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.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF MECHANICAL ENGINEERING
2.161 Signal Processing - Continuous and Discrete

Introduction to Least-Squares Adaptive Filters[1]

1 Introduction

In this handout we introduce the concepts of 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 Fig. 1. The adaptive filter is FIR of length M with coefficients b_k, $k = 0, 1, 2, \ldots, M-1$. 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

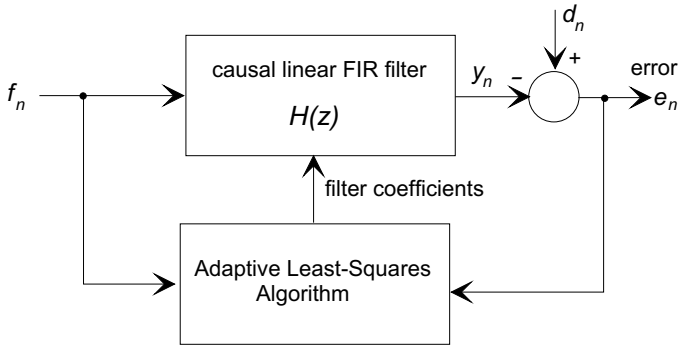

Figure 1: The adaptive least-squares filter structure.

handle a particular input. Because it is adaptive, it can adjust to a broadly defined task.

2 The Adaptive LMS Filter Algorithm

2.1 Simplified Derivation

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

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

$J(n)$ 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:

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

[1] D. Rowell December 9, 2008

where Λ(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.) From Eq. (1)

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

and the adaptive filter algorithm is

$$b_k(n+1) = b_k(n) + \Lambda e(n) f(n-k), \qquad k = 0, 1, 2, \ldots M-1 \tag{3}$$

or in matrix form

$$\mathbf{b}(n+1) = \mathbf{b}(n) + \Lambda e(n)\mathbf{f}(n), \tag{4}$$

where

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

is a column vector of the filter coefficients, and

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

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

Equation (3), or (4), defines the fixed-gain FIR adaptive Least-Mean-Square (LMS) filter algorithm. A Direct-Form implementation for a filter length $M = 5$ is shown in Fig. 2.

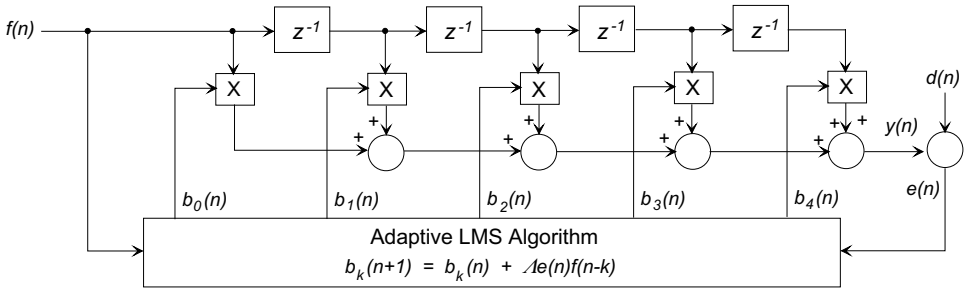

Figure 2: A Direct-Form LMS adaptive filter with length $M = 5$..

2.2 Expanded Derivation

For a filter as shown in Fig. 1, the mean-squared-error (MSE) is defined as

$$\begin{aligned} \text{MSE} &= E\left\{e^2(n)\right\} = E\left\{(d(n) - y(n))^2\right\} \\ &= E\left\{d^2(n)\right\} + E\left\{y^2(n)\right\} - 2E\left\{d(n)y(n)\right\} \\ &= \phi_{dd}(0) + \phi_{yy}(0) - 2\phi_{dy}(0) \end{aligned} \tag{5}$$

where $E\{\}$ is the expected value, $\phi_{dd}(k)$ and $\phi_{yy}(k)$ are autocorrelation functions and $\phi_{dy}(k)$ is the cross-correlation function between $\{d(n)\}$ and $\{y(n)\}$.

The filter output is

$$y(n) = \sum_{k=0}^{M-1} b_k f(n-k)$$

and for stationary waveforms, Eq. (5) at time step n reduces to

$$\text{MSE} = \phi_{dd}(0) + \sum_{m=0}^{M-1}\sum_{k=0}^{M-1} b_m(n)b_k(n)\phi_{ff}(m-k) - 2\sum_{k=0}^{N-1} b_k(n)\phi_{fd}(k) \tag{6}$$

2.2.1 The Optimal FIR Coefficients

The optimal FIR filter coefficients b_k^{opt}, $k = 0, \ldots, M-1$, that minimize the MSE, are found by setting the derivatives with respect to each of the $b_k(n)$'s equal to zero. From Eq. (6)

$$\frac{\partial\,(\text{MSE})}{\partial b_k(n)} = 2\sum_{m=0}^{N-1} b_m^{\text{opt}}\phi_{ff}(m-k) - 2\phi_{fd}(k) = 0, \qquad k = 0, 1, \ldots, N-1 \tag{7}$$

which is a set of linear equations in the optimal coefficients b_k^{opt}, and which in matrix form is written

$$\mathbf{R}\mathbf{b}^{\text{opt}} = \mathbf{P} \qquad \text{or} \qquad \mathbf{b}^{\text{opt}} = \mathbf{R}^{-1}\mathbf{P}, \tag{8}$$

where

$$\mathbf{R} = \begin{bmatrix} \phi_{ff}(0) & \phi_{ff}(1) & \phi_{ff}(2) & \cdots & \phi_{ff}(M-1) \\ \phi_{ff}(1) & \phi_{ff}(0) & \phi_{ff}(1) & \cdots & \phi_{ff}(M-2) \\ \phi_{ff}(2) & \phi_{ff}(1) & \phi_{ff}(0) & \cdots & \phi_{ff}(M-3) \\ \vdots & \vdots & \vdots & & \vdots \\ \phi_{ff}(M-1) & \phi_{ff}(M-2) & \phi_{ff}(M-3) & \cdots & \phi_{ff}(0) \end{bmatrix}$$

is a Toeplitz matrix, known as the *correlation matrix*, and

$$\mathbf{P} = \begin{bmatrix} \phi_{fd}(0) & \phi_{fd}(1) & \phi_{fd}(2) & \cdots & \phi_{fd}(M-1) \end{bmatrix}^T$$

is the *cross-correlation vector*.

With these definitions the MSE, as expressed in Eq. (6), reduces to

$$\text{MSE} = \phi_{dd}(0) + \mathbf{b}(n)^T\mathbf{R}\mathbf{b}(n) - 2\mathbf{P}^T\mathbf{b}(n), \tag{9}$$

and the minimum MSE is

$$\text{MSE}_{\text{min}} = \phi_{dd}(0) - \mathbf{P}^T\mathbf{b}^{\text{opt}} \tag{10}$$

2.2.2 The LMS Algorithm

Assume an update algorithm of the form

$$\mathbf{b}(n+1) = \mathbf{b}(n) + \frac{1}{2}\Lambda(n)\mathbf{S}(n) \tag{11}$$

where $\mathbf{S}(n)$ is a direction vector that will move $\mathbf{b}(n)$ toward $\mathbf{b}^{\text{opt}}$, and $\Lambda(n)$ is an empirically chosen gain schedule that determines the step-size at each iteration. In particular, with the method of steepest descent, let $\mathbf{S}(n) = -\mathbf{g}(n)$ where

$$\mathbf{g}(n) = \frac{d(\text{MSE})}{d\mathbf{b}}$$

is the gradient vector, and

$$g_k(n) = \frac{\partial \text{MSE}}{\partial b_k} = 2\sum_{m=0}^{M-1} b_k(n)\phi_{ff}(m-k) - 2\phi_{fd}(k), \qquad k = 0, 1, \ldots M-1.$$

from Eq. (6). Then, as above,

$$\mathbf{g}(n) = 2\left[\mathbf{R}\mathbf{b}(n) - \mathbf{P}\right], \tag{12}$$

and the LMS algorithm is

$$\begin{aligned} \mathbf{b}(n+1) &= \mathbf{b}(n) - \frac{1}{2}\Lambda(n)\mathbf{g}(n) && (13) \\ &= \left[\mathbf{I} - \Lambda(n)\mathbf{R}\right]\mathbf{b}(n) + \Lambda(n)\mathbf{P} && (14) \end{aligned}$$

where $\mathbf{I}$ is the identity matrix. It is interesting to note that if we define

$$\mathbf{\Delta b}(n) = \mathbf{b}^{\text{opt}} - \mathbf{b}(n)$$

the LMS algorithm becomes

$$\mathbf{b}(n+1) = \mathbf{b}(n) - \Lambda(n)\mathbf{R}\mathbf{\Delta b}(n) \tag{15}$$

and if any $b_k(n) = b_k^{\text{opt}}$, $b_k(n+1) = b_k^{\text{opt}}$ for all subsequent time steps.

In practice the LMS algorithm does not have $\mathbf{P}$ or $\mathbf{R}$ available, and we seek an estimator for $[\mathbf{R}\mathbf{b}(n) - \mathbf{P}]$. The error $e(n)$ is

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

and therefore

$$E\left\{e(n)f(n-j)\right\} = E\left\{d(n)f(n-j)\right\} - E\left\{\sum_{k=0}^{M-1} b(k)f(n-j)f(n-k)\right\}, \quad \text{for } j = 0, 1, 2, \ldots, M-1.$$

The individual equations may be collected into vector form

$$E\left\{e(n)\mathbf{f}(n)\right\} = \mathbf{P} - \mathbf{R}\mathbf{b}(n) \tag{16}$$

and using Eq. (12) the gradient vector can be written

$$\mathbf{g}(n) = -2E\left\{e(n)\mathbf{f}(n)\right\}. \tag{17}$$

An unbiased estimate $\widehat{\mathbf{g}}(n)$ of the gradient vector at the nth iteration is simply found from Eq, (17) as

$$\widehat{\mathbf{g}}(n) = -2e(n)\mathbf{f}(n), \tag{18}$$

and substituted into Eq, (11) to generate the LMS algorithm

$$\mathbf{b}(n+1) = \mathbf{b}(n) + \Lambda(n)e(n)\mathbf{f}(n), \tag{19}$$

which is identical to that obtained with the simplified derivation in Eq. (4).

2.3 Convergence Properties

A full discussion of the convergence properties of the LMS algorithm is beyond the scope of this handout The value of the gain constant Λ must be selected with some care. We simply state without proof that $\mathbf{b}(n)$ will converge to $\mathbf{b}^{\text{opt}}$ provided

$$0 < \Lambda < \frac{2}{\lambda_k}, \qquad \text{for } k = 0, 1, 2, \ldots, M-1$$

where λ_k is an eigenvalue of the matrix $\mathbf{R}$. Because $\mathbf{R}$ is an auto-correlation matrix, its eigenvalues are non-negative and

$$\lambda_{\max} < \sum_{k=0}^{M-1} \lambda_k = \text{trace}\mathbf{R} = M\phi_{ff}(0) = E\left\{f^2(n)\right\}$$

To ensure stability

$$\Lambda < \frac{2}{M.E\left\{f^2(n)\right\}} \tag{20}$$

If Λ is too large it will cause the system to overshoot the optimal values for the $\mathbf{b}(n)$ and the system will become unstable. The convergence rate is dependent on the ratio of the minimum to maximum eigenvalues of $\mathbf{R}$. If $\lambda_{\min}/\lambda_{\max} \approx 1$, the convergence will be fast, but conversely, if $\lambda_{\min}/\lambda_{\max} << 1$ the convergence will be sluggish, and the filter will not track rapidly changing conditions.

2.4 Variations on the Basic LMS Algorithm

2.4.1 Fixed Gain Schedule Implementation

The LMS algorithm is commonly used with a fixed gain schedule $\Lambda(n) = \Lambda$ for two reasons: first in order that the filter can respond to varying signal statistics at any time it is important that, $\Lambda(n)$ not be a direct function of n. If $\Lambda(n) \to 0$ as $n \to \infty$, adaptation could not occur. The second factor is that the fixed gain LMS algorithm is easy to implement in hardware and software.

2.4.2 The Normalized MLS Algorithm

Equation (20) demonstrates that the convergence and stability depend of the signal power. To normalize this dependence a modified form of the LMS algorithm, frequently used in practice is

$$\mathbf{b}(n+1) = \mathbf{b}(n) + \frac{\Lambda}{||f(n)||^2} e(n)\mathbf{f}(n), \tag{21}$$

which is essentially a variable gain method with

$$\Lambda(n) = \frac{\Lambda}{||f(n)||^2}$$

To avoid numerical problems when the norm of the signal vector is small, a small positive constant ϵ is often added

$$\Lambda(n) = \frac{\Lambda}{\epsilon + ||f(n)||^2}$$

These algorithms are known as *normalized* MLS, or NMLS, algorithms.

2.4.3 Smoothed MLS Algorithms

Because the update algorithm uses an estimator of the gradient vector the filter coefficients will be subject to random perturbations even when nominally converged to $\mathbf{b}^{\text{opt}}$. Several approaches have been used to smooth out these fluctuations:

(a) Use a running average over the last K estimates.
A FIR filter may be used to provide a smoothed estimate $\overline{\widehat{\mathbf{g}}}(n)$ of the gradient vector, $\widehat{\mathbf{g}}(n)$, for example a fixed length moving average

$$\overline{\widehat{\mathbf{g}}}(n) = \frac{1}{K} \sum_{k=0}^{K-1} \widehat{\mathbf{g}}(n-k). \tag{22}$$

The LMS algorithm (Eq. (13)) is then based on the averaged gradient vector eatimate:

$$\mathbf{b}(n+1) = \mathbf{b}(n) - \frac{1}{2}\Lambda(n)\overline{\widehat{\mathbf{g}}}(n)$$

(b) Update the coefficients every K steps, using the average.
A variation on the above is to update the coefficients every N time steps, and compute the average of the gradient vector estimates during the intermediate time steps.

$$\overline{\widehat{\mathbf{g}}}(Nn) = \frac{1}{K} \sum_{k=0}^{N-1} \widehat{\mathbf{g}}(nN+k). \tag{23}$$

The update equation, applied every N time-steps, is

$$\mathbf{b}((n+1)N) = \mathbf{b}(nN) - \frac{1}{2}\Lambda(Nn)\overline{\widehat{\mathbf{g}}}(Nn)$$

(c) Use a simple IIR filter to smooth the estimate of the gradient vector.

The noisy estimates of the gradient vector, Eq. (18), may be smoothed with a simple first-order, unity-gain, low-pass IIR filter with a transfer function

$$H_s(z) = \frac{1-\alpha}{1-\alpha z^{-1}}$$

to produce a smoothed estimate $\overline{\widehat{\mathbf{g}}}(n)$ of $\mathbf{g}(n)$. The difference equation is:

$$\overline{\widehat{\mathbf{g}}}(n) = \alpha\overline{\widehat{\mathbf{g}}}(n) + (1-\alpha)\widehat{\mathbf{g}}(n). \tag{24}$$

The value of α controls the degree of smoothing. If $\alpha = 0$ then $\overline{\widehat{\mathbf{g}}}(n) = \widehat{\mathbf{g}}(n)$ and there is no smoothing, but as $\alpha \to 1$ the contribution from the most recent estimate decreases and the smoothing increases. As above, the LMS algorithm (Eq. (13)) is then

$$\mathbf{b}(n+1) = \mathbf{b}(n) - \frac{1}{2}\Lambda(n)\overline{\widehat{\mathbf{g}}}(n)$$

2.4.4 Implementations Based on the Sign of the Error

For hardware implementations, where multiplication operations are computationally expensive, it may be possible to use steps that are related to the sign of all or part of the gradient vector estimate, for example three possibilities are:

$$\mathbf{b}(n+1) = \mathbf{b}(n) + \Lambda\text{sgn}(e(n))\mathbf{f}(n) \tag{25}$$

$$\mathbf{b}(n+1) = \mathbf{b}(n) + \Lambda e(n)\text{sgn}(\mathbf{f}(n)) \tag{26}$$

$$\mathbf{b}(n+1) = \mathbf{b}(n) + \Lambda\text{sgn}(e(n))\text{sgn}(\mathbf{f}(n)) \tag{27}$$

where sgn() is the signum function. In the last case numerical multiplication can be eliminated completely. Care must be taken to ensure stability and convergence with such reduced complexity implementations.

3 A MATLAB Demonstration Adaptive Least-Squares Filter

```
% --------------------------------------------------------------------------
% 2.161 Classroom Example - LSadapt - Adaptive Lleast-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
```

4 Application - Suppression of Narrow-band Interference in a Wide-band Signal

In this section we 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 in Fig. 3. This is similar to Fig. 1, with the addition of a delay block of Δ 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 $e(n) \approx s(n)$.

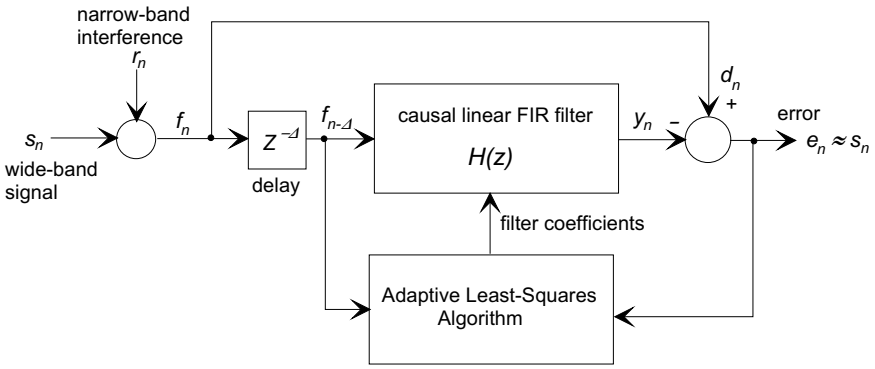

Figure 3: The adaptive least-squares filter structure for narrow-band noise suppression.

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 Δ-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 $H(z)$:

$$\begin{aligned} H(z) &= \frac{E(z)}{F(z)} = \frac{F(z) - z^{-\Delta} H_n(z) F(z)}{F(z)} \\ &= 1 - z^{-\Delta} H_n(z) \\ &= z^0 + 0z^{-1} + \ldots + 0z^{-(\Delta-1)} - b_0(n)z^{-\Delta} - b_1(n)z^{-(\Delta+1)} + \ldots \\ &\qquad \ldots - b_{M-1}(n)z^{-(\Delta+M-1)} \end{aligned} \tag{28}$$

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

$$h'(k) = \begin{cases} 1 & k = 0 \\ 0 & 1 \le k < \Delta \\ -b_{k-\Delta}(n) & \Delta \le k \le M + \Delta - 1 \end{cases} \tag{29}$$

and with frequency response

$$H(e^j\Omega) = \sum_{k=0}^{M+\Delta-1} h'(k) e^{-jk\Omega}. \tag{30}$$

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

$$\mathbf{b}(n+1) = \mathbf{b}(n) + \Lambda e(n) \mathbf{f}(n-\Delta)) \tag{31}$$

or

$$b_k(n+1) = b_k(n) + \Lambda e(n) f((n-\Delta) - k), \qquad k = 0, 1, 2, \ldots M-1. \tag{32}$$

4.1 Example 1: Demonstration of Convergence with a Sinusoidal Input

In the handout *MATLAB Examples of Least-Squares FIR Filter Design*, example *A One-step Linear Predictor for a Sinusoidal Input* we examined the static least-squares filter design for the case

described by Stearns and Hush for the one-step predictor of a sine wave with lengths $M = 2$ and $M = 3$. In this first example, we note the similarity of the adaptive filter to the one-step predictor and examine the convergence of the filter to the closed-form filter coefficients. The input is a noise-free sinusoid

$$f(n) = \sin\left(\frac{2\pi n}{12}\right).$$

The stability of the algorithm is governed by Eq. (20), and since for a sinusoid $E\left\{f(n)^2\right\} = 0.5$ we are constrained to

$$\Lambda < \frac{2}{M.E\left\{f(n)^2\right\}} \quad < \quad \begin{cases} 2 & \text{for } M = 2 \\ 4/3 & \text{for } M = 3 \end{cases}$$

The following MATLAB script was used:

```
% Example - Demonstration of convergence with a sinusoidal input
for J=1:1000
   f(J) = sin(2*pi*J/12);
end
% Initialize the filter with M = 2, Delta =1
% Choose filter gain parameter Lambda = 0.1
Delta = 1;
Lambda = 0.1;
M = 2;
x = LSadapt('initial',Lambda,M);
% Filter the data
y = zeros(1,length(f));
e = zeros(1,length(f));
f_delay = zeros(1,Delta+1);
% Filter - implement the delay
for J = 1:length(f)
    for K = Delta+1:-1:2
        f_delay(K) = f_delay(K-1);
    end
    f_delay(1) = f(J);
%   The desired output is f(J), the filter input is the delayed signal.
    [y(J),b_filter] = LSadapt(f_delay(Delta+1),f(J));
end;
```

The script was modified and run with $M = 2$ and 3, and with various values of Λ. The convergence of the error is demonstrated in Fig. 4. The dependence of the convergence upon M and Λ is clearly demonstrated.

The values reported for the filter coefficients with $M = 2$ were

$$b(0) = 1.73117, \qquad b(1) = -1$$

which are in agreement with the solution $b(0) = \sqrt{3}$, and $b_1 = -1$. For $M = 3$ the values returned were

$$b(0) = 1.24290, \qquad b(1) = -0.15275, \qquad b(2) = -0.48916.$$

As Stearns and Hush note, there is no unique solution for the coefficients for $M = 3$, but the optimal filter must satisfy the conditions:

$$b(0) - b(1) = \sqrt{3}, \qquad b(0) + \sqrt{3}b(1) + 2b(2) = 0$$

The reported values satisfy these constraints, indicating that the filter has found an optimal solution.

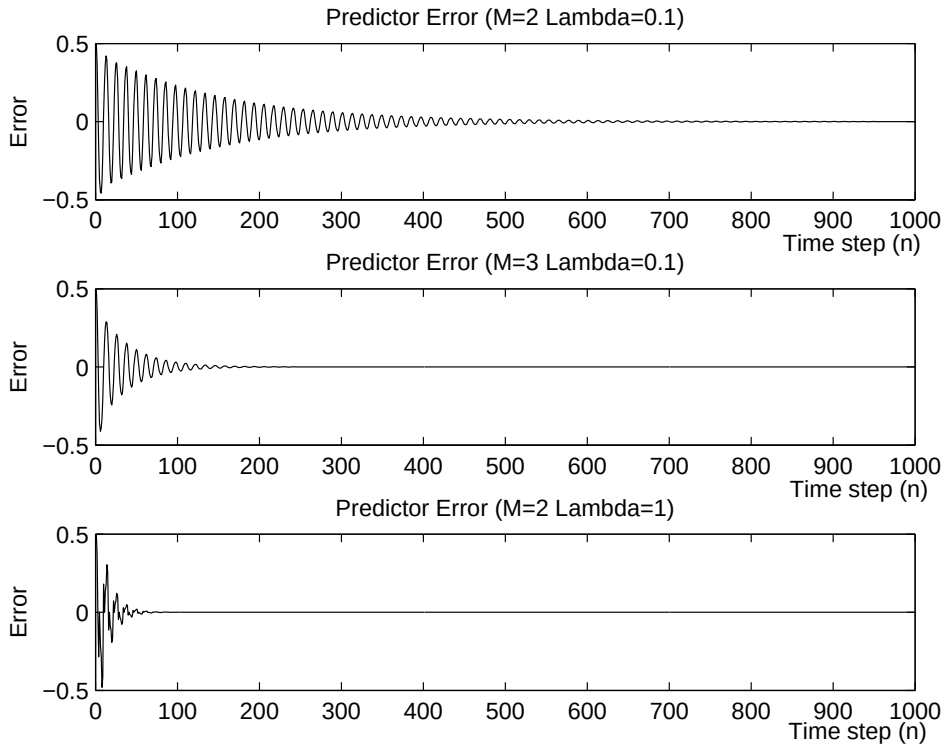

Figure 4: Convergence of predictor error $e(n)$ with filter length M and gain Λ.

4.2 Example 2: Suppression of a Sinusoid in Noise

For the second example we look at the rejection of a sinusoid of unknown frequency in white noise. This case is extreme in that the signal $\{s(n)\}$ has an auto-correlation function $\phi_{ss}(n) = \delta(n)$, while the interference has a periodic auto-correlation.

The following MATLAB script demonstrates the efficacy of the method.

```
% Create the input as white noise with a strong sinusoidal component
f = randn(1,10000);
L = length(f);
y = zeros(1,length(f));
e = zeros(1,length(f));
Lambda = .001; Delta = 1; M=15;
x = LSadapt('initial', Lambda, M);
f_delay = zeros(1,Delta+1);
for J = 1:L
    f(J) = f(J) + 3*sin(2*pi*J/12);
    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;
w=-pi:2*pi/1000:pi;
```

```
subplot(1,2,1), plot(w, fftshift(abs(fft(f(L-1000:L)))));
xlabel('Normalized frequency')
ylabel('Magnitude')
title('Input spectrum')
subplot(1,2,2), plot(w, fftshift(abs(fft(e(L-1000:L)))));
xlabel('Normalized frequency')
ylabel('Magnitude')
title('Output spectrum')
```

Figure 5 shows the input and output spectra of the last 1000 samples of the data record.

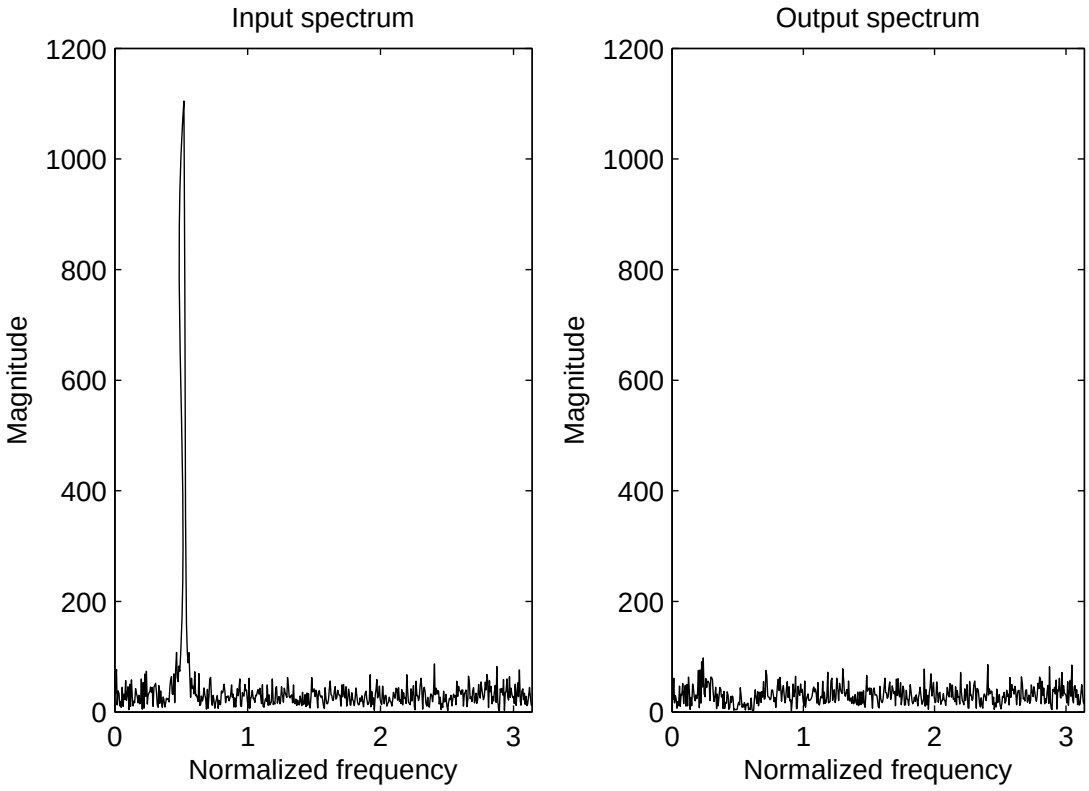

Figure 5: Input and output spectra for the filter in Example 2.

4.3 Example 3: Frequency Domain Characteristics of an LMS Suppression Filter

This example demonstrates the filter characteristics after convergence. The interfering signal is comprised of 100 sinusoids with random phase and random frequencies Ω between 0.3 and 0.6. The "signal" is white noise. The filter used has $M = 31$, $\Delta = 1$, and Λ was adjusted to give a reasonable convergence rate. The overall system $H(z) = 1 - z^{-\Delta}H_n(z)$ frequency response magnitude, Eq. (30), 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 0.3pi < Omega < 0.6pi 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_{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 input and output spectra are shown in Fig. 6, and the filter frequency response magnitude is shown in Fig. 7. The adaptive algorithm has clearly generated a notch-filter covering the bandwidth of the interference. The pole-zero plot in Fig. 8 shows how the zeros have been placed over the spectral region ($0.3 < \Omega < 0.6$) to create the band-reject characteristic..

4.4 Example 4: 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

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

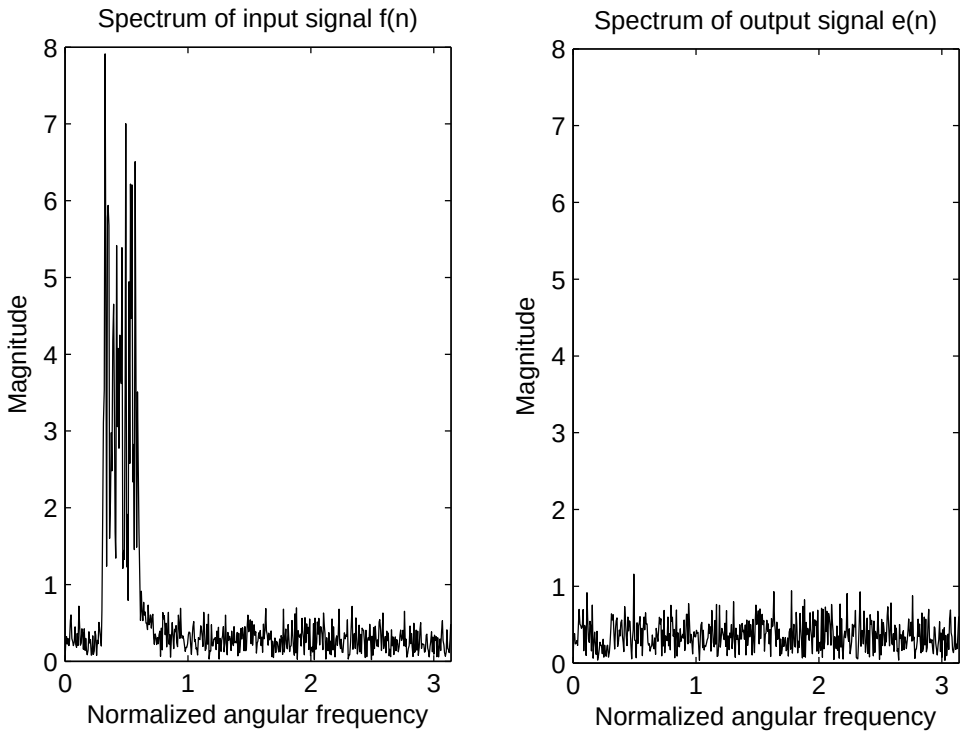

Figure 6: Input and output spectra from an adaptive suppression filter with interference in the band $0.3 < \Omega < 0.6$.

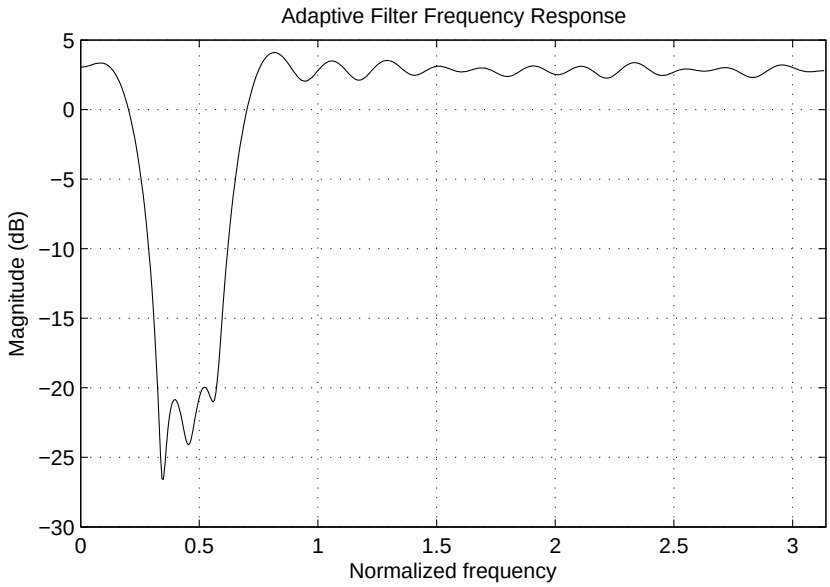

Figure 7: Frequency response magnitude of an adaptive suppression filter with interference in the band $0.3 < \Omega < 0.6$.

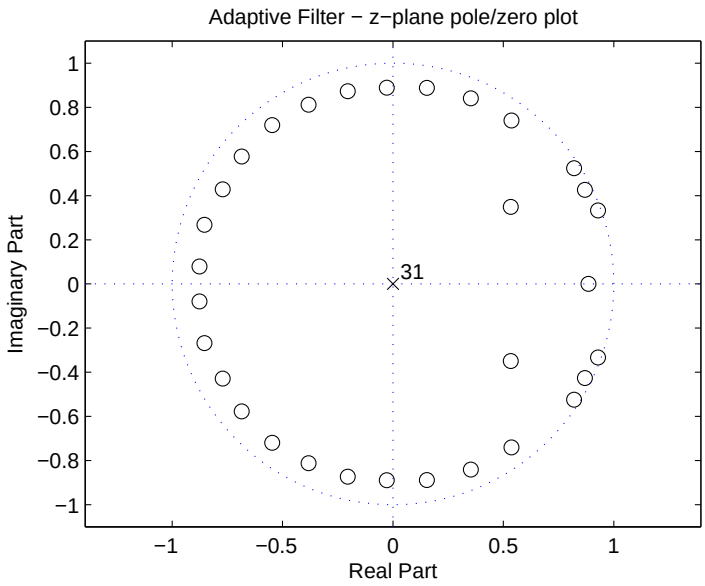

Figure 8: Pole/zero plot of an adaptive suppression filter with interference in the band $0.3 < \Omega < 0.6$.

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.

```
% Example 2: Suppression of a frequeny 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 = Delta+1:-1:2
          f_delay(K) = f_delay(K-1);
      end
      f_delay(1) = f(J);
      [y(J),b1] = LSadapt(f_delay(Delta+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.

Figs. 9 and 10 show the waveform spectra before and after filtering. Figure 9 clearly shows the superposition of the speech spectrum on the pedestal spectrum of the swept sinusoid. The pedestal has clearly been removed in Fig. 10. Figure 11 shows 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.

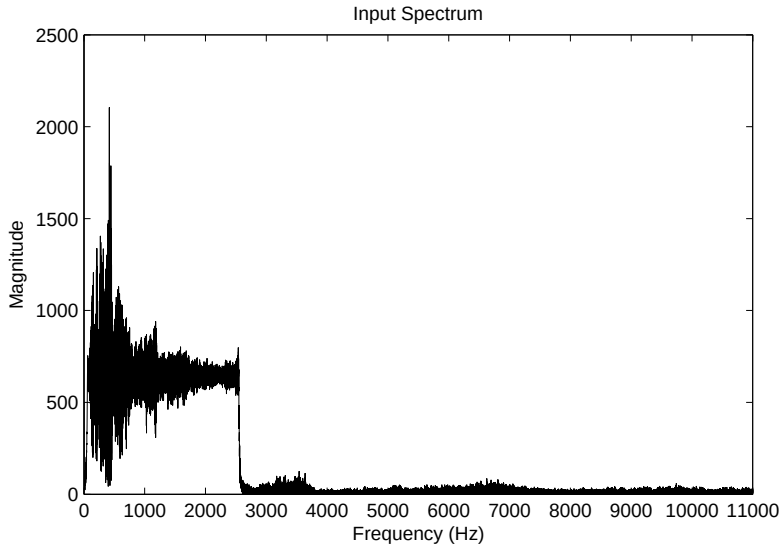

Figure 9: Example 4: Input spectrum of a sliding sinusoid superimposed on a speech waveform.

5 Application - 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 Fig. 12. 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

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

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.

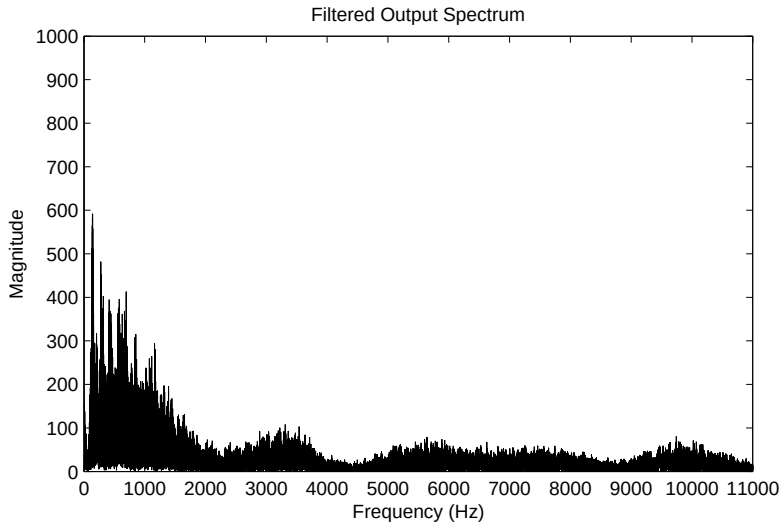

Figure 10: Example 4: Output spectrum from the adaptive least-squares filtering to enhance the speech waveform.

5.1 Example

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

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

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;
```

Figure 13 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

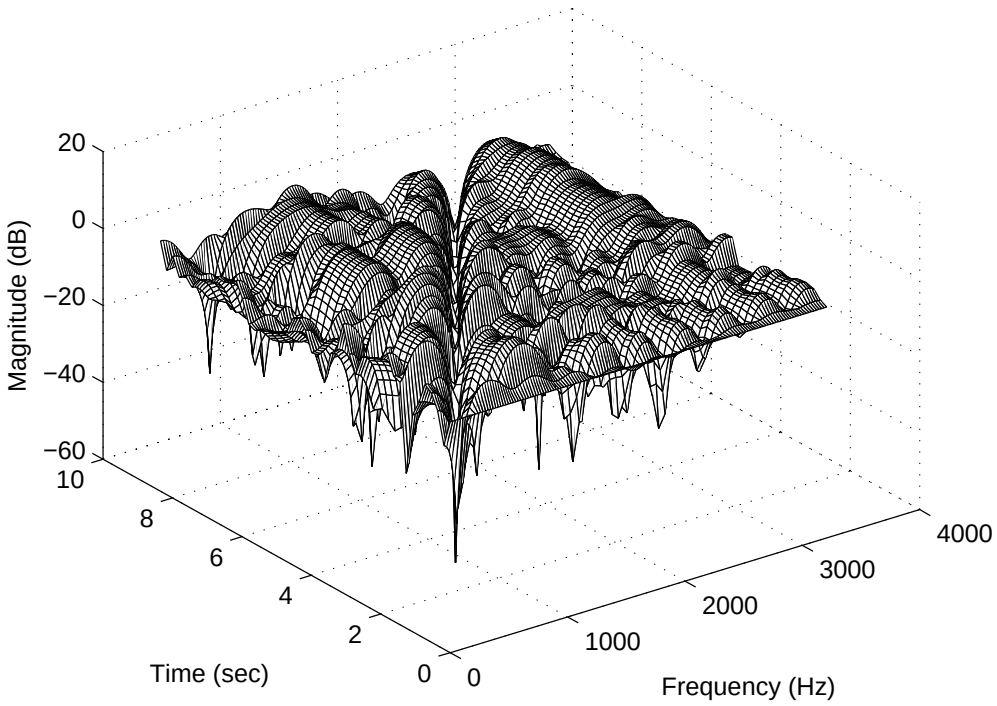

Figure 11: Meshed surface showing the time dependence the frequency response magnitude. The motion of the filter notch is seen as the valley in the response as the interference signal's frequency changes.

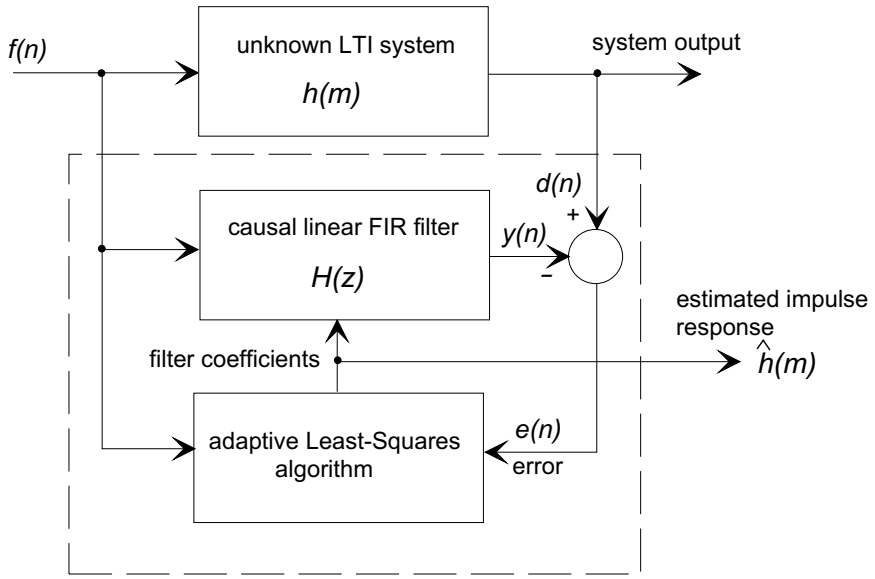

Figure 12: Adaptive filtering structure for system identification.

the changing system dynamics.

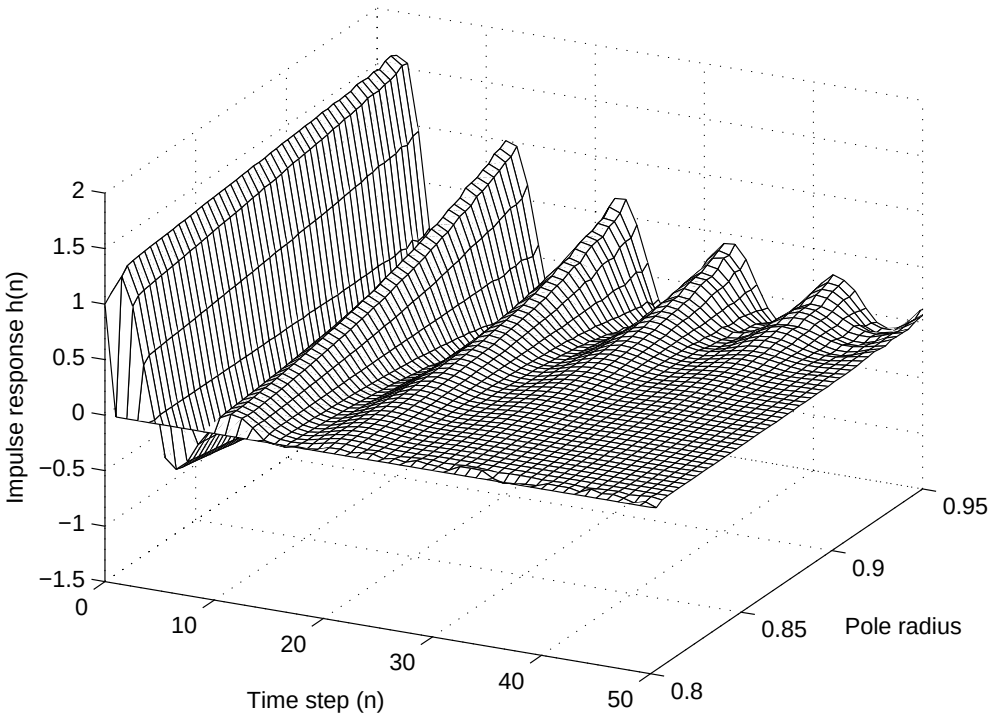

Figure 13: Estimated impulse response of a second-order system with dynamically changing poles using an adaptive LMS filter (length 51) with white noise as the input.