

MIT OpenCourseWare  
<http://ocw.mit.edu>

2.161 Signal Processing: Continuous and Discrete  
Fall 2008

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.

**Lecture 25**<sup>1</sup>

**Reading:**

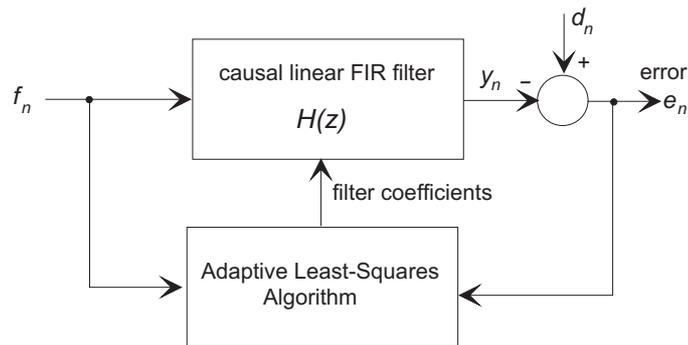
- Class Handout: *Introduction to Least-Squares Adaptive Filters*
- Class Handout: *Introduction to Recursive-Least-Squares (RLS) Adaptive Filters*
- Proakis and Manolakis: Secs. 13.1 – 13.3

## 1 Adaptive Filtering

In Lecture 24 we looked at the least-squares approach to FIR filter design. The filter coefficients  $b_m$  were generated from a one-time set of experimental data, and then used subsequently, with the assumption of stationarity. In other words, the design and utilization of the filter were decoupled.

We now extend the design method to *adaptive FIR filters*, where the coefficients are continually adjusted on a step-by-step basis during the filtering operation. Unlike the static least-squares filters, which assume stationarity of the input, adaptive filters can track slowly changing statistics in the input waveform.

The adaptive structure is shown in below. The adaptive filter is FIR of length  $M$  with coefficients  $b_k$ ,  $k = 0, 1, 2, \dots, M - 1$ . The input stream  $\{f(n)\}$  is passed through the filter to produce the sequence  $\{y(n)\}$ . At each time-step the filter coefficients are updated using an error  $e(n) = d(n) - y(n)$  where  $d(n)$  is the desired response (usually based of  $\{f(n)\}$ ).



The filter is not designed to handle a particular input. Because it is adaptive, it can adjust to a broadly defined task.

---

<sup>1</sup>copyright © D.Rowell 2008

## 1.1 The Adaptive LMS Filter Algorithm

### 1.1.1 Simplified Derivation

In the length  $M$  FIR adaptive filter the coefficients  $b_k(n)$ ,  $k = 1, 2, \dots, M-1$ , at time step  $n$  are adjusted continuously to minimize a step-by-step squared-error performance index  $J(n)$ :

$$J(n) = e^2(n) = (d(n) - y(n))^2 = \left( d(n) - \sum_{k=0}^{M-1} b(k)f(n-k) \right)^2$$

$J(n)$  is described by a quadratic surface in the  $b_k(n)$ , and therefore has a single minimum. At each iteration we seek to reduce  $J(n)$  using the “steepest descent” optimization method, that is we move each  $b_k(n)$  an amount proportional to  $\partial J(n)/\partial b(k)$ . In other words at step  $n+1$  we modify the filter coefficients from the previous step:

$$b_k(n+1) = b_k(n) - \Lambda(n) \frac{\partial J(n)}{\partial b_k(n)}, \quad k = 0, 1, 2, \dots, M-1$$

where  $\Lambda(n)$  is an empirically chosen parameter that defines the step size, and hence the rate of convergence. (In many applications  $\Lambda(n) = \Lambda$ , a constant.) Then

$$\frac{\partial J(n)}{\partial b_k} = \frac{\partial e^2(n)}{\partial b_k} = 2e(n) \frac{\partial e(n)}{\partial b_k} = -2e(n)f(n-k)$$

and the fixed-gain FIR *adaptive Least-Mean-Square (LMS) filter algorithm* is

$$b_k(n+1) = b_k(n) + \Lambda e(n)f(n-k), \quad k = 0, 1, 2, \dots, M-1$$

or in matrix form

$$\mathbf{b}(n+1) = \mathbf{b}(n) + \Lambda e(n)\mathbf{f}(n),$$

where

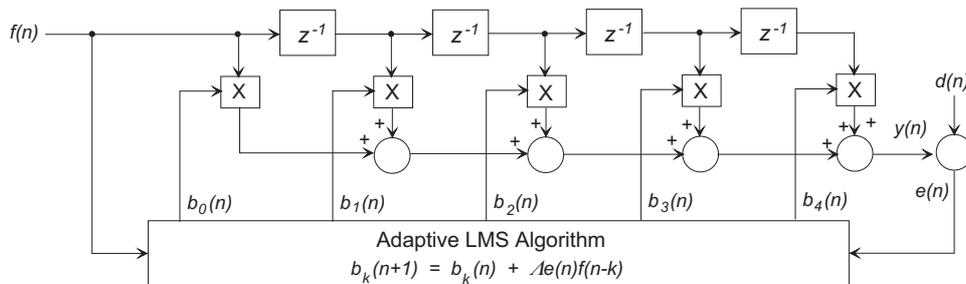
$$\mathbf{b}(n) = [b_0(n) \quad b_1(n) \quad b_2(n) \quad \dots \quad b_{M-1}(n)]^T$$

is a column vector of the filter coefficients, and

$$\mathbf{f}(n) = [f(n) \quad f(n-1) \quad f(n-2) \quad \dots \quad f(n-(M-1))]^T$$

is a vector of the recent history of the input  $\{f(n)\}$ .

A Direct-Form implementation for a filter length  $M = 5$  is



### 1.1.2 Expanded Derivation

A more detailed derivation of the LMS algorithm (leading to the same result) is given in the class handout *Introduction to Least-Squares Adaptive Filters*, together with a brief discussion of the convergence properties.

### 1.1.3 A MATLAB Tutorial Adaptive Least-Squares Filter Function

```
% -----
% 2.161 Classroom Example - LSadapt - Adaptive Least-squares FIR filter
%                               demonstration
% Usage :  1) Initialization:
%           y = LSadapt('initial', Lambda, FIR_N)
%           where Lambda is the convergence rate parameter.
%           FIR_N is the filter length.
%           Example:
%           [y, e] = adaptfir('initial', .01, 51);
%           Note: LSadapt returns y = 0 for initialization
%           2) Filtering:
%           [y, b] = adaptfir(f, d);
%           where f is a single input value,
%           d is the desired input value, and
%           y is the computed output value,
%           b is the coefficient vector after updating.
%
% Version:  1.0
% Author:   D. Rowell   12/9/07
% -----
function [y, bout] = LSadapt(f, d ,FIR_M)
persistent f_history b lambda M
%
% The following is initialization, and is executed once
%
if (ischar(f) && strcmp(f,'initial'))
    lambda = d;
    M = FIR_M;
    f_history = zeros(1,M);
    b = zeros(1,M);
    b(1) = 1;
    y = 0;
else
% Update the input history vector:
    for J=M:-1:2
        f_history(J) = f_history(J-1);
```

```

    end;
    f_history(1) = f;
% Perform the convolution
    y = 0;
    for J = 1:M
        y = y + b(J)*f_history(J);
    end;
% Compute the error and update the filter coefficients for the next iteration
    e = d - y;
    for J = 1:M
        b(J) = b(J) + lambda*e*f_history(J);
    end;
    bout=b;
end

```

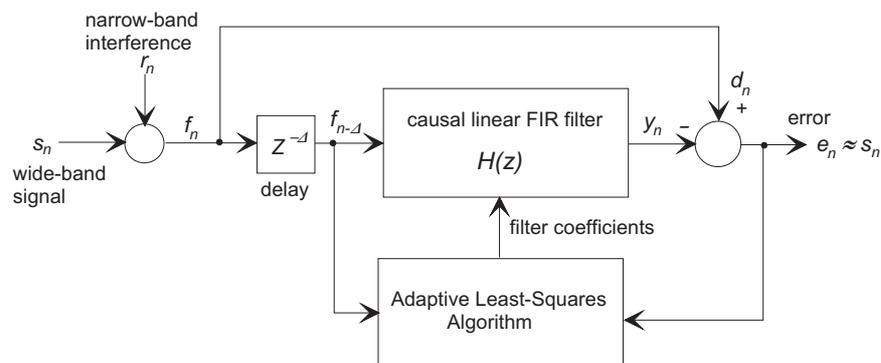
#### 1.1.4 Application Example - Suppression of Narrow-band Interference in a Wide-band Signal

Consider an adaptive filter application of suppressing narrow band interference, or in terms of correlation functions we assume that the desired signal has a narrow auto-correlation function compared to the interfering signal.

Assume that the input  $\{f(n)\}$  consists of a wide-band signal  $\{s(n)\}$  that is contaminated by a narrow-band interference signal  $\{r(n)\}$  so that

$$f(n) = s(n) + r(n).$$

The filtering task is to suppress  $r(n)$  without detailed knowledge of its structure. Consider the filter shown below:



This is similar to the basic LMS structure, with the addition of a delay block of  $\Delta$  time steps in front of the filter, and the definition that  $d(n) = f(n)$ . The overall filtering operation is a little unusual in that the error sequence  $\{e(n)\}$  is taken as the output. The FIR filter is used to predict the narrow-band component so that  $y(n) \approx r(n)$ , which is then subtracted from  $d(n) = f(n)$  to leave  $e(n) \approx s(n)$ .

The delay block is known as the *decorrelation delay*. Its purpose is to remove any cross-correlation between  $\{d(n)\}$  and the wide-band component of the input to the filter

$\{s(n - \Delta)\}$ , so that it will not be predicted. In other words it assumes that

$$\phi_{ss}(\tau) = 0, \quad \text{for } |\tau| > \Delta.$$

This least squares structure is similar to a  $\Delta$ -step linear predictor. It acts to predict the current narrow-band (broad auto-correlation) component from the past values, while rejecting uncorrelated components in  $\{d(n)\}$  and  $\{f(n - \Delta)\}$ .

If the LMS filter transfer function at time-step  $n$  is  $H_n(z)$ , the overall suppression filter is FIR with transfer function  $H(z)$ :

$$\begin{aligned} H(z) &= \frac{E(z)}{F(z)} = \frac{F(z) - z^{-\Delta}H_n(z)F(z)}{F(z)} \\ &= 1 - z^{-\Delta}H_n(z) \\ &= z^0 + 0z^{-1} + \dots + 0z^{-(\Delta-1)} - b_0(n)z^{-\Delta} - b_1(n)z^{-(\Delta+1)} + \dots \\ &\quad \dots - b_{M-1}(n)z^{-(\Delta+M-1)} \end{aligned}$$

that is, a FIR filter of length  $M + \Delta$  with impulse response  $h'(k)$  where

$$h'(k) = \begin{cases} 1 & k = 0 \\ 0 & 1 \leq k < \Delta \\ -b_{k-\Delta}(n) & \Delta \leq k \leq M + \Delta - 1 \end{cases}$$

and with frequency response

$$H(e^{j\Omega}) = \sum_{k=0}^{M+\Delta-1} h'(k)e^{-jk\Omega}.$$

The filter adaptation algorithm is the same as described above, with the addition of the delay  $\Delta$ , that is

$$\mathbf{b}(n+1) = \mathbf{b}(n) + \Lambda e(n)\mathbf{f}(n - \Delta)$$

or

$$b_k(n+1) = b_k(n) + \Lambda e(n)f((n - \Delta) - k), \quad k = 0, 1, 2, \dots, M - 1.$$

## ■ Example 1

The frequency domain Characteristics of an LMS Suppression Filter:

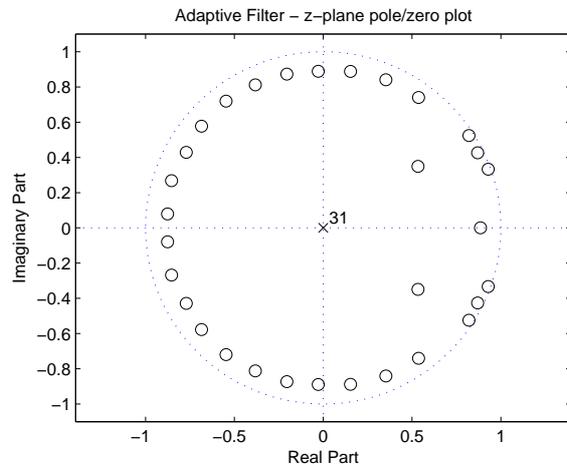
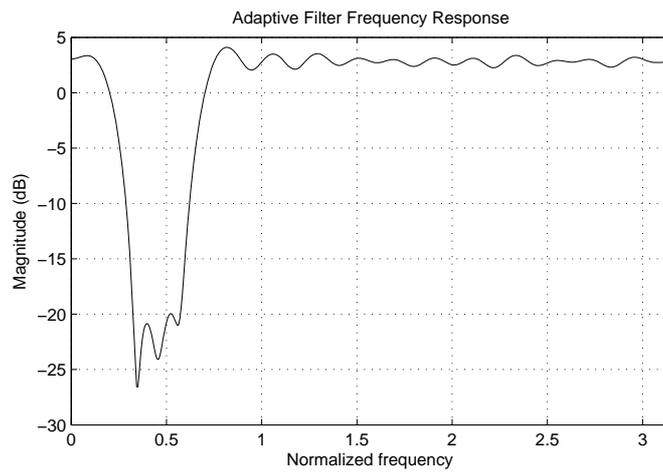
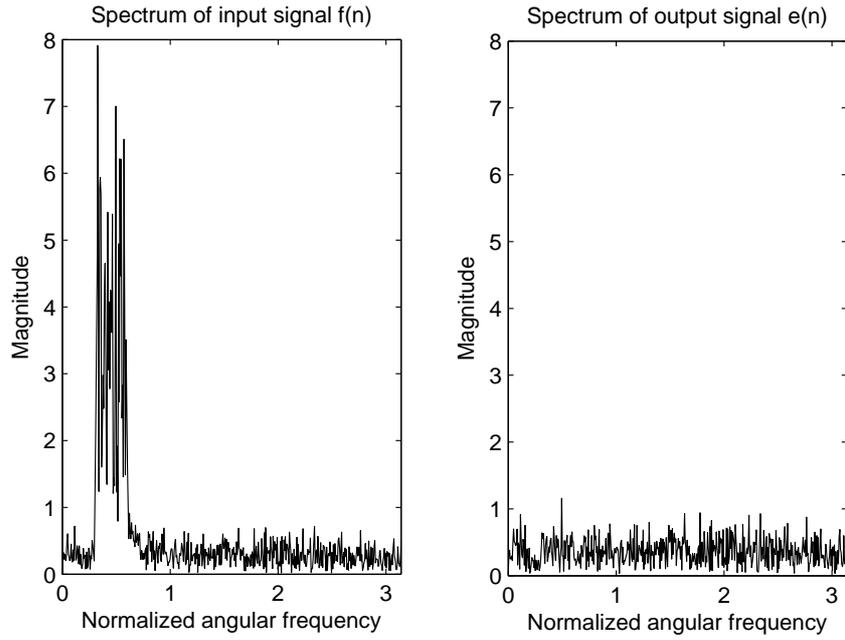
This example demonstrates the filter characteristics of an adaptive LMS filter after convergence. The interfering signal is comprised of 100 sinusoids with random phase and random frequencies  $\omega$  between 0.3 and 0.6. The “signal” is white noise. The filter used has  $M = 31$ ,  $\Delta = 1$ , and  $\Lambda$  was adjusted to give a reasonable convergence rate. The overall system  $H(z) = 1 - z^{-\Delta}H_n(z)$  frequency response magnitude is then computed and plotted, along with the  $z$ -plane pole-zero plot.

```

% The frequency domain filter characteristics of an interference
% suppression filter with finite bandwidth interference
%
% Create the interference as a closely packed sum of sinusoids
% between  $0.3\pi < \Omega < 0.6\pi$  with random frequency and phase
phase = 2*pi*rand(1,100);
freq = 0.3 + 0.3*rand(1,100);
f = zeros(1,100000);
for J=1:100000
    f(J) = 0;
    for k = 1:100
        f(J) = f(J) + sin(freq(k)*J + phase(k));
    end
end
% The "signal" is white noise
signal = randn(1,100000);
f = .005*f + 0.01*signal;
% Initialize the filter with M = 31 , Delta =1
% Choose filter gain parameter Lambda = 0.1
Delta = 1; Lambda = 0.5; M = 31;
x = LSadapt('initial',Lambda, M);
% Filter the data
f_delay = zeros(1,Delta+1);
y = zeros(1,length(f));
e = zeros(1,length(f));
for J = 1:length(f)
    for K = Delta+1:-1:2
        f_delay(K) = f_delay(K-1);
    end
    f_delay(1) = f(J);
    [y(J),b] = LSadapt(f_delay(Delta+1),f(J));
    e(J) = f(J) - y(J);
end;
% Compute the overall filter coefficients
%  $H(z) = 1 - z^{-\Delta}H_{\text{LMS}}(z)$ 
b_overall = [1 zeros(1,Delta-1) -b];
% Find the frequency response
[H,w] = freqz(b_overall,1);
zplane(b_overall,1)

```

The following plots show (i) the input and output spectra, the filter frequency response magnitude, and (iii) the pole-zero plot of the filter. Note that the zeros have been placed over the spectral region ( $0.3 < \omega < 0.6$ ) to create the band-reject characteristic.



---

---

## ■ Example 2

Suppression of a “Sliding” Sinusoid Superimposed on a Voice Signal:

In this example we demonstrate the suppression of a sinusoid with a linearly increasing frequency superimposed on a voice signal. The filtering task is to task is to suppress the sinusoid so as to enhance the intelligibility of the speech. The male voice signal used in this example was sampled at  $F_s = 22.05$  kHz for a duration of approximately 8.5 sec. The interference was a sinusoid

$$r(t) = \sin(\psi(t)) = \sin\left(2\pi\left(t + \frac{F_s}{150}t^2\right)\right)$$

where  $F_s = 22.05$  kHz is the sampling frequency. The instantaneous angular frequency  $\omega(t) = d\psi(t)/dt$  is therefore

$$\omega(t) = 2\pi(50 + 294t) \text{ rad/s}$$

which corresponds to a linear frequency sweep from 50 Hz to approx 2550 Hz over the course of the 8.5 second message. In this case the suppression filter must track the changing frequency of the sinusoid.

```
% Suppression of a frequency modulated sinusoid superimposed on speech.
% Read the audio file and add the interfering sinusoid
[f,Fs,Nbits] = wavread('crash');
for J=1:length(f)
    f(J) = f(J) + sin(2*pi*(50+J/150)*J/Fs);
end
wavplay(f,Fs);
% Initialize the filter
M = 55; Lambda = .01; Delay = 10;
x = LSadapt('initial', Lambda, M);
y = zeros(1,length(f));
e = zeros(1,length(f));
b = zeros(length(f),M);
f_delay = zeros(1,Delay+1);
% Filter the data
for J = 1:length(f)
    for K = Delay+1:-1:2
        f_delay(K) = f_delay(K-1);
    end
    f_delay(1) = f(J);
    [y(J),b1] = LSadapt(f_delay(Delay+1),f(J));
    e(J) = f(J) - y(J);
```

```

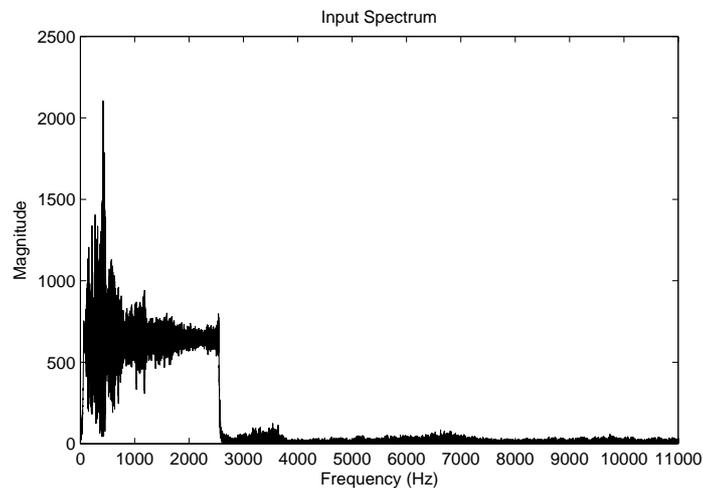
        b(J,:) = b1;
    end;
%
    wavplay(e,Fs);

```

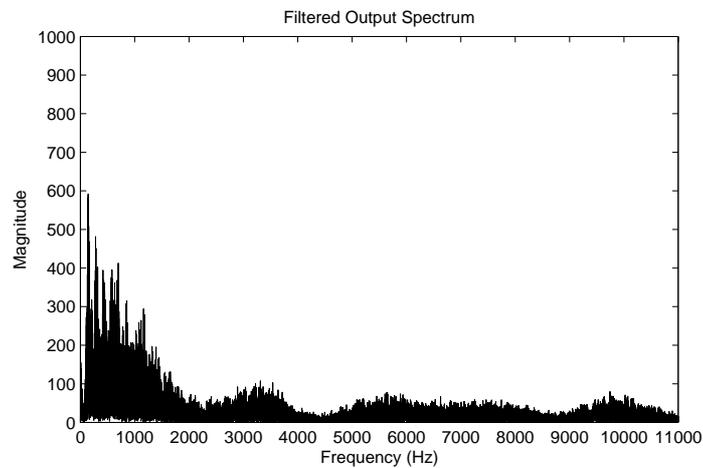
The script reads the sound file, adds the interference waveform and plays the file. It then filters the file and plays the resulting output. After filtering the sliding sinusoid can only be heard very faintly in the background. There is some degradation in the quality of the speech, but it is still very intelligible.

This example was demonstrated in class at this time.

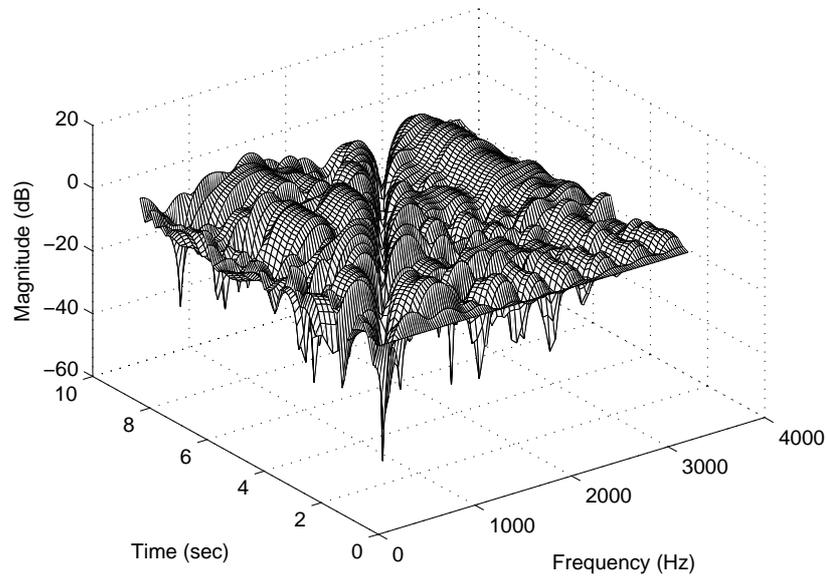
The following plot shows the waveform spectrum before filtering. The superposition of the speech spectrum on the pedestal spectrum of the swept sinusoid can be clearly seen.



The pedestal has clearly been removed after filtering, as shown below.



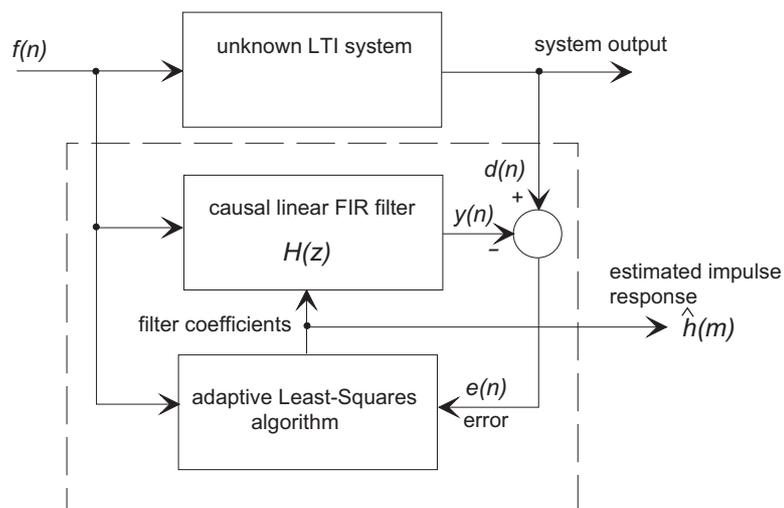
The magnitude of the frequency response filter as a meshed surface plot, with time as one axis and frequency as the other. The rejection notch is clearly visible, and can be seen to move from a low frequency at the beginning of the message to approximately 2.5 kHz at the end.



### ■ Example 3

#### Adaptive System Identification:

An adaptive LMS filter may be used for real-time system identification, and will track slowly varying system parameters. Consider the structure shown in below.



A linear system with an unknown impulse response is excited by wide-band excitation  $f(n)$ . The adaptive, length  $M$  FIR filter works in parallel with the system, with the same input. Since it is an FIR filter, its impulse response is the same as the filter coefficients, that is

$$h(m) = b(m), \quad \text{for } m = 0, 1, 2, \dots, M - 1.$$

and with the error  $e(n)$  defined as the difference between the system and filter outputs, the minimum MSE will occur when the filter mimics the system, at which time the estimated system impulse  $\hat{h}(m)$  response may be taken as the converged filter coefficients.

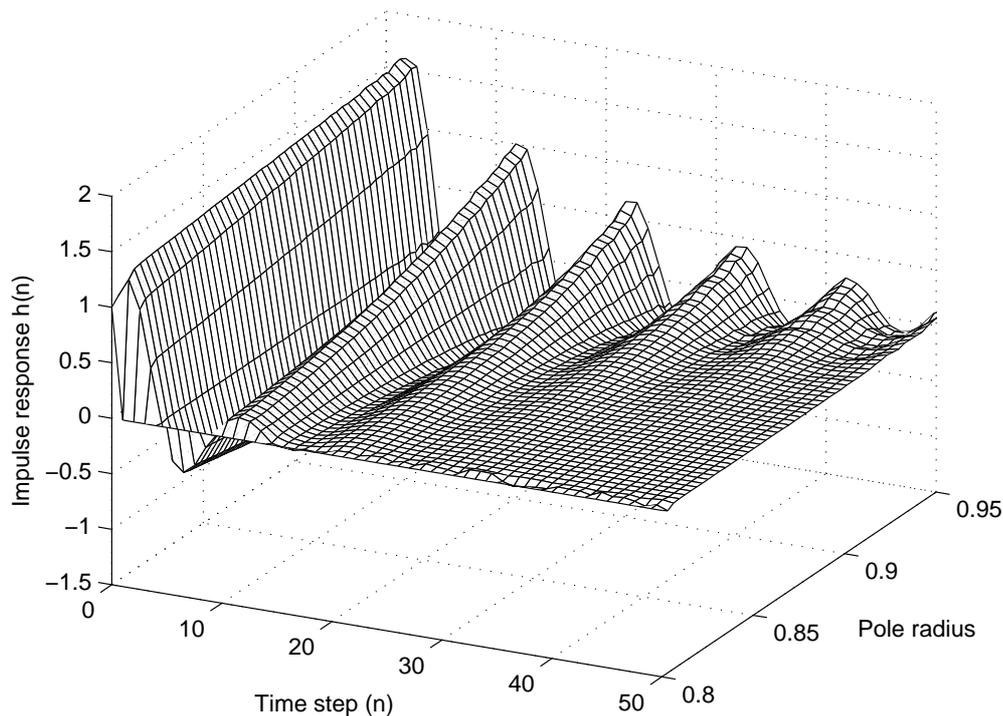
Consider a second-order “unknown” system with poles at  $z_1, z_2 = Re^{\pm j\theta}$ , that is with transfer function

$$H(z) = \frac{1}{1 - 2R\cos(\theta)z^{-1} + R^2z^{-2}},$$

where the radial pole position  $R$  varies slowly with time. The following MATLAB script uses `LSadapt()` to estimate the impulse response with 10,000 samples of gaussian white noise as the input, while the poles migrate from  $z_1, z_2 = 0.8e^{\pm j\pi/5}$  to  $0.95e^{\pm j\pi/5}$

```
% Adaptive SysID
f = randn(1,10000);
% Initialize the filter with M = 2, Delta = .8
% Choose filter gain parameter Lambda = 0.1
Lambda = 0.01; M = 51;
x = LSadapt('initial',Lambda,M);
% Define the "unknown" system
R0 = .8; R1 = 0.95; ctheta = cos(pi/5);
delR = (R1-R0)/L;
L = length(f);
b=zeros(M,L);
ynminus2 = 0; ynminus1 = 0;
for J = 1:L
% Solve the difference equation to determine the system output
% at this iteration
R = R0 + delR*(J-1);
yn = 2*R*ctheta*ynminus1 - R^2*ynminus2 + f(J);
ynminus2 = ynminus1;
ynminus1 = y;
[yout,b(:,J)] = LSadapt(f(J),yn);
end;
```

The following plot shows the estimated impulse response,  $\hat{h}(m) = b(m)$ , as the poles approach the unit circle during the course of the simulation, demonstrating that the adaptive algorithm is able to follow the changing system dynamics.



## 1.2 The Recursive Least-Squares Filter Algorithm

The recursive-least-squares (RLS) FIR filter is an alternative to the LMS filter described above, where the coefficients are continually adjusted on a step-by-step basis during the filtering operation. The filter structure is similar to the LMS filter but differs in the internal algorithmic structure.

Like the LMS filter, the RLS filter is FIR of length  $M$  with coefficients  $b_k$ ,  $k = 0, 1, 2, \dots, M - 1$ . The input stream  $\{f(n)\}$  is passed through the filter to produce the sequence  $\{y(n)\}$ . At each time-step the filter coefficients are updated using an error  $e(n) = d(n) - y(n)$  where  $d(n)$  is the desired response (usually based on  $\{f(n)\}$ ).

The LMS filter is implicitly designed around ensemble statistics, and uses a gradient descent method based on expected values of the waveform statistics to seek optimal values for the filter coefficients. On the other hand, the RLS filter computes the temporal statistics directly at each time-step to determine the optimal filter coefficients. The RLS filter is adaptive and can adjust to time varying input statistics. Under most conditions the RLS filter will converge faster than a LMS filter.

Refer to the class handout *Introduction to Recursive-Least-Squares (RLS) Adaptive Filters* for details.