Adaptive Filter Theory

In many cases we need a filter to have properties that change with time. This can be because if:

1. Changing conditions.
2. When there is spectral overlap between the signal and noise.
3. If the band occupied by the noise is unknown or varies with time.

For example:

1. EEG, eye movements and other artifacts are also picked up by the contacts. The "noise" is of greater amplitude than the EEG signal so standard filters aren't always effective.
2. ECG, can pick up 60 Hz noise off of the power-supply. In principle we can create a notch filter at 60 Hz, but this will also filter out part of the ECG.
3. Digital communications being affected by jamming noise. The jamming often is a narrowband but high intensity (and the band the noise occupies isn't known).

Let us start by looking at using an adaptive filter for noise cancellation.

There are two basic parts to the adaptive filter:
(1) Digital Filter with adjustable coefficients,
(2) Some adaptive algorithm to adjust or modify the coefficients,

\[ y_k = s_k + n_k \quad \text{(signal plus noise)} \]

\[ x_k \quad \text{(noise)} \]

Digital Filter

\[ \hat{n}_k \quad \text{(noise estimate)} \]

Adaptive Algorithm

\[ e_k = \hat{s}_k \quad \text{(signal estimate)} \]

\[ y_k = s_k + n_k \quad \text{This contains both the desired signal and noise. We assume the signal and noise are uncorrelated with each other.} \]

\[ x_k \quad \text{is a measure of the noise. It is probably not exactly the same as the real noise signal, } n_k \text{, but } x_k \text{ is somehow correlated with } n_k. \]

For example, in picking up a fetal heartbeat we may also pick up part of the mother’s heartbeat (the “noise”). We can use another set of sensors to measure the mother’s heartbeat (ECG) which is \( x_k \).

The actual noise pickup in the Fetal ECG, \( n_k \), will obviously be related to \( x_k \) but will not be exactly the same. (Thus \( n_k \neq x_k \), but they are correlated.)
*We can use the digital filter (or transfer function) to modify \( x_k \) into something that is as close to \( n_k \) as possible. This is \( \hat{n}_k \) (an estimate of \( n_k \)).
* We then subtract out \( \hat{n}_k \) from \( y_k \) to try to obtain just the desired signal \( s_k \).

We won’t be exact most of the time so we get \( \hat{s}_k \), an estimate of the actual signal,

\[
\hat{s}_k = y_k - \hat{n}_k = s_k + n_k - \hat{n}_k
\]

If we can achieve \( \hat{n}_k = n_k \) then \( \hat{s}_k = s_k \).

The digital filter (or transfer function) is usually implemented as an FIR filter structure,

\[
\begin{align*}
\chi_k & \xrightarrow{\text{FIR}} \hat{n}_k \\
\hat{n}_k & = \sum_{i=0}^{N-1} w_{k}(i) \chi_{k-i}
\end{align*}
\]

\( w_k(i) \) = The \( N \) filter coefficients at sample time \( k \)

\( \chi_{k-i} \) = The inputs to the filter (could be current & past inputs for single input system) (could be several current inputs for multiple input system)
What happens if we try to minimize the output signal power?

\[ \hat{S}_k = y_k - \hat{\eta}_k = S_k + n_k - \hat{\eta}_k \leftarrow \text{This is value of output signal at time } k \]

To get to signal power, square the equation.

\[ \hat{S}_k^2 = S_k^2 + 2S_k(n_k - \hat{\eta}_k) + (n_k - \hat{\eta}_k)^2 \leftarrow \text{We could try to look at average values but since these involve random processes we need to look at “expectation values”} \]

\[ E[\hat{S}_k^2] = E[S_k^2] + E[(n_k - \hat{\eta}_k)^2] + E[2S_k(n_k - \hat{\eta}_k)] \]

However, since \( S_k \) is uncorrelated with the noise \( n_k \) or \( \hat{\eta}_k \) this expectation value will tend to zero. Thus

\[ E[\hat{S}_k^2] = E[S_k^2] + E[(n_k - \hat{\eta}_k)^2] \]

\[ \uparrow \quad \uparrow \quad \uparrow \]

Total Output \quad Total Signal Power \quad Noise Power

We can adjust \( \hat{\eta}_k \) and try to minimize this

\[ \min E[\hat{S}_k^2] = E[S_k^2] + \min E[(n_k - \hat{\eta}_k)^2] \]

\[ \uparrow \]

This isn’t affected by adjusting \( \hat{\eta}_k \) since it is uncorrelated \( S_k \).
So if we minimize the output power $E\left[ \hat{S}_K^2 \right]$ we will in turn be minimizing $E\left[ (\hat{\mathbf{n}}_K - \hat{\mathbf{n}}_K)^2 \right]$ and thus maximizing the signal-to-noise ratio.

**The Wiener Filter**

This is the underlying structure of many adaptive filters. Two signals are applied simultaneously.

\[
\begin{align*}
\mathbf{y}_k \text{ (signal + noise)} & \xrightarrow{\Sigma} \mathbf{e}_k \text{ (output)} \\
\mathbf{x}_k \text{ (noise)} & \xrightarrow{\text{Wiener Filter}} \hat{\mathbf{n}}_k = \sum_{i=0}^{N-1} w(i) \mathbf{x}_{k-i} \\
\end{align*}
\]

A fixed set of weights, we just need to figure out what to use.

\[
\mathbf{e}_k = \mathbf{y}_k - \hat{\mathbf{n}}_k = \mathbf{y}_k - \sum_{i=0}^{N-1} w(i) \mathbf{x}_{k-i} = \mathbf{y}_k - \overrightarrow{\mathbf{w}}^T \overrightarrow{\mathbf{x}}
\]

This is just a scalar number.

\[
\overrightarrow{\mathbf{x}}_k = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{x}_{k-1} \\ \mathbf{x}_{k-2} \\ \vdots \\ \mathbf{x}_{k-(N-1)} \end{bmatrix} \quad \overrightarrow{\mathbf{w}} = \begin{bmatrix} w(0) \\ w(1) \\ w(2) \\ \vdots \\ w(N-1) \end{bmatrix}
\]

Recall $\overrightarrow{\mathbf{w}}^T \overrightarrow{\mathbf{x}}_k$ is $[1 \times N][N \times 1] = [1 \times 1]$ scalar.

So, the square of the error can be written as

\[
E^2_k = \mathbf{y}_k^2 - 2\mathbf{y}_k \overrightarrow{\mathbf{w}}^T \overrightarrow{\mathbf{x}}_k + (\overrightarrow{\mathbf{w}}^T \overrightarrow{\mathbf{x}}_k)^2
\]
\[ e_k^2 = y_k^2 - 2y_k \overline{X}_k^r W + \overline{W}^T \overline{X}_k \overline{X}_k^r \overline{W} \]

The Mean Squared Error (MSE), \( J \), is obtained by taking the expectation values of this equation,

\[ J = E[e_k^2] = E[y_k^2] - 2E[y_k \overline{X}_k^r \overline{W}] + E[\overline{W}^T \overline{X}_k \overline{X}_k^r \overline{W}] \]

\[ J = \sigma^2 - 2 \overline{P}^T \overline{W} + \overline{W}^T \overline{R} \overline{W} \]

\[ \sigma^2 = E[y_k^2] = \text{variance of } y_k \text{ (since } y_k \text{ contains noise and is random)} \]

\[ \overline{P} = E[y_k \overline{X}_k] = N \text{-length cross-correlation vector of } y_k \text{ with each } x_k \text{ element} \]

\[ \overline{R} = E[\overline{X}_k \overline{X}_k^T] = N \times N \text{ autocorrelation matrix} \]

\[ \Rightarrow [N \times 1][1 \times N] = [N \times N] \]

\( J \) is still a scalar though:

\[ \sigma^2 = [1 \times N][N \times 1] \Rightarrow [1 \times 1] \]

\[ \overline{P}^T \overline{W} = [1 \times N][N \times 1] \Rightarrow [1 \times 1] \]

\[ \overline{W}^T \overline{R} \overline{W} = [1 \times N][N \times N][N \times 1] \Rightarrow [1 \times 1] \]

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

Assuming only two weights we have something like this:
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).

\[
\vec{\nabla} I = \frac{dI}{d\vec{w}} = \frac{d\theta^2}{d\vec{w}} + \frac{d(-2\vec{P}\vec{w})}{d\vec{w}} + \frac{d(\vec{w}^T\vec{R}\vec{w})}{d\vec{w}}
\]

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

\[
\vec{\nabla} I = -2\vec{P} + 2\vec{R}\vec{w}
\]

We want the minimum value so set \( \vec{\nabla} I = 0 \)

\[
\vec{\nabla} I = 0 = -2\vec{P} + 2\vec{R}\vec{w}
\]

\( \vec{P} = \vec{R}\vec{w} \) Now solve for the \( \vec{w} \) that makes this true.

\[
\star \quad \vec{w}_{opt} = \vec{R}^{-1}\vec{P} \quad \text{Wiener-Hopf Solution} \quad \star
\]

There are problems with this approach though.

\( \star \) It requires the autocorrelation matrix, \( \vec{R} \), and cross-correlation vector, \( \vec{P} \), which we don’t usually know a priori.

\( \star \) It requires matrix inversion.

\( \star \) 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 $\hat{R}$ and $\hat{P}$), we work our way towards the bottom (the minimum) by taking steps in the direction of steepest descent.

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

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

where $\vec{W}_k$ = Weight Vector at sample instant, $k$.
$\vec{\nabla}_k$ = Gradient Vector at sample instant, $k$.
$\mu$ = Learning/Update coefficient (lower this to help stability)

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

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.

Let us write $\vec{\nabla}_k J$ as:

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

* A very important but subtle change has been made from the Wiener-Hopf solution! *
In going from $\nabla J$ to $\nabla_k J$ we have switched from using the expectation values of the terms to using the current values of the terms at time step $k$.

$$\nabla J = -2\bar{\rho} + 2\bar{R}\bar{w}$$

$$\bar{\rho} = \mathbb{E}[y_K X_K]$$

$$\bar{R} = \mathbb{E}[\hat{x}_K \hat{x}_K^T]$$

Now, $\nabla_k J = \frac{dJ}{d\omega_k} = -2\bar{\rho}_k + 2\bar{R}_k \bar{w}_k = -2y_K \hat{x}_K + 2\hat{x}_K \hat{x}_K^T \hat{w}_k$

$$\nabla_k J = -2\hat{x}_K y_K + 2\hat{x}_K \hat{x}_K^T \hat{w}_k$$

$$= -2\hat{x}_K \left( y_K - \hat{x}_K^T \hat{w}_k \right) \quad (y_k \text{ is a scalar})$$

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

$$y_K \quad \sum \quad e_K = y_K - \hat{x}_K^T \hat{w}_k$$

$$\hat{x}_K \quad \text{filter} \quad \hat{X}_K = \sum_{i=0}^{N-1} \omega_K(i) \hat{x}_K(i) = \hat{w}_k^T \hat{x}_K = \hat{x}_K^T \hat{w}_k$$

So $\nabla_k J = -2e_k \hat{x}_K$. Thus we may rewrite our steepest descent equation as:

$$\omega_{k+1} = \omega_k - \mu \nabla_k J$$

$$\omega_{k+1} = \omega_k + 2\mu e_k \hat{x}_K$$
This is a very useful weight update rule as it only uses the (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 previous two values), then we have:

\[ \begin{align*}
    i = 0 & \text{ to } 2 \\
    W_{k+1}(0) &= W_k(0) + 2\mu e_k x_k \\
    W_{k+1}(1) &= W_k(1) + 2\mu e_k x_{k-1} \\
    W_{k+1}(2) &= W_k(2) + 2\mu e_k x_{k-2}
\end{align*} \]

\[ \hat{y}_k = \sum_{i=0}^{2} W_k(i) x_{k-i} \]

with the weights updated according to:

So for a particular application you would:
(1) Initialize the weights to arbitrary values (like 1)
(2) Compute filter output \( \hat{y}_k = \sum_{i=0}^{2} W_k(i) x_{k-i} \)
(3) Compute the error estimate \( e_k = y_k - \hat{y}_k \)
(4) Update the next filter weights \( W_{k+1}(i) = W_k(i) + 2\mu e_k x_{k-i} \)
Initialize $W_k(i)$ and $X_{k-1}$

Read $X_k$ and $Y_k$ from ADC

Filter $X_k$

$\hat{X}_k = \sum_{i \in B} W_k(i)X_k$

Compute “error” (signal estimate)

$e_k = Y_k - \hat{X}_k$

Compute factor

$Z_{μ}e_k$

Update Coefficients

$W_{k+1} = W_k + Z_{μ}e_kX_{k-1}$

Guide for picking the learning coefficient

$0 < μ < \frac{2}{\lambda_{\text{max}}}$

$0 < μ < \frac{2}{||X(n)||^2}$

largest

eigenvalue
of $\bar{R}$

$||X(n)||^2 = x^2(n) + x^2(n-1) + x^2(n-2) + ... + x^2(n-N+1)$

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.

\[ N(k) + B(k) \]

![Diagram of signal processing system]

\[ \hat{B}(k) = e(k) \]

Narrowband signal \( N(k) \) is not correlated with broadband signal \( B(k) \).

We want to delay the broadband signal enough that it is uncorrelated with itself.

\[ N_c(k) \text{ will stay correlated with } N(k) \]
\[ B_n(k) \text{ will be uncorrelated with } B(k) \]

Assuming this is your “noise” signal,

If we minimize \( E[e^2(k)] \) the output of the adaptive filter becomes an estimate of \( N(k) \).
That is to say, \( N_e(k) \) becomes an estimate of \( N(k) \) which is called \( \hat{N}(k) \) (because the filter can't convert \( B_u(k) \) to \( B(k) \)), since they are uncorrelated.

\[
e(k) = \hat{B}(k) = N(k) + B(k) - \hat{N}(k)
\]

Since \( \hat{N}(k) \) is an estimate of \( N(k) \), then the output \( e(k) \) becomes an estimate of \( B(k) \) called \( \hat{B}(k) \).

So if we want to isolate the noise/broadband signal, use \( \hat{B}(k) \) [Narrowband Interference Cancelor].

If we want to predict/extract the narrowband signal, use \( \hat{N}(k) \) [Adaptive Line Enhancer].

Adaptive Line Enhancer (ALE)

Assume the useful signal is \( N(k) = \sin\left(\frac{2\pi f_o k}{f_s}\right) \)

\( f_s \) = Sampling Frequency
\( f_o \) = Frequency of sine wave

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

Assume we delay the input one sample to decorrelate \( B_u(k) \) with \( B(k) \).

Observe the MATLAB results from m-files on the website.
More taps generally yields more 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 desired signal through!

It's as though the filter automatically adjusts its passband!