EE434
Biomedical Signal Processing
Lecture # 6

I would like to thank

Professor Robi Polikar for using his lecture notes.

Optimal & Adaptive Filtering

Optimal Filtering

Optimal Filtering

- Optimal Filters
  - The case against standard FIR / IIR filters
- Wiener Filter
  - Development of Wiener Filter
  - Wiener-Hopf Equations
- Applications / Demos of Wiener Filter
  - Optimal Filtering
  - Noise Removal

The Case Against FIR/IIR Filters

- The many forms of FIR and IIR filter design described in DSP allows us to design an arbitrarily specific frequency response with arbitrary precision.
  - Is there anything that we cannot do with IIR/FIR filters?
    - No, not really. You can design any filter with any accuracy
  - Then what is the problem?
    - The user needs to know the precise filter characteristics that need to be designed.
  - Is that not usually the case…?
    - Not really! Often, we specify the filter characteristics based on some past experience, and/or trial and error.
    - FIR/IIR design techniques can design any filter you specify – it does not tell you whether that is the right filter for that specific signal.
  - Is there a technique that can…?
    - Yes! There are design techniques that ensures that filter specs they specify are optimal with respect to some meaningful criterion
    - Wiener filters

Problem Statement

The optimal filter design problem can be stated as follows:

- Design an optimal filter to remove noise from a signal, given that the signal and noise processes are independent, (wide sense) stationary random processes. We will assume that we have access to the “desired” or signal, or the ideal characteristics of the uncorrupted signal to be known. We further assume that the statistical properties of the noise process are also known.

- The optimality is understood to be in “minimum mean square error” sense.

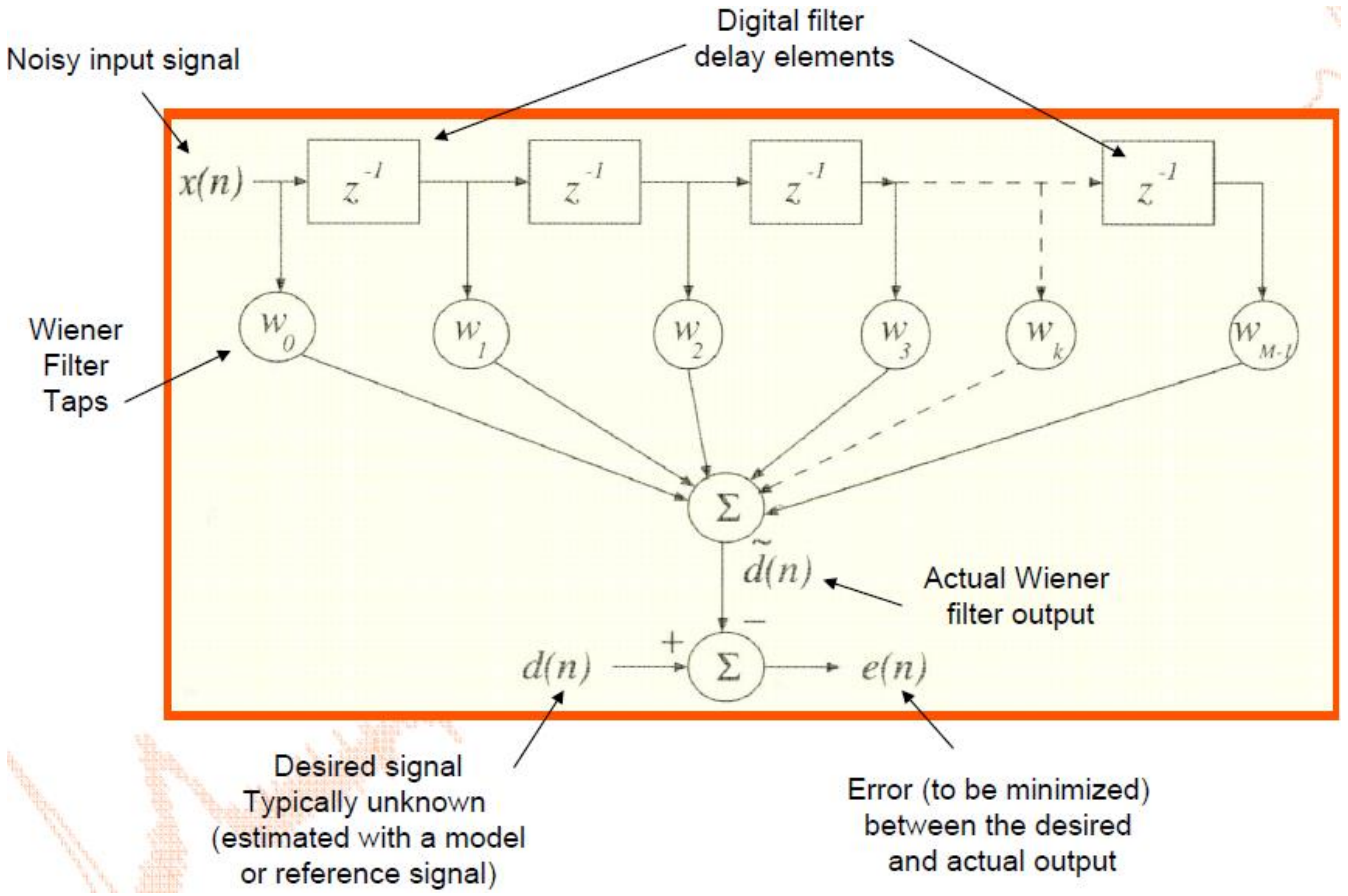
Shaky Assumptions?

- What does it mean that we have access to the desired / clean signal? If we have access to the desired signal, why do we need to filter in the first place?
    - Of course, we typically do not have the “clean” version of the noisy signal – if we did, we would not need to clean it.
    - However, we often have access to a reference signal, or at least a model of such a signal, that can reasonably estimate the true, uncorrupted signal.
    - As we will also see later, we technically need the autocorrelation sequence of the input signal, along with its cross correlation with the desired signal – both of which are normally estimated from the model / reference signal.
    - This is discussed in more detail later in this lecture.
- How about noise?
    - Assuming that the noise is white with constant power is not an unreasonable one most applications.

Wiener Filter

- **Wiener filter** requires no spectral specification of which frequencies should be blocked and which ones should be passed.
  - Instead, it takes the statistical properties of the signal and noise processes into account, and determines an optimal set of filter coefficients that are optimized with respect to a performance criterion – a figure of merit.
    - The criterion function is typically the difference between the filter output and the desired signal (reference /model).
    - **Wiener filter** minimizes the average squared difference between the two.
  - The output of this filter is then guaranteed to be the best under the stated assumptions.

Wiener Filter


Digital filter delay elements
Noisy input signal
$x(n)$
z^{-1}
w_0
w_1
w_2
w_3
w_k
w_{M-1}
Wiener Filter Taps
Σ
$\tilde{d}(n)$
Actual Wiener filter output
$-$
$+$
$d(n)$
$e(n)$
Desired signal Typically unknown (estimated with a model or reference signal)
Error (to be minimized) between the desired and actual output

Some Filter Math…

- The **Wiener filter** is typically designed to be an FIR filter – hence its output is the convolution of $\mathbf{x}$ and $\mathbf{w}$ vectors:

$$\tilde{d}(n) = \sum_{k=0}^{M-1} w_k x(n-k) = \mathbf{w}^T \mathbf{x}(n) = \mathbf{x}^T(n)\mathbf{w} = \langle \mathbf{x}, \mathbf{w} \rangle$$

$$\mathbf{w} = \left[w_0, w_1, \cdots, w_{M-1}\right]^T, \qquad \mathbf{x}(n) = \left[x(n), x(n-1), \cdots, x(n-M+1)\right]^T$$

- Note that the vector $\mathbf{x}(n)$ is time dependent – it includes the currently available sample at time instant n and its preceding M-1 values.
  - The estimation error is then
$$e(n) = d(n) - \tilde{d}(n) = d(n) - \mathbf{w}^T \mathbf{x}(n)$$
$$= d(n) - \sum_{k=0}^{M-1} w_k x(n-k)$$
  - And the error we wish to minimize is

$$J(\mathbf{w}) = E\left[e^2(n)\right] \propto \sum_{n=0}^{M-1} \left[d(n) - \sum_{k=0}^{M-1} w_k x(n-k)\right]^2$$

Some Random Process Math…

- Considering all signals as random variables, we have the following scenario

$$\boxed{\begin{aligned} J(\mathbf{w}) &= E\left[e^2(n)\right] = E\left[\left\{d(n)-\mathbf{w}^T\mathbf{x}(n)\right\}\left\{d(n)-\mathbf{x}^T(n)\mathbf{w}\right\}\right] \\ &= E\left[d^2(n)\right]-\mathbf{w}^T E\left[\mathbf{x}(n)d(n)\right]-E\left[d(n)\mathbf{x}^T(n)\right]\mathbf{w}+\mathbf{w}^T E\left[\mathbf{x}(n)\mathbf{x}^T(n)\right]\mathbf{w} \end{aligned}}$$

  - Note that $\mathbf{w}$ is fixed (filter weights) and hence is not a random variable.

- Under the (somewhat incorrect) assumption that $\mathbf{x}(n)$ and $d(n)$ are jointly stationary, we have the following interpretations

  - Since $d(n)$ can be normalized to have zero mean, $E[d^2(n)]$ is the variance of $d(n)$, denoted as σ_d^2
  - $\Theta_{xd} = \Theta = E[\mathbf{x}(n)\ d(n)]$ is the cross correlation of the input and the desired signals.

    Similarly, $\Theta^{\mathrm{T}} = E[d(n)\ \mathbf{x}^{\mathrm{T}}(n)] = (E[\mathbf{x}(n)\ d(n)])^{\mathrm{T}}$
  - Also note that Θ is a M–by-1 vector of values $\Theta = [\ \theta(0), \theta(-1), \ldots, \theta(1-M)]^{\mathrm{T}}$, where $\theta(k) = \theta(-k) = E[\mathbf{x}(n-k)\ d(n)]$, $k = 0, 1, \ldots, M-1$. The stationarity assumption makes the covariance function independent of *n*.
  - $\Phi_{xx} = \Phi = E[\mathbf{x}(n)\ \mathbf{x}^{\mathrm{T}}(n)]$ is the autocorrelation function of the signal $\mathbf{x}$.

Some Random Process Math…

- The autocorrelation matrix, $\Phi_{xx} = \Phi = E[\mathbf{x}(n)\,\mathbf{x}^T(n)]$ can be shown in the full M-by-M matrix form as

$$\Phi = \begin{bmatrix} \phi(0) & \phi(1) & \cdots & \phi(M-1) \\ \phi(-1) & \phi(0) & \cdots & \phi(M-2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(-M+1) & \phi(-M+2) & \cdots & \phi(0) \end{bmatrix} = \begin{bmatrix} \phi(0) & \phi(1) & \cdots & \phi(M-1) \\ \phi(1) & \phi(0) & \cdots & \phi(M-2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(M-1) & \phi(M-2) & \cdots & \phi(0) \end{bmatrix}$$

  - where row k, column i element is $\phi(k - i) = \phi(i - k) = E[x(n-k)\, x(n - i)]$
    - This is a toeplitz matrix, and for a WSS process, it is completely defined by its single row of values, $\phi = \phi_0, \ldots, \phi_{M-1}$.
    - In Matlab, the autocorrelation matrix can be obtained by *Phi_matrix* = **toeplitz**(*phi*);

- Putting it all together, then, the criterion function to be minimized is

$$J(\mathbf{w}) = \sigma_d^2 - \mathbf{w}^T\Theta - \Theta^T\mathbf{w} + \mathbf{w}^T\Phi\mathbf{w}$$

Some Random Process Math…

$$J(\mathbf{w}) = \sigma_d^2 - \mathbf{w}^T\Theta - \Theta^T\mathbf{w} + \mathbf{w}^T\Phi\mathbf{w}$$

- The optimal filter weights, **w*** are the ones that minimize this (error) function → we set the derivative of $J(\mathbf{w})$ wrt to **w** to zero and solve for **w** :
  - The following derivatives apply:

$$\frac{d}{d\mathbf{w}}\left(\Theta^T\mathbf{w}\right) = \frac{d}{d\mathbf{w}}\left(\mathbf{w}^T\Theta\right) = \Theta$$

$$\frac{d}{d\mathbf{w}}\left(\mathbf{w}^T\Phi\mathbf{w}\right) = 2\Phi\mathbf{w}$$

(Why not $\frac{d}{d\mathbf{w}}\left(\mathbf{w}^T\Phi\mathbf{w}\right) = 2\mathbf{w}\Phi$?)

  - Then $\boxed{\frac{d}{d\mathbf{w}}J(\mathbf{w}) = -2\Theta + 2\Phi\mathbf{w} \Rightarrow \Phi\mathbf{w}^* = \Theta \Rightarrow \boxed{\mathbf{w}^* = \Phi^{-1}\Theta}}$

  - This is the **Wiener-Hopf equation**, also called **normal equation**, the solution of which gives the optimal filter coefficients **w***

Wiener-Hopf Equations

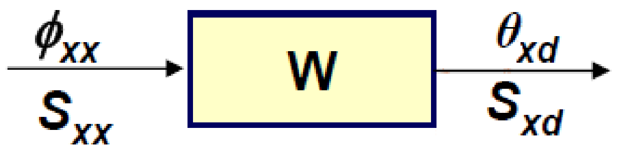
- In expanded form, the Wiener-Hopf equations can be written as

$$\begin{bmatrix} \phi(0) & \phi(1) & \cdots & \phi(M-1) \\ \phi(1) & \phi(0) & \cdots & \phi(M-2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(M-1) & \phi(M-2) & \cdots & \phi(0) \end{bmatrix} \begin{bmatrix} w_0^* \\ w_1^* \\ \vdots \\ w_{M-1}^* \end{bmatrix} = \begin{bmatrix} \theta(0) \\ \theta(1) \\ \vdots \\ \theta(M-1) \end{bmatrix}$$

  - or as

$$\sum_{i=0}^{M-1} w_i^* \phi(k-i) = \theta(k), \quad k = 0,1,\cdots,M-1$$

- Note that this equation defines the convolution of filter coefficients with the autocorrelation function of the input signal, which means we have



$$w_k^* * \phi(k) = \theta(k)$$
$$W(\omega) \cdot S_{xx}(\omega) = S_{xd}(\omega)$$

Cross-spectral density between the input signal and the desired signal

- The **frequency response of the optimal Wiener filter** is then

$$\boxed{W(\omega) = \frac{S_{xd}(\omega)}{S_{xx}(\omega)}}$$

Wiener-Hopf Equations

- Having obtained the optimal filter coefficients as $\mathbf{w}^* = \Phi^{-1}\Theta$, we can now calculate the minimum mean square error (MMSE) as

$$\boxed{\begin{gathered} J(\mathbf{w})\Big|_{\mathbf{w}=\Phi^{-1}\Theta} = \sigma_d^2 - \mathbf{w}^T\Theta - \Theta^T\mathbf{w} + \mathbf{w}^T\Phi\mathbf{w}\Big|_{\mathbf{w}=\Phi^{-1}\Theta} \\ \Rightarrow J_{\min} = \sigma_d^2 - \Theta^T\Phi\Theta \end{gathered}}$$

- We note that the calculation of optimal filter weights require knowledge of the signal's autocorrelation function (easy to obtain), and the crosscorrelation function of the input with the desired output (not so easy to obtain).
  - Typically, the cross correlation function is estimated from some prior knowledge about the second order statistics of the desired signal, and / or from a reference or model signal.

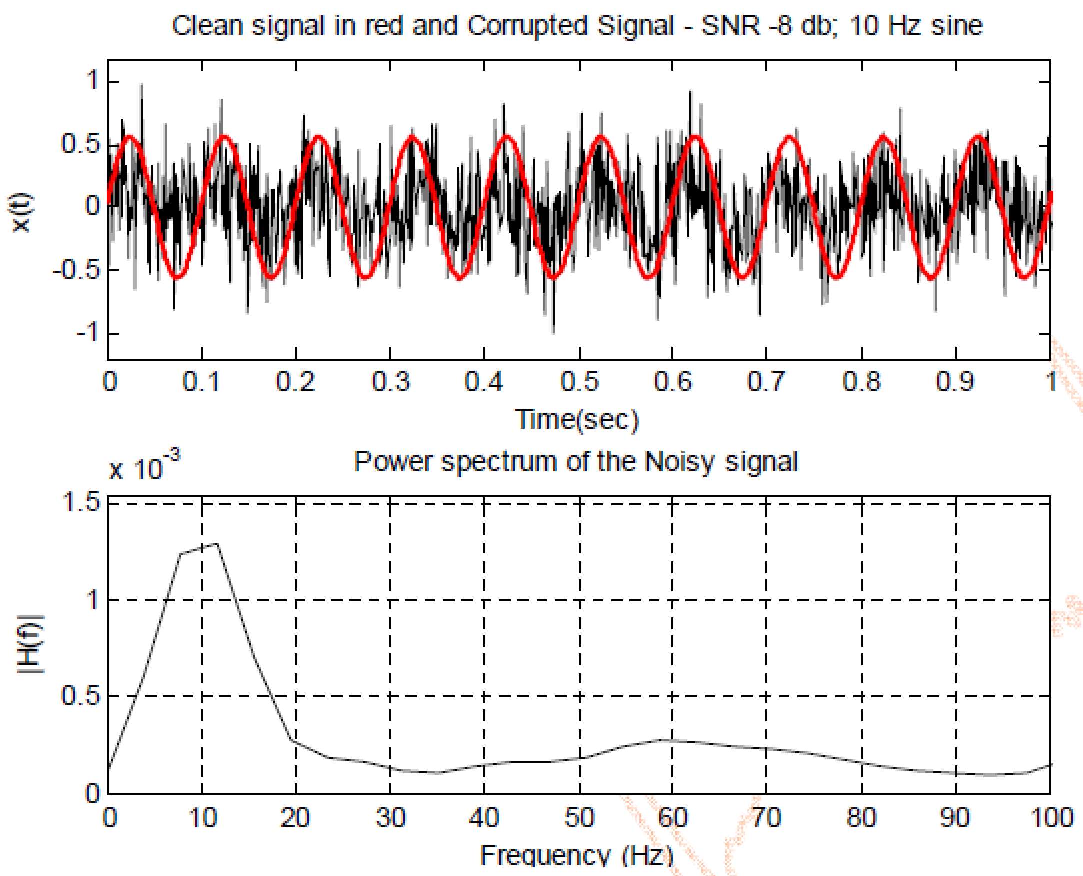
Matlab Implementation

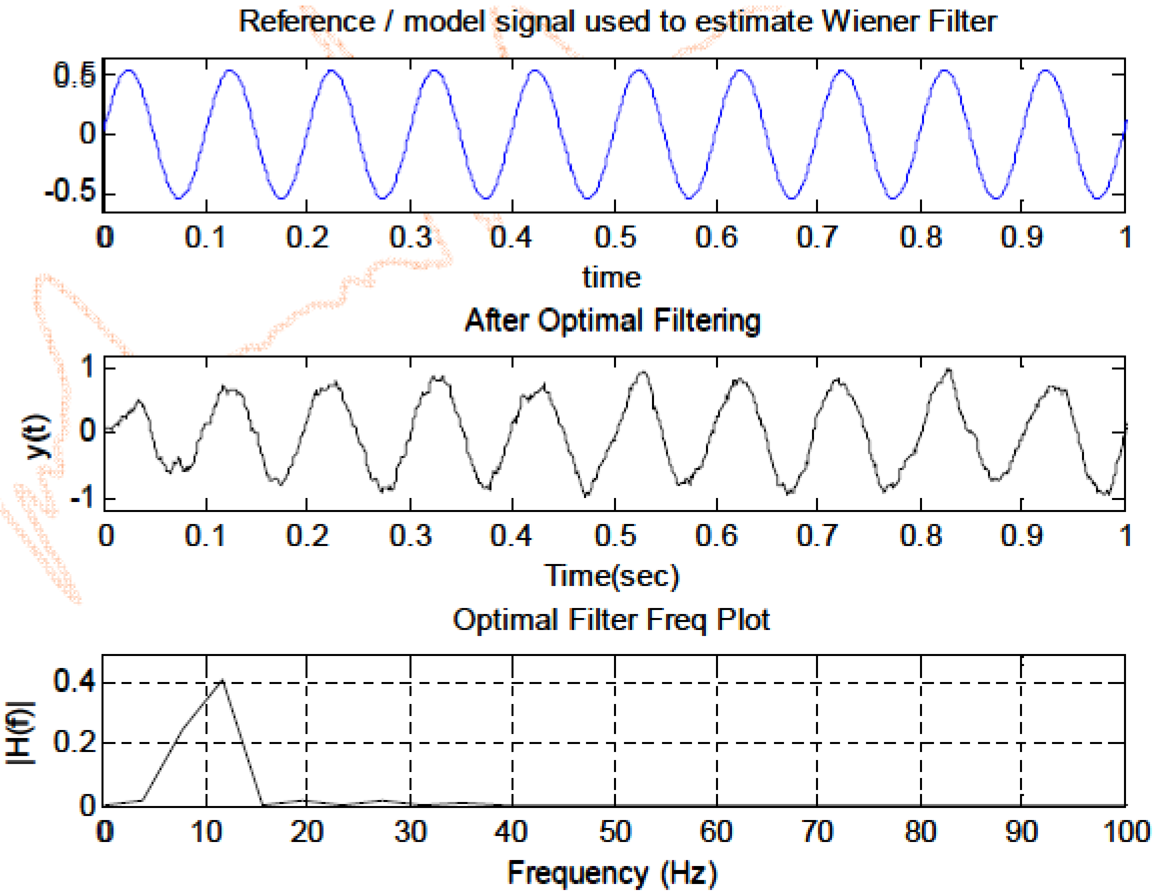
- It seems rather simple to obtain the **optimum filter coefficients** once the autocorrelation and cross-correlation functions are known:

$$\mathbf{w}^* = \Phi^{-1}\Theta$$

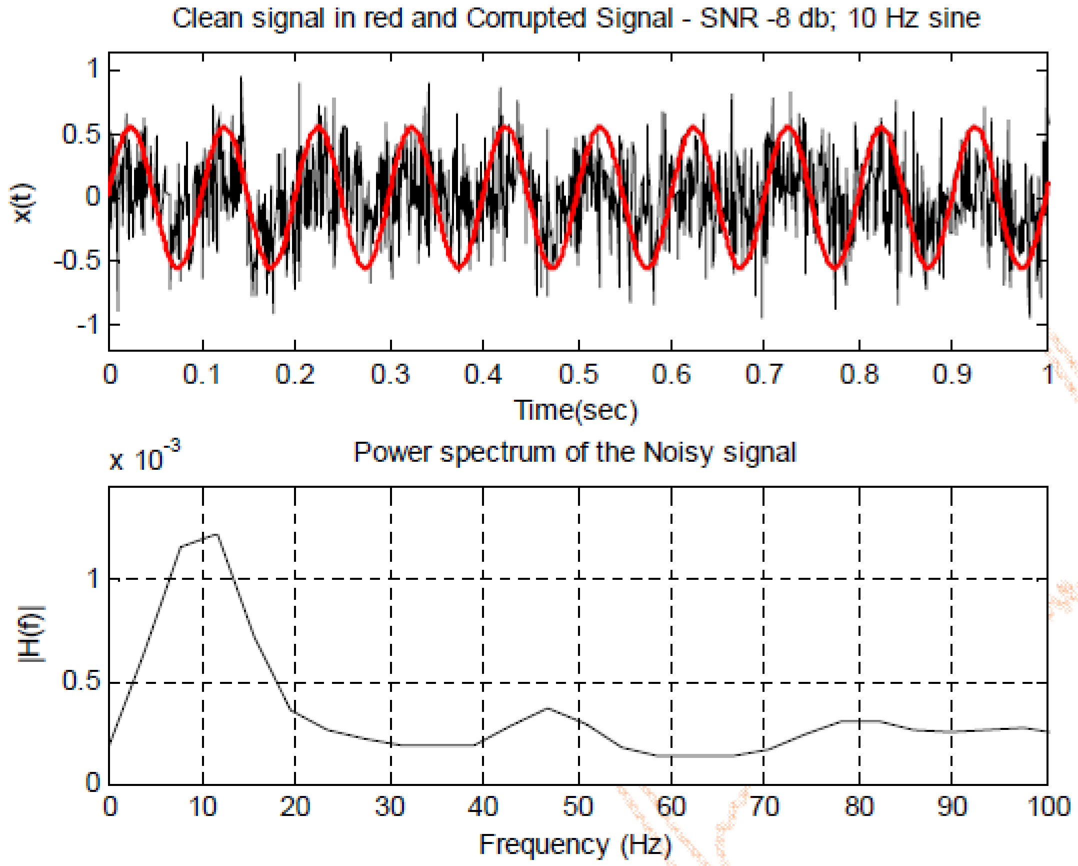
  - There are two potential problems:
    - The autocorrelation function can be near singular (it cannot be completely singular, why not…?), in which case the inverse will be unreliable
    - Even if the matrix is not near-singular, for large matrices, the inversion may take a very long time
  - These problems can usually be solved by
    - Test the condition of the matrix with $c = \mathbf{cond}(\Phi)$. If $c > 10^4$, Φ is near singular! Then, reduce the number of coefficients in the Φ, which will result in a shorter, but a more stable filter
    - Use **Levinson – Durbin** to solve the **Wiener-Hopf equations**.

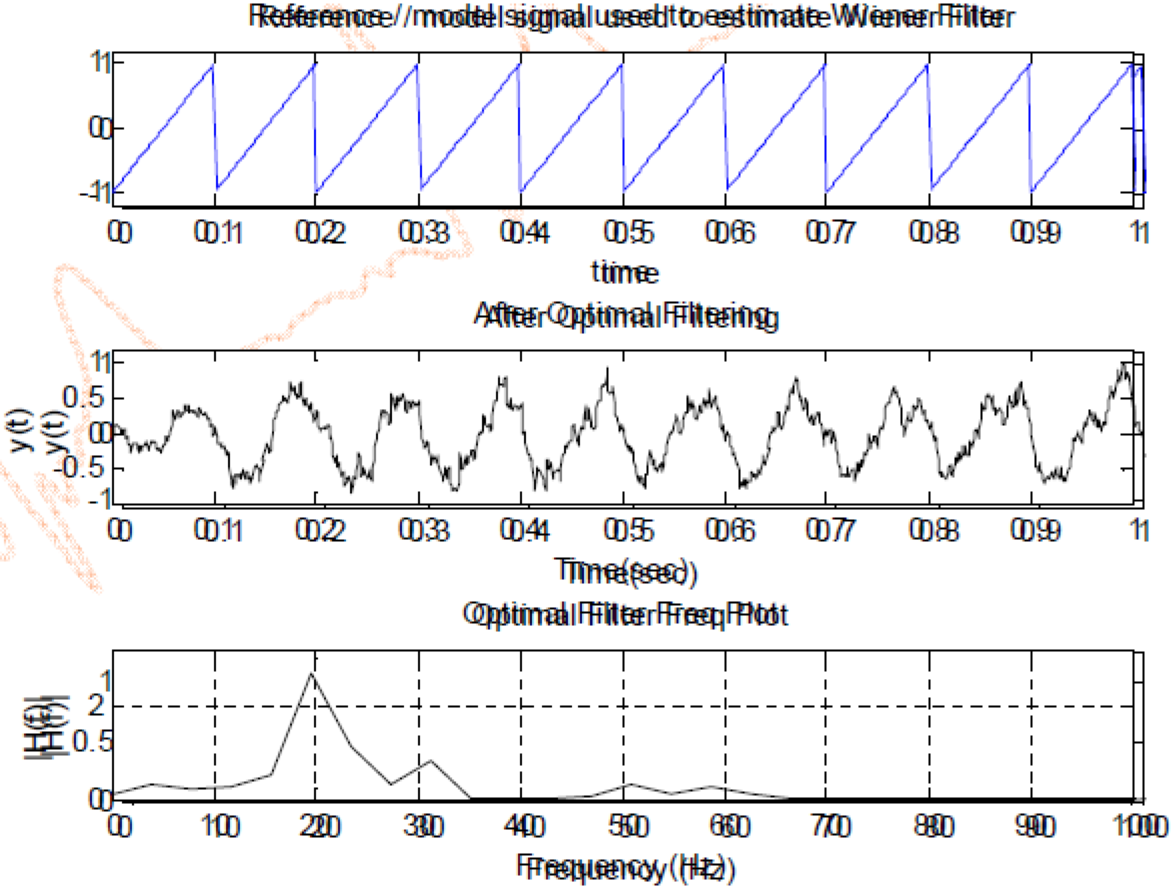
Matlab Demo


Clean signal in red and Corrupted Signal - SNR -8 db; 10 Hz sine
x(t)
Time(sec)
x 10^-3
Power spectrum of the Noisy signal
|H(f)|
Frequency (Hz)

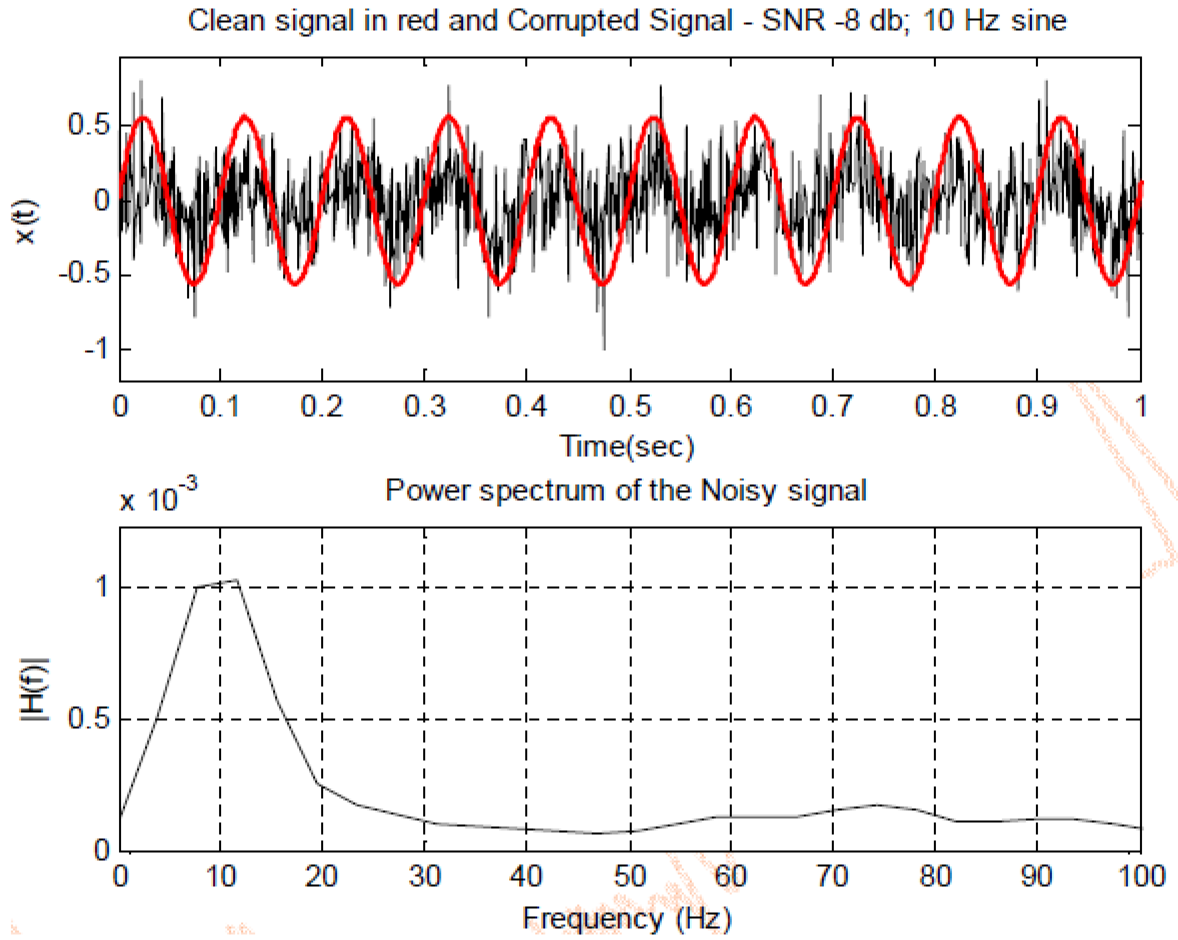

Reference / model signal used to estimate Wiener Filter
time
After Optimal Filtering
y(t)
Time(sec)
Optimal Filter Freq Plot
|H(f)|
Frequency (Hz)

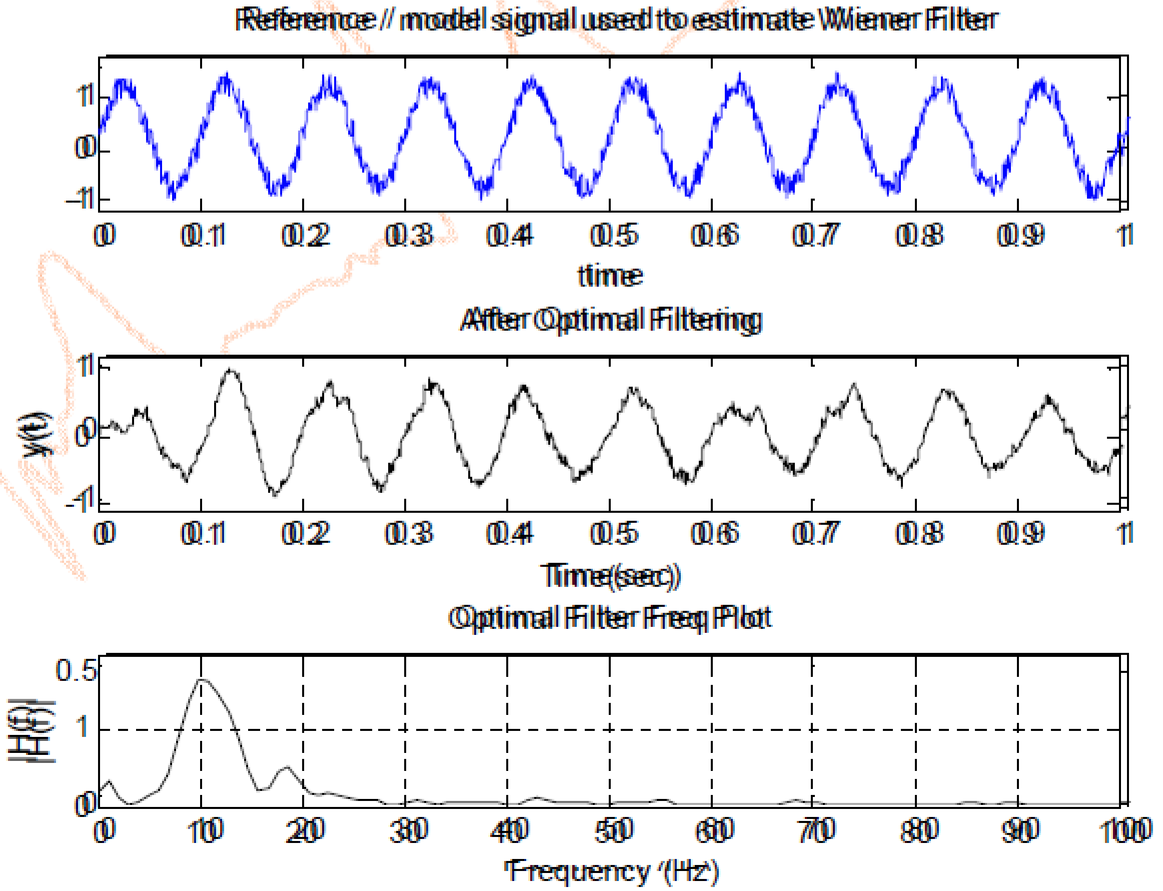
Matlab Demo


Clean signal in red and Corrupted Signal - SNR -8 db; 10 Hz sine
x(t)
Time(sec)
x 10^-3
Power spectrum of the Noisy signal
|H(f)|
Frequency (Hz)


Reference/model signal used to estimate Wiener Filter
time
After Optimal Filtering
y(t)
Time(sec)
Optimal Filter Freq Plot
|H(f)|
Frequency (Hz)

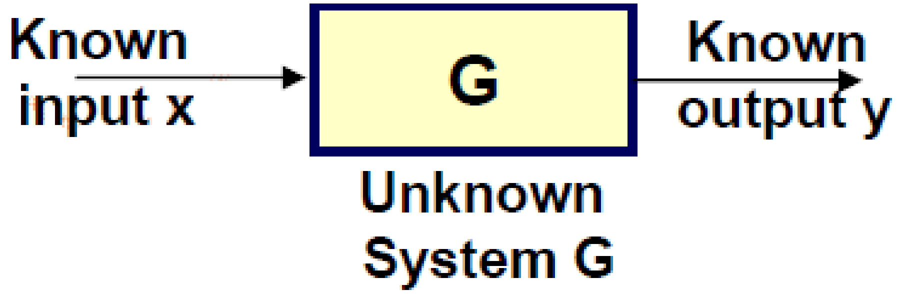
Matlab Demo


Clean signal in red and Corrupted Signal - SNR -8 db; 10 Hz sine
x(t)
Time(sec)
x 10^-3
Power spectrum of the Noisy signal
|H(f)|
Frequency (Hz)

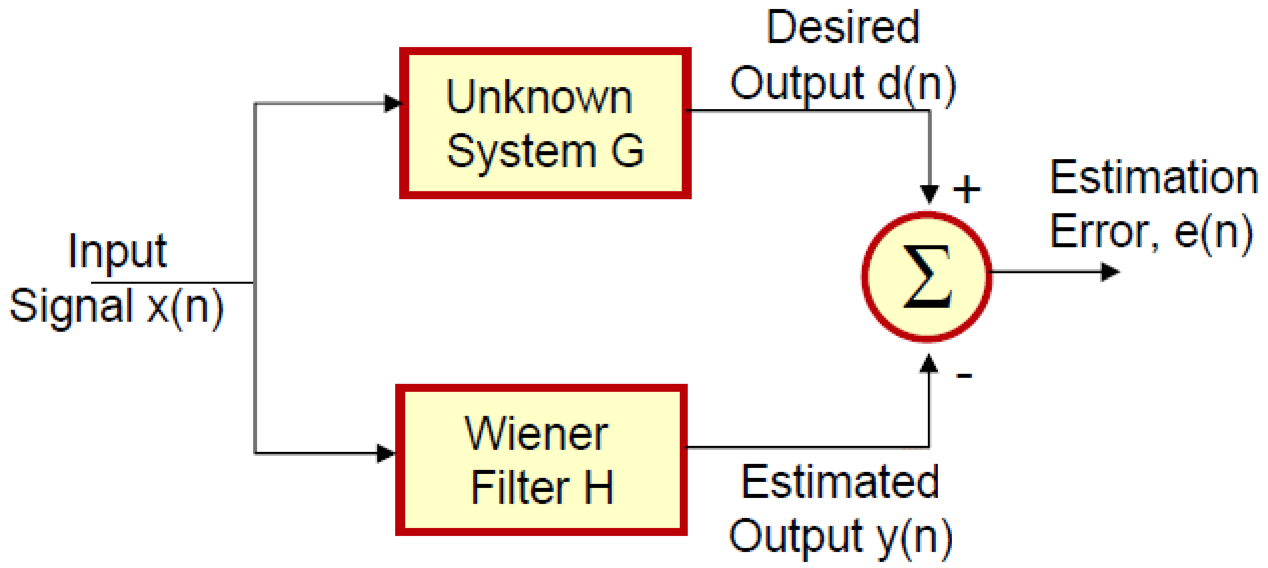

Reference / model signal used to estimate Wiener Filter
time
After Optimal Filtering
y(t)
Time(sec)
Optimal Filter Freq Plot
|H(f)|
Frequency (Hz)

System Identification Using Optimal Wiener Filter

- Wiener filters can be used for different types of problems, such as **system identification**:
  - **System identification**: There is an unknown system, and we would like to determine the characteristics (e.g., frequency response) of this system from an input / output signal pair.
    - For example to model the lungs from respiratory signals, or to model response of a prosthetic, etc.



System Identification



$$\boxed{\mathbf{h} = \Phi^{-1}\Theta}$$

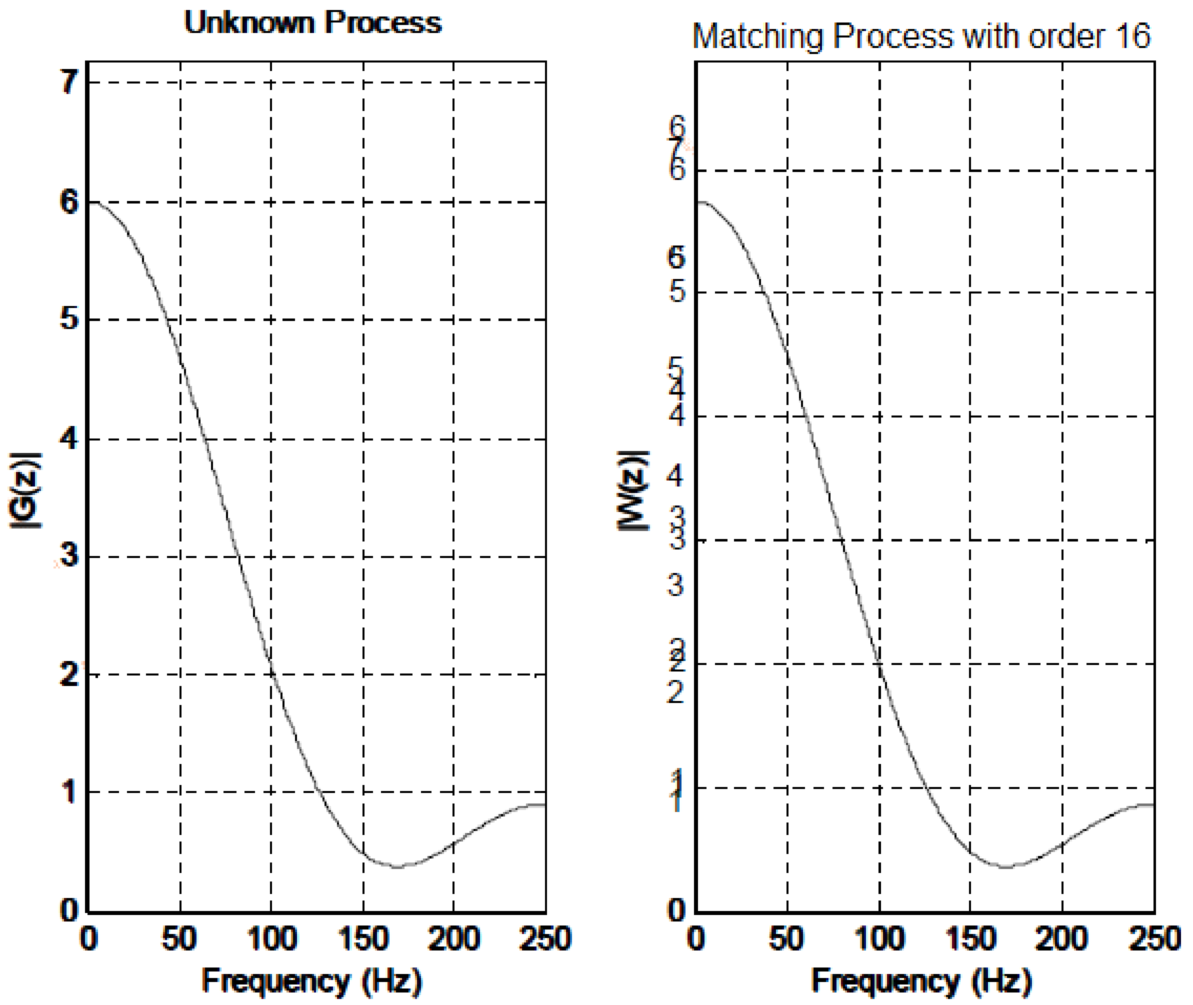
Autocorrelation of the input signal x(n)

Cross correlation of input x and desired d

- This is an example of match filter, where the wiener filter parameters are adjusted such that its output can match the desired output.
  - In this case, the unknown system output to a known input constitutes the desired (reference) signal.
  - Wiener filter will work particularly well, if the unknown system is an all-zero type system (that is, an MA type system, with no poles), since the Wiener filters are FIR filters.

System Identification

```
fs = 500; % Sampling frequency
N = 1024; % Number of points
L = 8; % Optimnal filter order
f = (1:N) * fs/N; % freq. vector for plotting
%Generate unknown system and noise input
b_unknown = [.5 .75 1.2]; % Define unknown process
xn = randn(1,N);
xd = conv(b_unknown,xn); % Generate unknown system output
xd = xd(3:N+2); % Truncate extra points
% Apply Weiner filter
b = wiener_hopf(xn,xd,L); % Compute matching filter coefficients
b = b/N; % Scale filter coefficients
% Calculate frequency characteristics
ps_wiener = (abs(fft(b,N))).^2; %Spectrum of the matched filter
ps_unknown = (abs(fft(b_unknown,N))).^2; % of the unknown system
ps_out = (abs(fft(xd))).^2;
% Plot frequency characteristic of unknown and identified processes
subplot(1,2,1);
        plot(f(1:N/2),ps_unknown(1:N/2),'k'); % Plot filtered data
        xlabel('Frequency (Hz)'); ylabel('|G(z)|');
        axis([0 fs/2 0 1.2*max(ps_unknown)]); grid
        title('Unknown Process');
subplot(1,2,2);
        plot(f(1:N/2),ps_wiener(1:N/2),'k'); % Plot filtered data
        xlabel('Frequency (Hz)'); ylabel('|W(z)|');
        title(['Matching Process with order ', num2str(L)]); grid
```



```
      1.1740    0.7337    0.4891   -0.0001
b =   0.0002    0.0002   -0.0002    0.0000
      0.0002    0.0000   -0.0000    0.0000
      0.0002   -0.0000   -0.0004   -0.0004
```

Filtering Out Noise

- Consider the problem of separating the additive noise $\eta(n)$ from the (desired) clean signal $d(n)$ in the signal $x(n) = d(n) + \eta(n)$. In vector notation:

$$\mathbf{x}(n) = \boldsymbol{d}(n) + \boldsymbol{\eta}(n)$$

- We are still looking for the optimal wiener filter: $\mathbf{w}^* = \Phi^{-1}\Theta$
- Let's look at these terms individually:
  - The autocorrelation of the input signal:

$$\Phi = E[\mathbf{x}(n)\, \mathbf{x}^{\mathrm{T}}(n)] = E[\{\boldsymbol{d}(n) + \boldsymbol{\eta}(n)\}\{\boldsymbol{d}(n) + \boldsymbol{\eta}(n)\}^{\mathrm{T}}]$$

  - Making the reasonable assumption that noise and signal are statistically independent:

$$E[\boldsymbol{d}(n)\, \boldsymbol{\eta}^{\mathbf{T}}(n)] = E[\boldsymbol{d}^{\mathbf{T}}(n)\, \boldsymbol{\eta}(n)] = 0 \text{ and}$$

$$\Phi = E[\boldsymbol{d}(n)\, \boldsymbol{d}^{\mathbf{T}}(n)] + E[\boldsymbol{\eta}(n)\, \boldsymbol{\eta}^{\mathbf{T}}(n)] = \Phi_{\boldsymbol{d}} + \Phi_{\boldsymbol{\eta}}$$

Filtering Out Noise

- As for the cross-correlation function:

M-by-1 autocorrelation vector of the desired signal

$$\Theta = E[\mathbf{x}(n)\,\boldsymbol{d}(n)] = E[\{\boldsymbol{d}(n) + \boldsymbol{\eta}(n)\}\,\boldsymbol{d}(n)]$$

$$= E[\boldsymbol{d}(n)\,\boldsymbol{d}(n)] + \underbrace{E[\boldsymbol{\eta}(n)\,\boldsymbol{d}(n)]}_{0} = E[\boldsymbol{d}(n)\,\boldsymbol{d}(n)] = \varphi_d$$

- Then we have the optimal Wiener filter as

$$\mathbf{w}^* = \Phi^{-1}\Theta = (\Phi_d + \Phi_\eta)^{-1}\varphi_d$$

- The frequency responses are then

$$\boxed{\left.\begin{aligned} S_{xx}(\omega) &= S_d(\omega) + S_\eta(\omega) \\ S_{xd}(\omega) &= S_d(\omega) \end{aligned}\right\} \Rightarrow W(\omega) = \frac{S_{xd}(\omega)}{S_{xx}(\omega)} = \frac{S_d(\omega)}{S_d(\omega) + S_\eta(\omega)}}$$

This is because $\Theta = \varphi_d$

Mini Project

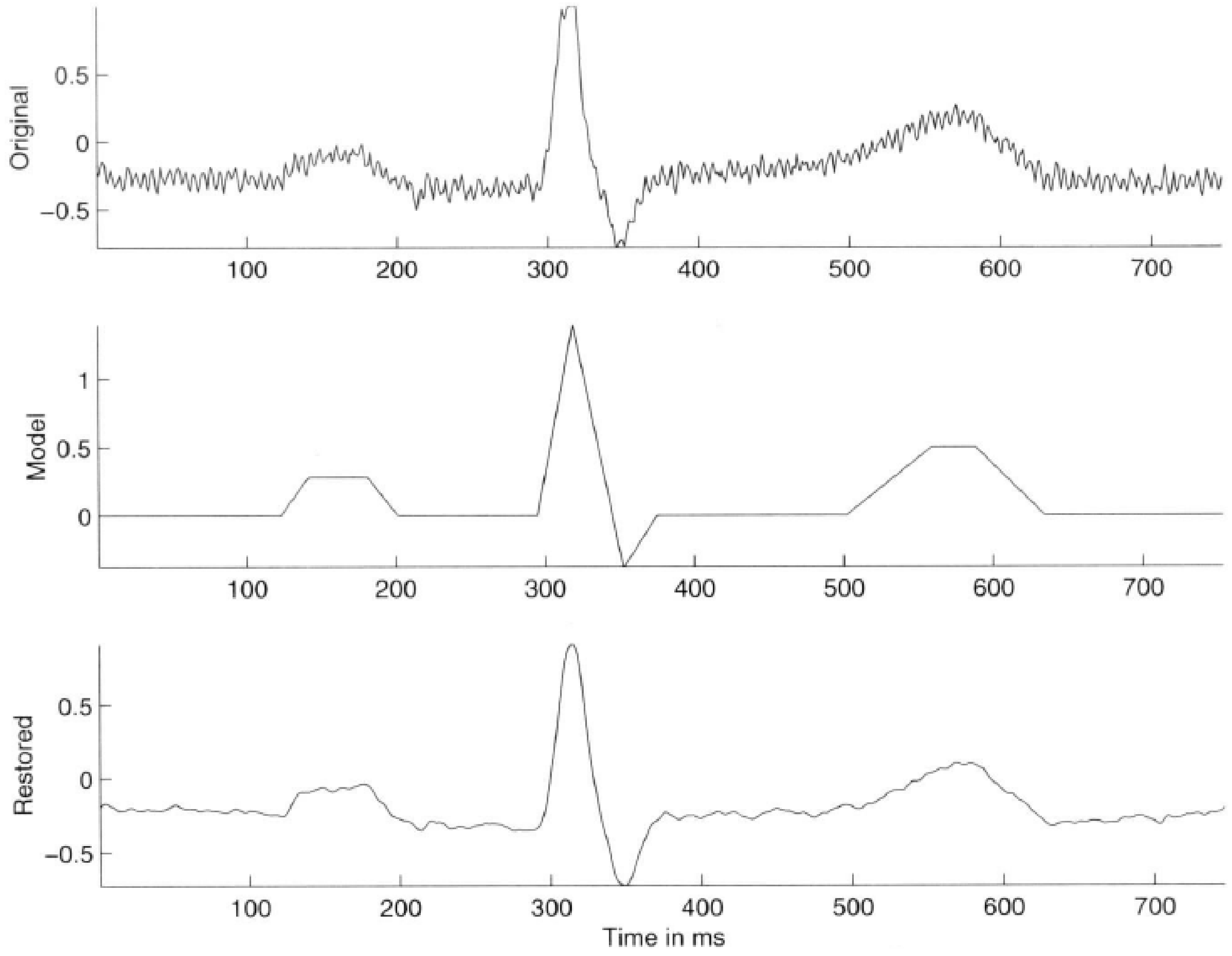
- Acquire several cycles of ECG – make sure it is noisy with, say 60 Hz noise as well as EMG noise.
- Also collect as clean of an ECG as possible to be used as reference.
  - Clean the noisy ECG using the Wiener filter approach
  - Use a canonical ECG (time synchronized) as in the figure (in the next page) as a reference ECG. How well does this work? Try to replicate the result in the figure.
- Now, add random noise to the clean ECG as well as the noisy ECG and denoise this signal using Wiener filter, using each of the references mentioned above (clean ECG, canonical ECG, etc.)
- Clearly explain and discuss your findings. Include matlab code, plots, etc.

Mini Project

Noisy ECG

Canonical Reference

Denoised ECG



Adaptive Filtering

Adaptive Filtering

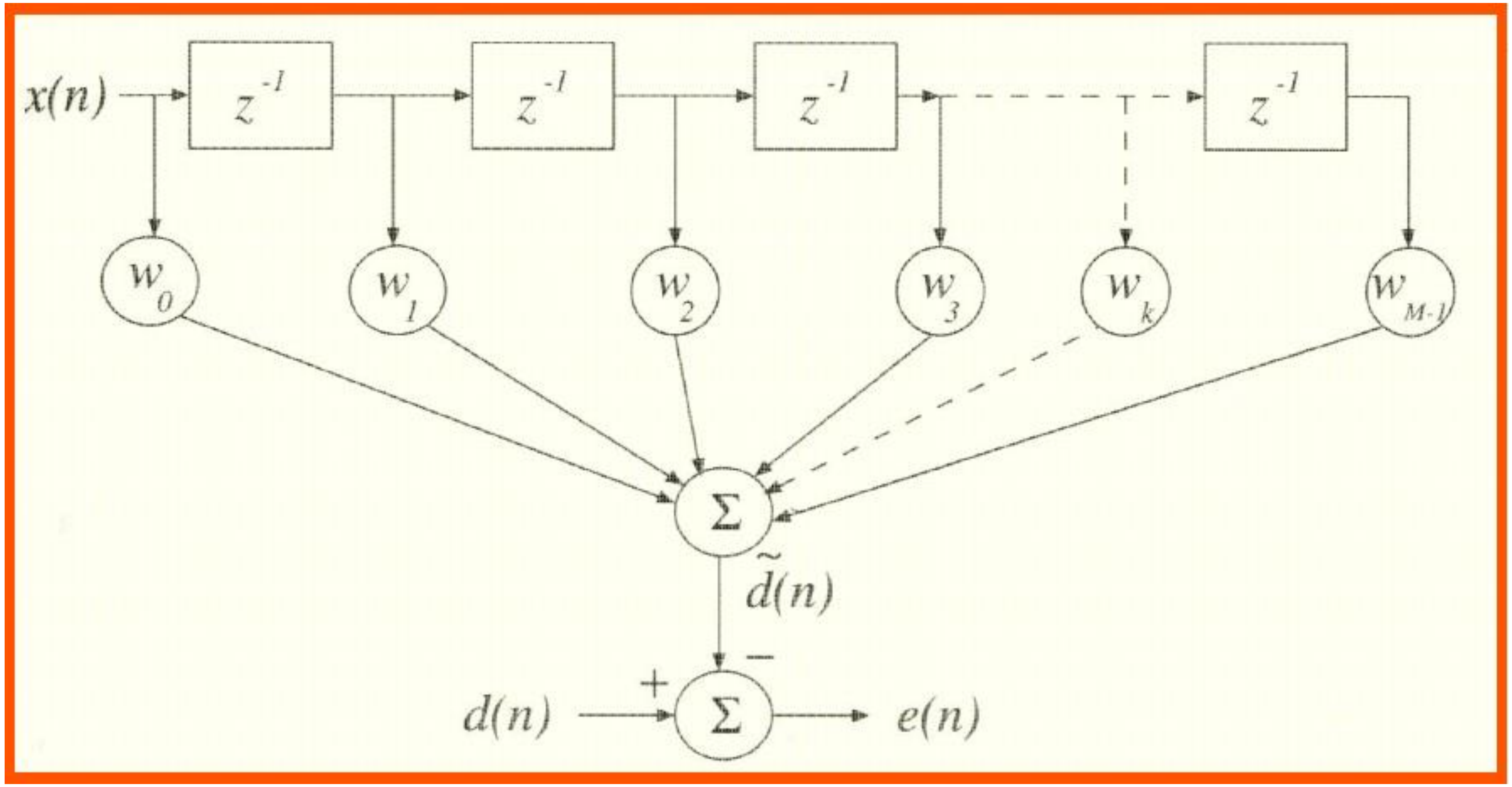
- Adaptive Filters
  - LMS algorithm
  - Gradient descent

The Case Against FIR/IIR Filters

The main disadvantage of FIR & IIR filters has already discussed :

- Despite techniques available to design very accurate filters with specific design parameters (specifications), we need such specification in the first place to design them
  - We need to know the specific spectral properties of the information carrying signal, as well as the spectral properties of the corrupting signal (noise) in order to design filters that can effectively separate the two signals
- Such information is often not available for many applications
  - In such cases we need mechanism to determine what the filter is supposed to do – or what kind of a filter we need in the first place.
- Wiener filters provide such a solution:
  - Wiener filter provides us with the (FIR type) filter coefficients that are optimal in the mean square error sense;
  - The error is computed with respect to a reference signal, which is assumed to be available.

Wiener Filter



$$J(\mathbf{w}) = E\left[e^2(n)\right] \propto \sum_{n=0}^{M-1}\left[d(n) - \sum_{k=0}^{M-1} w_k x(n-k)\right]^2$$

The Case Against The Wiener Filter

- But, the Wiener filter also has a main problem:
  - And no, it is not the requirement that we have access to a reference signal, as in most application we usually have access to some kind of a reference/model signal
- Although the Wiener filter does not require any information regarding the spectral properties of the signal or the noise, it is too **fixed** for any given application.
- Just like classical filter design techniques, Wiener filters cannot respond to changes in the signal and/or noise properties that may occur in time
  - Once the Wiener filter is designed, its coefficients are fixed and remain unchanged in time – whether the signal / noise properties do so.

Adaptive Filters

- Applications where the signal / noise characteristics change in time are common in practice, in particular in biological signals:
    - The spectral characteristics of fetus ECG that is mixed with mother's ECG vary often
    - The EMG noise that corrupts the ECG also varies in time depending on how strong the muscle contractions are
    - Often, the signal's spectral properties themselves change in time, as is often the case in EEG signals
- In such cases, where the interference (noise) is non-stationary, a filter whose characteristics can be changed to track the changes in the noise is needed. Such a filter is called an **adaptive filter**.

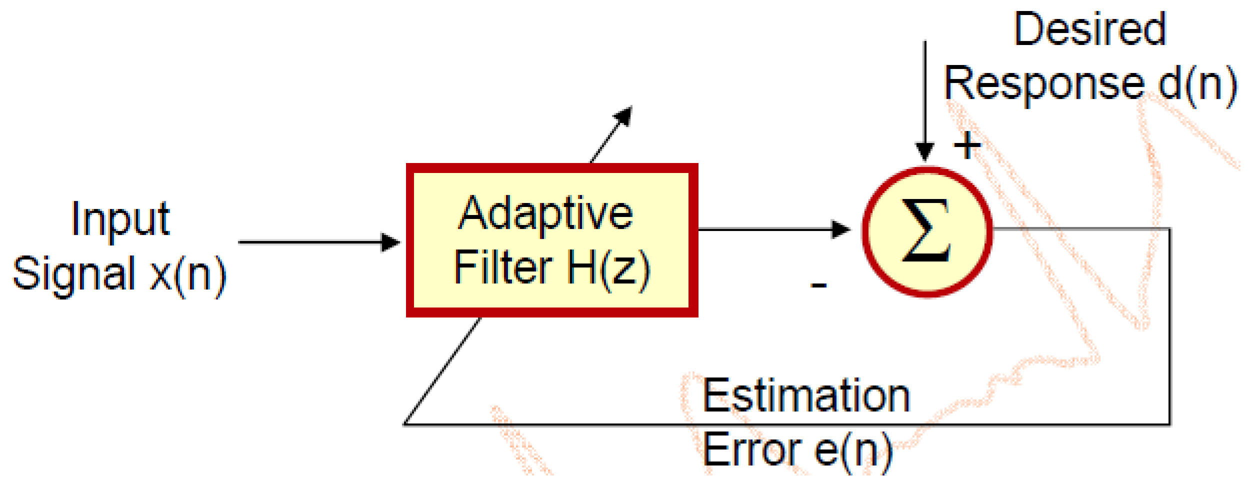
Adaptive Filters

The problem statement:

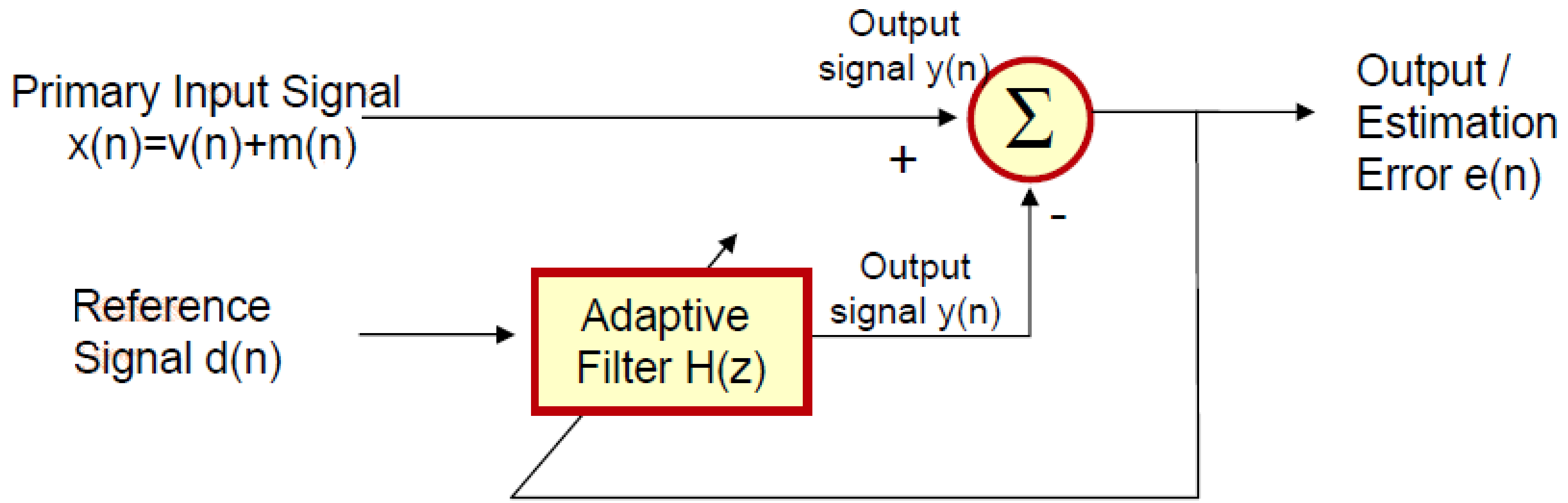
- Design an optimal filter to remove a nonstationary interference from a nonstationary signal. An additional channel of information related to the interference is assumed to be available. The filter should then continuously adapt to the changing characteristics of the signal and the interference.
    - The filter should be **adaptive**; the two weights of the filter must vary with time
    - The filter should be **optimal**.
- Such an **adaptive filter** are also known as an **adaptive noise canceller** (**ANC**)

Adaptive Filter Structure

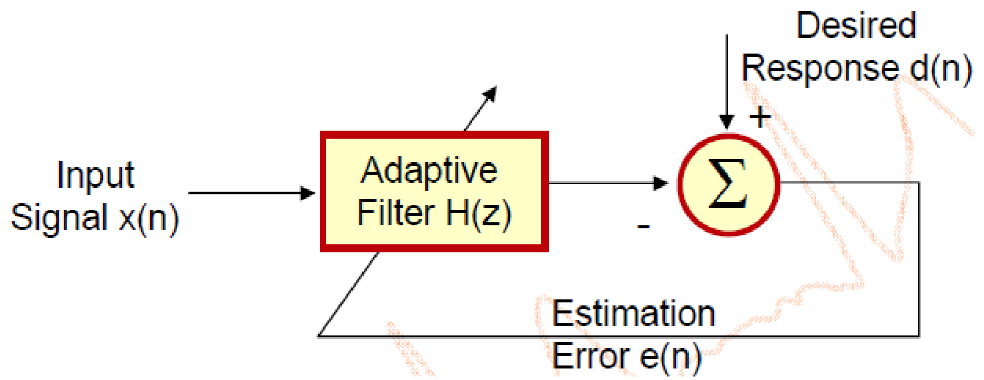
Adaptive filter model (to be analyzed first):



Alternative adaptive filter model where input is signal + noise (to be analyzed later):



How Does It Work?



- The filter taps, lets call them "b" parameters in reference to numerator coefficients of an FIR filter, are continuously updated – with every time step – through a feedback loop
  - The update is intended to bring the filters output $y(n)$ closer and closer to the desired signal $d(n)$, such that the error between them, $e(n)$ is minimized.
  - The approach used for this minimization is a well known technique in optimization, called the steepest descent, the same approach used in determining the synaptic weights of a neural network.
  - The algorithm that uses the **steepest descent** for determining optimal filter coefficients is also known as the **LMS (least mean square) algorithm**.

How Does It Work?

Similar to the Wiener case, we assume that the filter is of FIR type

- The inherent stability of FIR filters make them an attractive choice – in particular when designing **adaptive filters**, as we do not want to deal with the additional issue of stability while continuously updating the filter taps.
- The output signal $y(n)$ is once again the convolution of input and filter taps, but the filter coefficients are now themselves a function of time: $\boxed{y(n) = \sum_{k=1}^{L} b_n(k) x(n-k)}$
- Hence while in Wiener filters, the filter taps are all determined at once (in a block approach), in **adaptive filters**, they are updated continuously
  - While the Wiener filter approach can be adapted for this case, the recursive **LMS algorithm** is more attractive because it does not require the computation of auto / cross correlation functions – but rather uses the gradient method known as the steepest descent.
  - In such an approach, the taps are determined iteratively, updating the weights at each time step by an amount equal to the gradient $\boxed{b_{n+1}(k) = b_n(k) + \nabla_n(J(b))}$

Gradient Descent

- In gradient descent the goal is to find the minimum of the error function. Since in LMS type approaches, the error is "least mean square" type, it is a quadratic function:

$$J(\mathbf{b}) = E[e^2(n)]$$

- Therefore, this is a quadratic function, and has a parabolar shape. If we consider the error as a one dimensional function, then it is a parabola (quadratic). If we consider it as a two dimensional function (two weights) it is a parabolic surface. If we have more than two weights, it is a higher dimensional surface, called parabolic hypersurface.
- In gradient descent type approaches, we are then iteratively trying to find the minimum point of this parabolic hypersurface, one step at a time.

Gradient (Steepest) Descent

- We are again interested in minimizing an error function, much similar to the one used for Wiener filter:

$$\boxed{J(\mathbf{b}) = E\left[e^2(n)\right] \propto \sum_{n=0}^{L-1}\left[d(n) - \sum_{k=0}^{L-1} b_n(k)x(n-k)\right]^2}$$

- In this case, we iteratively adjust the weights based on the negative gradient of the error with respect to the filter coefficients $b_n(k)$

$$\boxed{\nabla_n J(\mathbf{b}) = \nabla_n = \frac{\partial J(\mathbf{b})}{\partial b_n(k)} = \frac{\partial e^2(n)}{\partial b_n(k)} = 2e(n)\frac{\partial\left(d(n) - y(n)\right)}{\partial b_n(k)}}$$

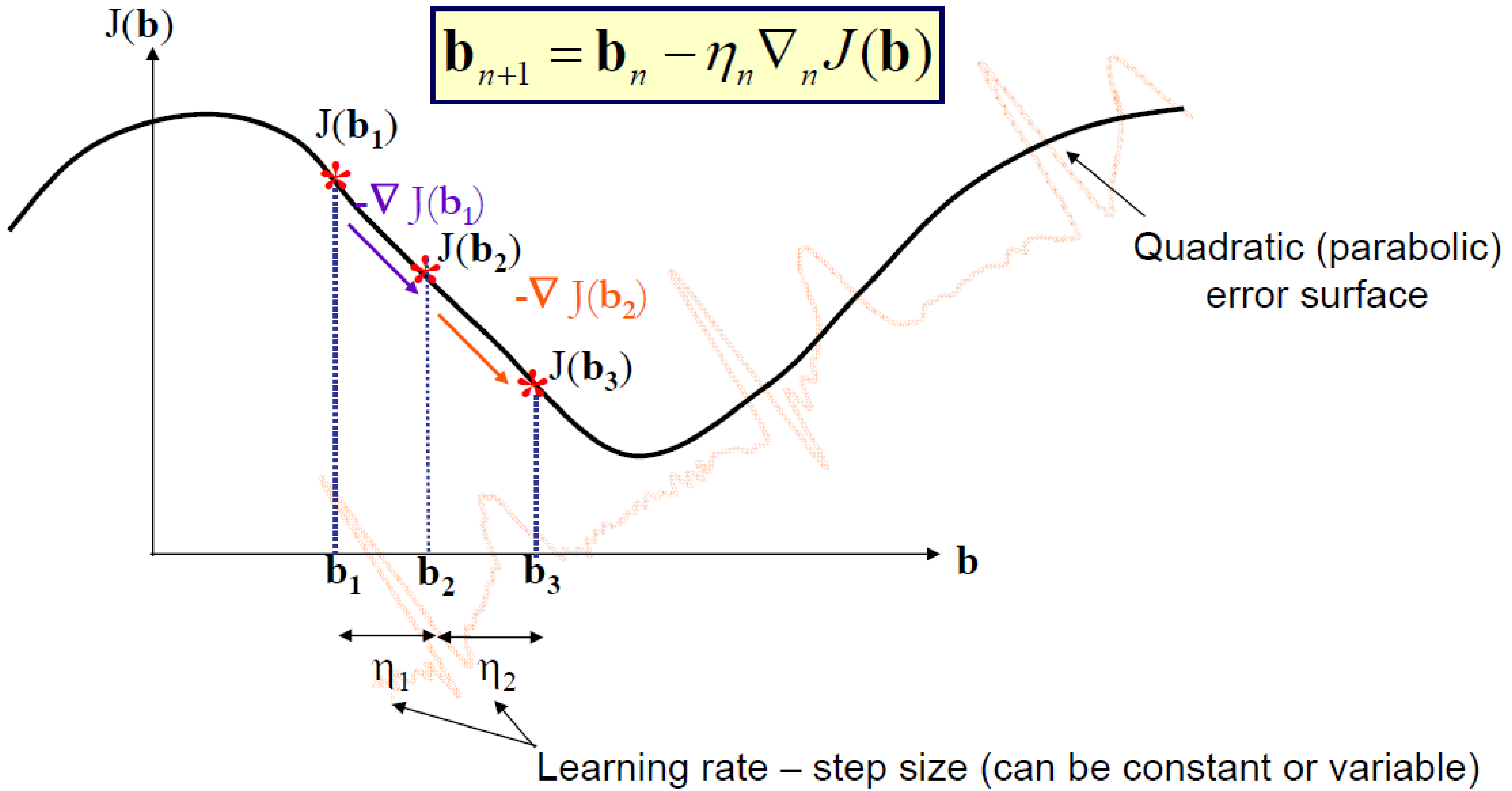
  - Note that $d(n)$ is independent of $b_n(k)$, and the partial derivative of $y(n)$ with respect to $b_n(k)$ is just $x(n\text{-}k)$. Then, the gradient can be written in terms of the instantaneous product of error and the input: $\boxed{\nabla_n = 2e(n)x(n-k)}$

Gradient Descent

- The update rule is then $\boxed{b_{n+1}(k) = b_n(k) + \nabla_n(J(b_n(k))) = b_n(k) + \eta\, e(n)\, x(n\text{-}k)}$ where η is a constant, called the **learning rate**, which controls the rate of descent.
  - Large values of the **learning rate** allows the algorithm to take big steps towards the solution, but risks missing the minimum error point. Small values of **learning rate** takes smaller steps, and hence may causes slow convergence and hence unnecessary computation time.
  - A common rule of thumb is usually to choose the **learning rate** as where L is the filter order, and P_x is the power of the input signal: $\boxed{0 < \eta < \frac{1}{10LP_x}}$ $\boxed{P_x \approx \frac{1}{N}\sum_{n=0}^{N-1} x^2(n)}$
- All confused…?
  - Remember that the error is quadratic, hence we are dealing with a parabolic hypersurface, and we are trying to find the "bottom" of this surface, one step at a time. The **learning rate** is then the step size.

Gradient Descent

$$\boxed{\mathbf{b}_{n+1} = \mathbf{b}_n - \eta_n \nabla_n J(\mathbf{b})}$$



Learning rate – step size (can be constant or variable)