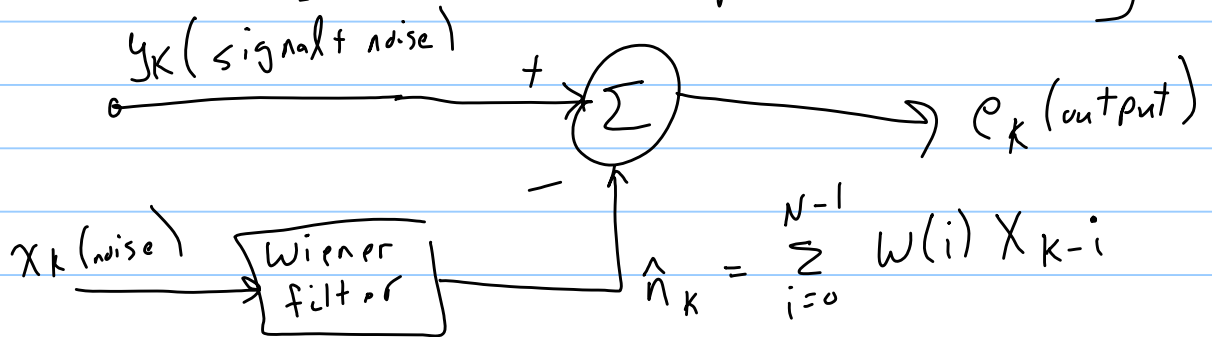


The Wiener Filter

This is the underlying structure of many adaptive filters

Two signals are applied simultaneously



$$e_k = y_k - \hat{n}_k = y_k - \sum_{i=0}^{N-1} w(i) x_{k-i} = y_k - \underbrace{\vec{w}^T \vec{x}_k}_{\text{This is just a number}}$$

$$\vec{x}_k = \begin{bmatrix} x_k \\ x_{k-1} \\ x_{k-2} \\ \vdots \\ x_{k-(N-1)} \end{bmatrix} \quad \vec{w} = \begin{bmatrix} w(0) \\ w(1) \\ w(2) \\ \vdots \\ w(N-1) \end{bmatrix}$$

Recall $\vec{w}^T \vec{x}_k$ is $[1 \times N][N \times 1] = [1 \times 1]$ (scalar)

So the square of the error can be written as

$$e_k^2 = y_k^2 - 2y_k \vec{w}^T \vec{x}_k + (\vec{w}^T \vec{x}_k)(\vec{w}^T \vec{x}_k)$$

$$e_k^2 = y_k^2 - 2y_k \vec{X}_k^T \vec{w} + \vec{w}^T \vec{X}_k \vec{X}_k^T \vec{w}$$

The mean squared error (MSE), J , is obtained by taking the expectation values of this equation

$$J = E[e_k^2] = E[y_k^2] - 2E[y_k \vec{X}_k^T \vec{w}] + E[\vec{w}^T \vec{X}_k \vec{X}_k^T \vec{w}]$$

$$J = \sigma^2 - 2\vec{p}^T \vec{w} + \vec{w}^T \vec{R} \vec{w}$$

$\sigma^2 = E[y_k^2] \equiv$ variance of y_k (since y_k contains noise and is random)

$\vec{p} = E[y_k \vec{X}_k] \equiv$ N -length cross-correlation vector of y_k with each X_k element

$\vec{R} = E[\vec{X}_k \vec{X}_k^T] \equiv N \times N$ autocorrelation matrix

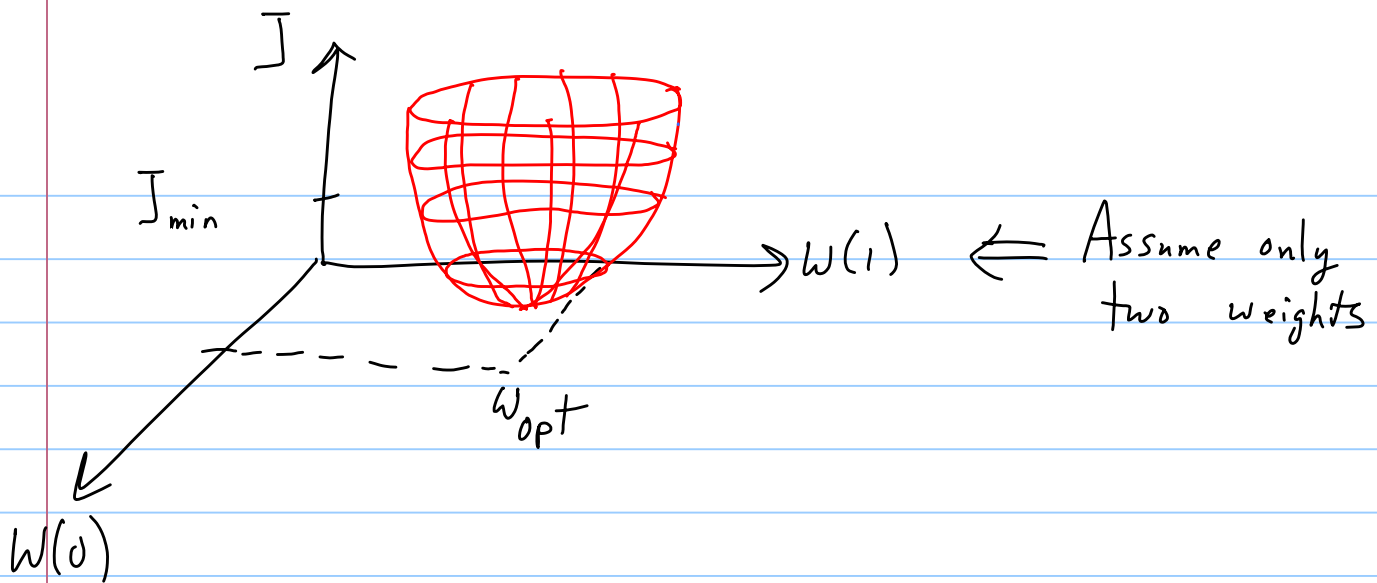
$$\hookrightarrow [N \times 1][1 \times N] \Rightarrow [N \times N]$$

J is still a scalar though $\sigma^2 = [1 \times 1]$

$$\vec{p}^T \vec{w} = [1 \times N][N \times 1] \Rightarrow [1 \times 1]$$

$$\vec{w}^T \vec{R} \vec{w} = [1 \times N][N \times N][N \times 1] \Rightarrow [1 \times N][N \times 1] \Rightarrow [1 \times 1]$$

Now since these are all squared terms J will be non-negative and parabolic in nature



We now want to find the minimum value of this surface which we know is at the bottom, which exists, and is where the gradient is equal to $\vec{0}$.

So let's find the slope of this surface (the gradient)

$$\vec{\nabla} J = \frac{dJ}{d\vec{w}} = \frac{d\sigma^2}{d\vec{w}} + \frac{d(-2\vec{P}^T \vec{w})}{d\vec{w}} + \frac{d(\vec{w}^T \vec{R} \vec{w})}{d\vec{w}}$$

$$= 0 + (-2\vec{P}) + (2\vec{R}\vec{w}) \quad \leftarrow \text{like } \frac{d(rx^2)}{dx} \Rightarrow 2rx$$

$$\vec{\nabla} J = -2\vec{P} + 2\vec{R}\vec{w}$$

We want the minimum of this gradient so set $\vec{\nabla} J = 0$

$$\vec{\nabla} J = 0 = -2\vec{p} + 2\vec{R}\vec{w}$$

$$\vec{p} = \vec{R}\vec{w}$$

Solve for the weights, \vec{w} , that make this true

$$\vec{w}_{\text{opt}} = \vec{R}^{-1}\vec{p} \quad \text{Wiener-Hopf Solution}$$

There are problems with this approach though

- * It requires autocorrelation matrix, \vec{R} , and cross-correlation vector \vec{p} which we don't usually know ahead of time
- * It requires matrix inversion (time consuming)
- * If the signals are nonstationary, then \vec{R} and \vec{p} will be changing with time and we have to compute \vec{w}_{opt} repeatedly

In principle, this technique would find the optimal weights in one set of calculations.

The LMS Algorithm

Instead of looking at the "bowl-surface" and going directly to the bottom (which we can only do if we can precompute \vec{R} and \vec{P}), we work our way towards the bottom (the minimum) by taking steps in the direction of the steepest descent.

Thus, in principle we can go step-by-step (sample-by-sample) towards the bottom according to:

$$\vec{W}_{k+1} = \vec{W}_k - \mu_k \vec{\nabla}_k J$$

$\vec{W}_k \equiv$ Weight vector at sample instant, k ,

$\vec{\nabla}_k \equiv$ Gradient vector at sample instant, k .

$\mu \equiv$ Learning/Update coefficient (lower this to help stability)

However, $\vec{\nabla}_k J$ would still require knowing \vec{R} and \vec{P}

$$\vec{\nabla}_k J = \frac{dJ}{d\vec{W}_k} = -2\vec{P}_k + 2\vec{R}_k \vec{W}_k$$

* So we still want to update the weights from \vec{W}_k to \vec{W}_{k+1} but we need a more practical way to estimate how to step down the bowl towards a minimum error.

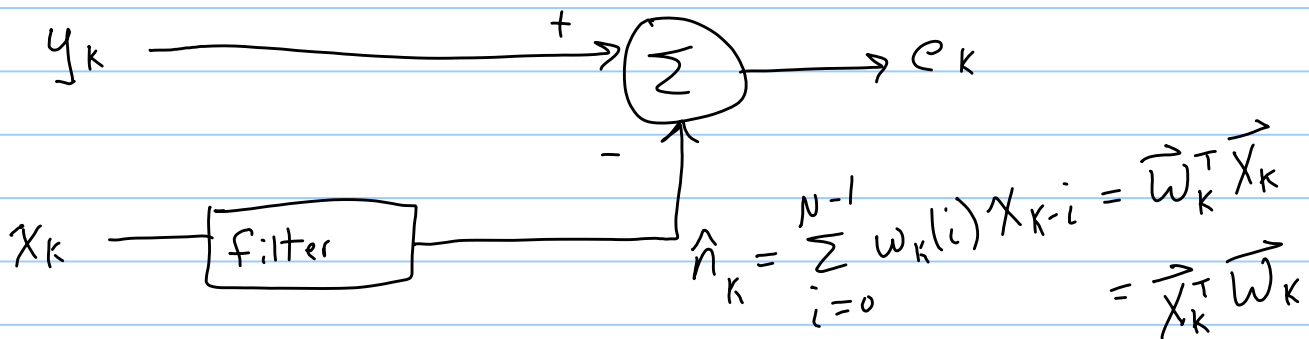
Not the expectation value
of cross and autocorrelation

The current
values of
correlation

$$\begin{aligned}\vec{\nabla}_k J &= -2\vec{p}_k + 2\vec{R}_k \vec{w}_k \approx -2y_k \vec{x}_k + 2\vec{x}_k \vec{x}_k^T \vec{w}_k \\ &= -2\vec{x}_k y_k + 2\vec{x}_k \vec{x}_k^T \vec{w}_k \quad (y_k \text{ is a scalar})\end{aligned}$$

$$\vec{\nabla}_k J = -2\vec{x}_k (y_k - \vec{x}_k^T \vec{w}_k)$$

but $y_k - \vec{x}_k^T \vec{w}_k = e_k$ our current output (estimate
of the signal)



$$\text{So } \vec{\nabla}_k J = -2e_k \vec{x}_k$$

Thus we may rewrite our steepest descent
equation as:

$$\vec{w}_{k+1} = \vec{w}_k - \mu \vec{\nabla}_k J$$

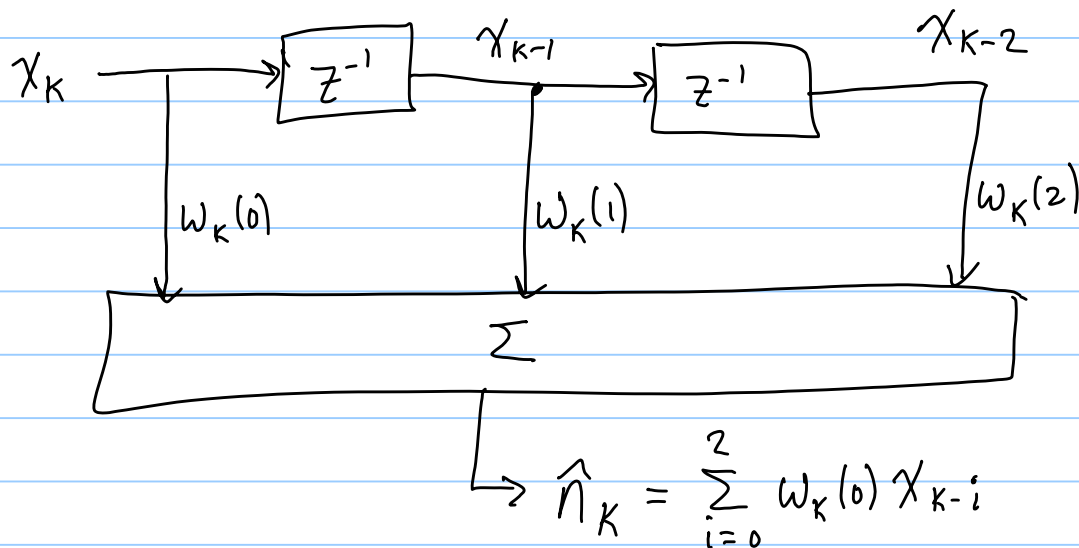
$$\vec{w}_{k+1} = \vec{w}_k + 2\mu e_k \vec{x}_k$$

This is a very useful weight update rule as it
only uses:

- (1) Current weights
- (2) Current output
- (3) Current x -vector (noise input signal)

For example, if we are using 3 values of noise (the current value plus the two previous values) then we have:
 $i = 0$ to 2

$$\begin{aligned} w_{k+1}(0) &= w_k(0) + 2\mu e_k x_k \\ w_{k+1}(1) &= w_k(1) + 2\mu e_k x_{k-1} \\ w_{k+1}(2) &= w_k(2) + 2\mu e_k x_{k-2} \end{aligned}$$



Weights are updated according to

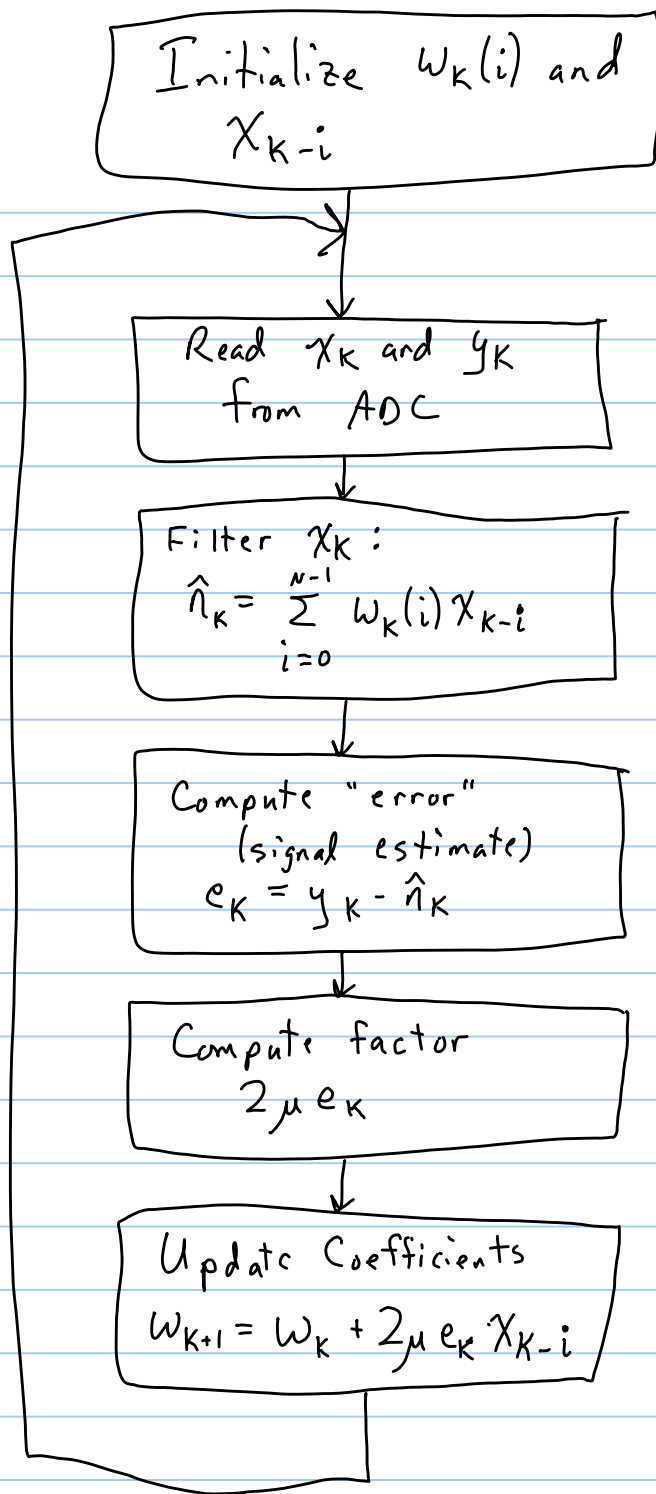
So for a particular application you would:

(1) Initialize the weights to arbitrary values (like 0)

(2) Compute filter output $\hat{n}_k = \sum_{i=0}^{N-1} w_k(i) x_{k-i}$

(3) Compute the error estimate $e_k = y_k - \hat{n}_k$

(4) Update the next filter weights $w_{k+1}(i) = w_k(i) + 2\mu e_k x_{k-i}$



So what should the learning coefficient be? There are two suggested measures:

$$0 < \mu < \frac{2}{\lambda_{\max}}$$

largest eigenvalue
of \vec{R}

$$0 < \mu < \frac{2}{\|\vec{x}(n)\|^2}$$

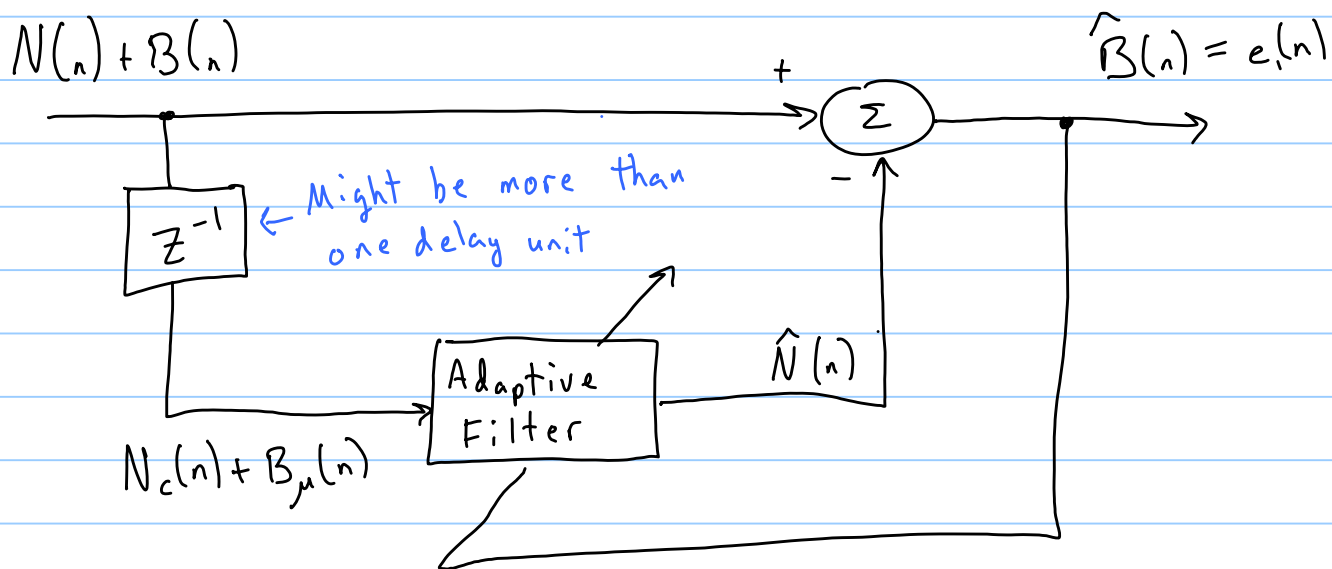
$$\|\vec{x}(n)\|^2 = x^2(n) + x^2(n-1) + x^2(n-2) + \dots + x^2(n-N+1)$$

Euclidean Norm

We have assumed to this point that we have two inputs (signal+noise and noise). This is not always the case.

* We may only have a signal with noise (one input)

Assume a narrowband signal and a wideband (broadband) noise component.



Narrowband signal $N(n)$ is not correlated with the broadband signal $B(n)$.

Delay the broadband signal enough such that it is uncorrelated with itself

$N_c(n)$ will stay correlated with $N(n)$

$B_u(n)$ will be uncorrelated with $B(n)$ ← Assuming this is your "noise" signal

If we minimize $E[e^2(n)]$ the output of the adaptive filter becomes an estimate of $N(n)$

That is, $N_c(n)$ becomes an estimate of $N(n)$ which is called $\hat{N}(n)$

(because the filter can't convert $B_u(n)$ to $B(n)$ since they are uncorrelated)

$$e(n) = \hat{B}(n) = N(n) + B(n) - \hat{N}(n)$$

since $\hat{N}(n)$ is an estimate of $N(n)$ then output $e(n)$ becomes an estimate of $B(n)$ called $\hat{B}(n)$

So if we want to isolate noise/broadband signal use $\hat{B}(n)$ (Narrowband Interference Canceller).

If we want to predict/extract narrowband signal use $\hat{N}(n)$ (Adaptive Line Enhancer).

Adaptive Line Enhancer

Assume the useful signal is $N(n) = \sin\left(\frac{2\pi f_0}{f_s} n\right)$

f_s = sampling frequency

f_0 = frequency of sine wave

Assume the noise is white Gaussian noise with a variance such that SNR is "large"

Assume we delay the input one sample to decorrelate $B_u(n)$ with $B(n)$

Try 5-tap FIR filter

Try 10-tap FIR filter

More taps generally yields better performance.

However, we could get similar performance by just using a bandpass filter to keep our narrowband signal and filter out the noise.

So why bother with adaptive?

Notice what happens when you change the frequency of the input signal...

You still pass the signal through!

The filter automatically adjusts its passband!