented the Kalman filter. The Kalman filter is an optimal filter which solves the optimization in space state using the recursive minimum mean square estimation. See Haykin's book on

Adaptive filter theory for a signal processing view of Kalman filters.

Wiener

Norbert Wiener, Extrapolation, Interpolation and Smoothing of Stationary Time Series, MIT Press, 1949.

Kolmogorov

Andrei Kolmogorov (Russian mathematician), see the English translation in the book by Kailath, Linear least-square estimation, Dowden, Hutchinson and Ross, Stroudsburg, Pa, 1977.

Eq.41

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

ergodic

is a condition that some stationary random processes obey. The condition states that the stationary random process is ergodic if the statistical moments obtained by statistical operators equal the moments obtained with time operators.

A nonstationary random process can NOT be ergodic.

adaptive theory

is a branch of signal processing which analyzes and designs filters that learn from the data. Instead of having coefficients that are set by designer specifications, adaptive filters have a learning algorithm that modifies their coefficients such that the desired response is approximated.

Karhunen

Karhunen and Loeve were two European matematicians that discovered at the same time the eigendecomposition in Hilbert spaces. See Loeve, Probability Theory, Van Nostrand, 1963.

Eq.34

$$\mathbf{R} = A\left[\mathbf{x}_n \mathbf{x}_n^T\right] = \begin{bmatrix} r(0,0) & \dots & r(0,N-1) \\ \dots & \dots & \dots \\ r(N-1,0) & \dots & r(N-1,N-1) \end{bmatrix}$$

Eq.59

$$y(n) = \sum_{p=1}^{P} b_p y(n-p) + \sum_{q=0}^{Q} a_q x(n-q)$$

Widrow

Adaptive Signal Processing, Prentice Hall, 1985

Haykin

Adaptive Filter Theory, Prentice-Hall, 1986

Van Trees

Detection, Estimation, and Modulation Theory, Wiley, 1968

Index