2.161 Signal Processing: Continuous and Discrete
Fall 2008

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

Introduction to Recursive-Least-Squares (RLS) Adaptive Filters[1]

1 Introduction

In this handout we introduce the concepts of adaptive recursive-least-squares (RLS) FIR filters, where the coefficients are continually adjusted on a step-by-step basis during the filtering operation. The filter structure shown in Fig. 1 is similar to the least-mean-squares (LMS) filter described in the handout *Introduction to Least-Squares Adaptive Filters*, but differs in the internal algorithmic structure.

Like the LMS filter, the RLS filter is FIR of length M with coefficients b_k, $k = 0, 1, 2, \ldots, M-1$. 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 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.

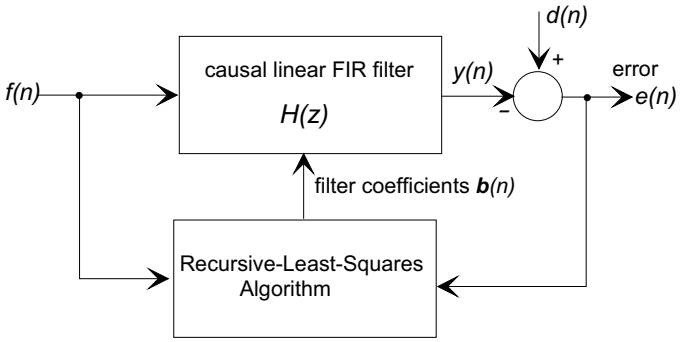

Figure 1: The recursive-least-squares (RLS) filter structure.

2 The Recursive-Least-Squares Filter Algorithm

For a filter as shown in Fig. 1, the total-squared-error $\mathcal{E}(n)$ at the nth iteration is defined as

$$\mathcal{E}(n) \;=\; \sum_{i=0}^{n} e^2(i) = \sum_{i=0}^{n} \left(d(i) - y(i)\right)^2 \tag{1}$$

We modify the standard least-squares approach by including an exponential "forgetting factor" λ^{n-i}, $(0 < \lambda \leq 1)$ to each error term, and modify Eq. 1 as follows

$$\mathcal{E}'(n) \;=\; \sum_{i=0}^{n} \lambda^{n-i} e^2(i) = \sum_{i=0}^{n} \lambda^{n-i} \left(d(i) - y(i)\right)^2$$

[1] D. Rowell December 9, 2008

$$= \sum_{i=0}^{n} \left(d'(i) - y'(i)\right)^2 \tag{2}$$

where $d'(i) = \sqrt{\lambda^{n-i}}d(i)$, and $y'(i) = \sqrt{\lambda^{n-i}}y(i)$. The purpose of the factor λ is to weight recent data points more heavily, and thus allow the filter to track changing statistics in the input data.

The FIR filter output is given by the convolution sum

$$y(n) = \sum_{k=0}^{M-1} b_k f(n-k) \tag{3}$$

and for stationary waveforms, Eq. (1) at time step n reduces to

$$\begin{aligned}
\mathcal{E}'(n) &= \sum_{i=0}^{n} d'^2(i) + \sum_{i=0}^{n} y'^2(i) - 2\sum_{i=0}^{n} d'(i)y'(i) \\
&= \sum_{i=0}^{n} \lambda^{n-i} d^2(i) + \sum_{i=0}^{n} \lambda^{n-i} \sum_{k=0}^{M-1}\sum_{m=0}^{M-1} b_k b_m f(n-k)f(n-m) \\
&\qquad -2\sum_{i=0}^{n} \lambda^{n-i} \sum_{k=0}^{M-1} b_k f(n-k)d(i)
\end{aligned} \tag{4}$$

2.1 The RLS FIR Filter Coefficients

As we have seen previously, the optimal FIR filter coefficients $b_k(n)$, $k = 0, \ldots, M-1$, that minimize $\mathcal{E}'(n)$, are found by setting the derivatives with respect to each of the filter coefficients $b_k(n)$ equal to zer, that is

$$\frac{\partial\left(\mathcal{E}'\right)}{\partial b_k(n)} = 0, \qquad k = 0, 1, \ldots, M-1, \tag{5}$$

which generates a set of linear equations in the optimal coefficients $b_k(n)$. Using Eq. (4), in matrix form these equations may be written

$$\mathbf{R}(n)\mathbf{b}(n) = \mathbf{P}(n) \tag{6}$$

or

$$\mathbf{b}(n) = \mathbf{R}^{-1}(n)\mathbf{P}(n), \tag{7}$$

where

$$\mathbf{b}(n) = \begin{bmatrix} b_0(n) & b_1(n) & b_2(n) & \ldots & b_{M-1}(n) \end{bmatrix}^T$$

is a $M \times 1$ column vector of the filter coefficients,

$$\mathbf{R}(n) = \sum_{i=0}^{n} \lambda^{n-i}\mathbf{f}(i)\mathbf{f}^T(i), \tag{8}$$

is a $M \times M$ matrix, and

$$\mathbf{f}(i) = \begin{bmatrix} f(i) & f(i-1) & f(i-2) & \ldots & f(i-(M-1)) \end{bmatrix}^T$$

is a column vector of the recent input history, and

$$\mathbf{P}(n) = \sum_{i=0}^{n} \lambda^{n-i}\mathbf{f}(i)d(i) \tag{9}$$

is a $M \times 1$ column vector.

Notice the similarity between these definitions of $\mathbf{R}(n)$ and $\mathbf{P}(n)$ in Eqs. (8) and (9) to the correlation matrix and cross-correlation vector in least-mean-squares (LMS) filter design. In this case however, the use of the weighting factor λ destroys the Toeplitz nature of the matrix $\mathbf{R}(n)$, and efficient numerical inversion methods are not available.

The RLS design algorithm does not attempt to solve Eq. (7) at each time-step, which would be impractical, requiring repeated inversion of $\mathbf{R}(n)$. Instead, the method uses an iterative algebraic procedure to find the updated inverse of $\mathbf{R}(n)$ using the result from the previous step. Equation (8) shows that $\mathbf{R}(n)$ may be computed recursively:

$$\mathbf{R}(n) = \lambda \mathbf{R}(n-1) + \mathbf{f}(n)\mathbf{f}^T(n). \tag{10}$$

However, we need a recursive definition of $\mathbf{R}^{-1}(n)$ based on $\mathbf{R}^{-1}(n-1)$ to compute the filter coefficients in Eq, (7).

The *matrix inversion lemma* (see the Appendix) states that if a square non-singular $n \times n$ matrix $\mathbf{A}$, with known inverse $\mathbf{A}^{-1}$, is updated with an additive term the new inverse is given by

$$\left(\mathbf{A} + \mathbf{BCB}^T\right)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{C}^{-1} + \mathbf{B}^T\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{B}^T\mathbf{A}^{-1} \tag{11}$$

where $\mathbf{B}$ is $n \times k$, and $\mathbf{C}$ is $k \times k$ and non-singular.

From Eq. (10), if we let $\mathbf{A} = \mathbf{R}^{-1}(n-1)$, $\mathbf{B} = \mathbf{f}(n)$, and $\mathbf{C} = \lambda^{-1}$, Eq. (11) gives

$$\begin{aligned}
\mathbf{R}^{-1}(n) &= \lambda^{-1}\left[\mathbf{R}(n-1) + \mathbf{f}(n)\lambda^{-1}\mathbf{f}^T(n)\right]^{-1} \\
&= \lambda^{-1}\left[\mathbf{R}^{-1}(n-1) - \frac{\mathbf{R}^{-1}(n-1)\mathbf{f}(n)\mathbf{f}^T(n)\mathbf{R}^{-1}(n-1)}{\lambda + \mathbf{f}^T(n)\mathbf{R}^{-1}(n-1)\mathbf{f}(n)}\right],
\end{aligned} \tag{12}$$

which is an algebraic recursion relationship that allows computation of $\mathbf{R}^{-1}(n)$ from the result of the previous time-step and the current input history vector $\mathbf{f}(n)$. Note that no matrix inversion is required.

In addition, from Eq. (9) $\mathbf{P}(n)$ may be defined recursively

$$\mathbf{P}(n) = \lambda \mathbf{P}(n-1) + d(n)\mathbf{f}(n) \tag{13}$$

so that Eqs.(12) and (13), when combined with Eq. (7), form the basis for the RLS filter design.

If we define an M-dimensional vector of *Kalman gains*

$$\mathbf{k}(n) = \frac{\mathbf{R}^{-1}(n-1)\mathbf{f}(n)}{\lambda + \mathbf{f}^T(n)\mathbf{R}^{-1}(n-1)\mathbf{f}(n)} \tag{14}$$

we can rewrite Eq. (12) as

$$\mathbf{R}^{-1}(n) = \lambda^{-1}\left[\mathbf{R}^{-1}(n-1) - \mathbf{k}(n)\mathbf{f}^T(n)\mathbf{R}^{-1}(n-1)\right]. \tag{15}$$

The filter update equation, Eq. (7), may then be written as a recursion

$$\begin{aligned}
\mathbf{b}(n) &= \mathbf{R}^{-1}(n)\mathbf{P}(n) \\
&= \lambda^{-1}\left[\mathbf{R}^{-1}(n-1) - \mathbf{k}(n)\mathbf{f}^T(n)\mathbf{R}^{-1}(n-1)\right]\left[\lambda\mathbf{P}(n-1) + d(n)\mathbf{f}(n)\right] \\
&= \mathbf{R}^{-1}(n-1)\mathbf{P}(n-1) + \lambda^{-1}d(n)\mathbf{R}^{-1}(n-1)\mathbf{f}(n) \\
&\quad -\mathbf{k}(n)\mathbf{f}^T(n)\mathbf{R}^{-1}(n-1)\mathbf{P}(n-1) \\
&\quad -\lambda^{-1}d(n)\mathbf{k}(n)\mathbf{f}^T(n)\mathbf{R}^{-1}(n-1)\mathbf{f}(n) \\
&= \mathbf{b}(n-1) + \mathbf{k}(n)\left[d(n) - \mathbf{f}^T(n)\mathbf{b}(n-1)\right].
\end{aligned} \tag{16}$$

We also note that $y(n) = \mathbf{f}^T(n)\mathbf{b}(n-1)$ is the convolution sum generating the filter output $y(n)$ using the previous set of filter coefficients, and therefore

$$\begin{aligned} \mathbf{b}(n) &= \mathbf{b}(n-1) + \mathbf{k}(n)\left(d(n) - y(n)\right) \\ &= \mathbf{b}(n-1) + \mathbf{k}(n)e(n) \end{aligned} \tag{17}$$

are the recursive filter update equations in terms of the Kalman gains and the filter output error.

2.2 Summary of the RLS Filter Coefficient Algorithm

With a new input sample $f(n)$, and desired output value $d(n)$,

1. Update the input history vector $\mathbf{f}(n)$.
2. Compute the filter output using the previous set of filter coefficients $\mathbf{b}(n-1)$
$$y(n) = \mathbf{f}^T(n)\mathbf{b}(n-1)$$
3. Compute the error
$$e(n) = d(n) - y(n)$$
4. Compute the Kalman gain vector
$$\mathbf{k}(n) = \frac{\mathbf{R}^{-1}(n-1)\mathbf{f}(n)}{\lambda + \mathbf{f}^T(n)\mathbf{R}^{-1}(n-1)\mathbf{f}(n)}$$
5. Update the matrix $\mathbf{R}^{-1}(n)$ for the next iteration
$$\mathbf{R}^{-1}(n) = \lambda^{-1}\left[\mathbf{R}^{-1}(n-1) - \mathbf{k}(n)\mathbf{f}^T(n)\mathbf{R}^{-1}(n-1)\right]$$
6. Update the filter coefficients for the next iteration.
$$\mathbf{b}(n) = \mathbf{b}(n-1) + \mathbf{k}(n)e(n)$$

2.3 Filter Initialization

2.3.1 Initial Filter Coefficients $\mathbf{b}(0)$

If *a-priori* knowledge of suitable coefficient values is available, it may be used to define the initial coefficients $\mathbf{b}(0)$, otherwise it is usual to define the initial filter coefficients as $\mathbf{b}(0) = \mathbf{0}$.

2.3.2 Initial Input Data History $\mathbf{f}(0)$

In the absence of prior knowledge of the input history, the *pre-windowing* convention sets the input $f(n) = 0$ for $n \leq 0$.

2.3.3 The Matrix $\mathbf{R}^{-1}(0)$

We note from Eq. (8) that

$$\mathbf{R}(n) = \sum_{i=0}^{n} \lambda^{n-i}\mathbf{f}(i)\mathbf{f}^T(i),$$

and therefore true initialization of $\mathbf{R}^{-1}(0)$ requires knowledge of the the input history for $n < 0$. Further, for small values of n, when the statistics are unreliable, there is a risk that $\mathbf{R}(n)$ may become singular. The usual convention is to define

$$\mathbf{R}^{-1}(0) = \delta \mathbf{I}$$

where δ is a positive scalar, to ensure that $\mathbf{R}(n)$ is well behaved for small n. As n increases the effect of this initialization error will decrease because of the effect of the weighting factor λ. It may be necessary to determine a suitable value of δ by experimentation, balancing stability with convergence rate, but a guideline that is often given is that

$$\delta > 100\sigma_f^2$$

where σ_f^2 is the variance of the input.

2.3.4 The Weighting Factor λ

The value of λ defines the system memory, and affects (1) the convergence and the ability of the filter to track time-varying statistics in the input sequence, and conversely (2) the stability of filter coefficients.

The effective memory-length (time-constant) N, the number of iterations before the effect of a disturbance has decayed to e^{-1} of its initial value, is

$$N = \frac{\sum_{k=0}^{\infty} k\lambda^k}{\sum_{k=0}^{\infty} \lambda^k} = \frac{\lambda}{1-\lambda}$$

Common values used are between $0.95 < \lambda < 0.995$, giving $19 < N < 199$.

2.4 A Potential Problem

Unlike the LMS filter algorithm, which does not use the auto-correlation matrix directly, the RLS algorithm requires that $\mathbf{R}(n)$ be non-singular. With deterministic input signals, if the filter length M is greater than that required to reduce the error to zero, $\mathbf{R}(n)$ will become singular. For example, when used as a one-step linear predictor with a sinusoidal input sequence, a filter length $M = 2$ is sufficient to predict the input. An RLS filter with $M = 2$ is well-behaved, but a filter with $M = 3$ fails because there is no unique solution for the filter coefficients $\mathbf{b}(n)$, and $\mathbf{R}(n)$ is singular. Even though $\mathbf{R}(n)$ is not inverted explicitly within the algorithm, the recursive estimation of $\mathbf{R}^{-1}(n)$ also fails.

This is explored further in Example 1 below.

3 A MATLAB Demonstration Recursive-Least-Squares Filter

```
% --------------------------------------------------------------------------
% 2.161 Classroom Example - RLSFilt - Demonstration Recursive Least-Squares FIR filter
%                           demonstration
% Usage :   1) Initialization:
%            y = RLSFilt('initial', lambda, M, delta)
%             where Lambda is the convergence rate parameter.
%                   lambda is the "fprgetting" exponential weight factor
%                   M is the filter length
%                  delta are the initial diagonal R^{-1}(n) matrix elements.
%             Example:
```

```
%                  [y, e] = adaptfir('initial', .95, 51, 0.01);
%              Note: RLSFilt returns y = 0 for initialization
%           2) Filtering:
%            [y, b] = RLSFilt(f, d};
%             where f is a single input value,
%                   d is the desired value, and
%                   y is the computed output value,
%                   b is the coefficient vector.
%
% Version:  1.0
% Author:   D. Rowell   12/9/07
% ------------------------------------------------------------------------
%
function [y, Bout] = RLSFilt(f, d, FIR_M, delta_n )
persistent F B lambda delta  M Rinv
% The following is initialization, and is executed once
if (ischar(f) && strcmp(f,'initial'))
    lambda = d;
    M = FIR_M;
    delta = delta_n;
    F = zeros(M,1);
    Rinv = delta*eye(M);
    B = zeros(M,1);
    y = 0;
else
% Filtering:
    for J = M:-1:2
       F(J) = F(J-1);
   end;
    F(1) = f;
% Perform the convolution
    y= F'*B;
    error = d - y;
% Kalman gains
    K = Rinv*F/(lambda + F'*Rinv*F);
% Update Rinv
    Rinvn = (Rinv - K*F'*Rinv)/lambda;
% Update the filter coefficients
   B = B + K*error;
   Bout=B;
   Rinv = Rinvn;
end
```

4 Examples

4.1 Example 1: Demonstration of Convergence with a Sinusoidal Input

In the handout *MATLAB Examples of Least-Squares FIR Filter Design*, example we examined a static least-squares filter design for a case described by Stearns and Hush: a one-step predictor of a sine wave with with filter lengths $M = 2$ and $M = 3$. We also examined this example in the

handout *Introduction to Least-Squares Adaptive Filters*. The input is a noise-free sinusoid

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

The one-step linear predictor structure is shown in Fig. 2. The following MATLAB script was used to examine the ability of the RLS algorithm to act as a one step predictor:

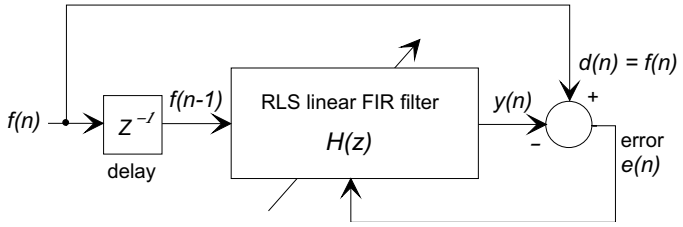

Figure 2: The one-step linear predictor.

```
% Example - Demonstration of convergence with a sinusoidal input
% See Stearns & Hush p. 246
L = 5000;
f=zeros(1,L);
for J=1:L
f(J) = sin(2*pi*J/12);
end
% Optionally Add a small random component to allow a non-singular R(n)
%f = f +0.001*randn(1,length(f));
% Initialize the filter with M = 2, Delta =1
% Choose filter gain parameter Lambda = 0.1
delta = .01; lambda = .99; M = 2;
x = RLSFilt('initial',lambda,M, delta);
% Filter the data
Delta = 1;
f_delay = zeros(1,Delta+1);
% Filter
for J = 1:length(f)
    for K = Delta+1:-1:2
        f_delay(K) = f_delay(K-1);
    end
    f_delay(1) = f(J);
    d = f_delay(Delta+1);
    [y,b] = RLSFilt(f_delay(Delta+1),f(J));
end;
% Report the final filter coefficients
b
```

The script was modified and run with $M = 2$ and 3, and with various values of λ.

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 $b_1 = -1$ as reported by Stearns and Hush.

For $M = 3$ the RLS algorithm failed and no sensible values were returned. As Stearns and Hush note, there is no unique solution for the coefficients for $M = 3$, and the optimal filter coefficients must satisfy the conditions:

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

Under these conditions the matrix $\mathbf{R}(n)$ is singular. However, when a small random component was added to the sinusoidal input sequence (see the script), the matrix $\mathbf{R}(n)$ remains invertible, and the values returned were

$$b(0) = 1.1232, \qquad b(1) = 0.0545, \qquad b(2) = -0.6088$$

which satisfy the above conditions. The LMS filter algorithm does not suffer from this problem.

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. The filter structure is as shown in Fig. 3. A delay of Δ time steps is used to remove the correlation between $s(n - \Delta)$ and $d(n)$, so that only the narrow-band interference $r(n)$ is predicted by the filter, and $y(n) \approx r(n)$. The filter output is taken as the error signal, $e(n) = d(n) - y(n) \approx s(n)$. The following MATLAB script demonstrates the efficacy of the method.

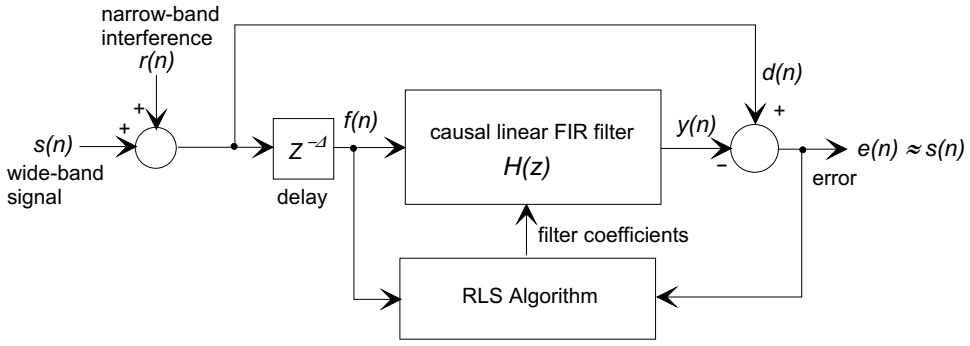

Figure 3: Narrow band interference rejection filter.

```
% Create the input as white noise with a strong sinusoidal component
f = 0.1*randn(1,10000);
y = zeros(1,length(f));
e = zeros(1,length(f));
M=15;  delta = .001; lambda = .95;
x = RLSFilt('initial',lambda, M, delta);
L = length(f);
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] = RLSFilt(f_delay(Delta+1),f(J));
    e(J) = f(J) - y(J);
end;
```

Figure 4 shows the input and output spectra of the last 1000 samples of the data record.

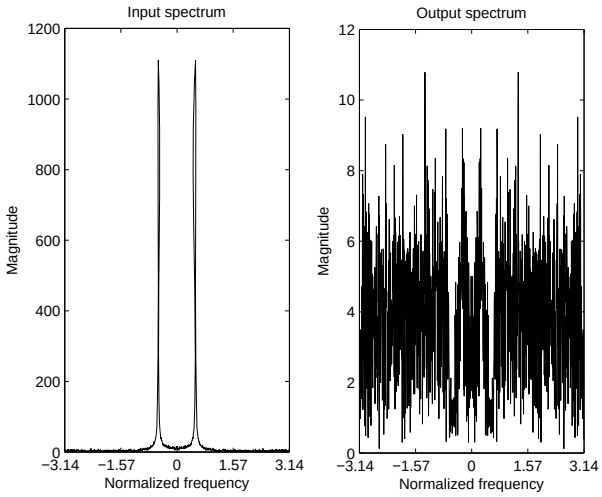

Figure 4: Input and output spectra for the filter in Example 2.

4.3 Example 3: Frequency Domain Characteristics of an RLS Narrow-Band Suppression Filter

This example is similar to Example 2. It uses the narrow-band interference suppression structure shown in Fig. 3. The interfering signal is comprised of 100 sinusoids with random phase and random normalized frequencies $0.3 < \Omega < 0.6$. The "signal" is white noise. The filter used has $M = 31$, and is initialized with $\delta = 1$. Because the system input is stationary, the weighting factor is set to $\lambda = 1$. The decorrelation delay $\Delta = 1$. The following MATLAB script runs a simulation with 10,000 steps and determines the overall frequency response magnitude of the system $H(z) = E(z)/D(z) = 1 - z^{-\Delta}H_n(z)$, and plots the system pole-zero plot.

```
% 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(1,10000);
 for J=1:10000
     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(1,10000);
 f      = .005*f + 0.01*signal;
% Initialize the filter
 lambda = 1;  M = 31; delta = 1;
 x = RLSFilt('initial',lambda, M, delta);
% Filter the data
 Delta = 1;
 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] = RLSFilt(f_delay(Delta+1),f(J));
    e(J) = f(J) - y(J);
 end;
% Compute the overall filter coefficients
% H(z) = 1 - z^{-Delta}H_{RLS}(z)
 b_overall = [1 zeros(1,Delta-1) -b'];
% Find the frequency response
 [H,w] = freqz(b_overall,1);
 figure(1);
 plot(w,20*log10(abs(H)));
 figure(2);
 zplane(b_overall,1)
```

The input and output spectra are shown in Fig. 5. The filter frequency response magnitude and the pole-zero plot of the filter are shown in Fig. 7. The adaptive algorithm has clearly generated a notch-filter covering the bandwidth of the interference.

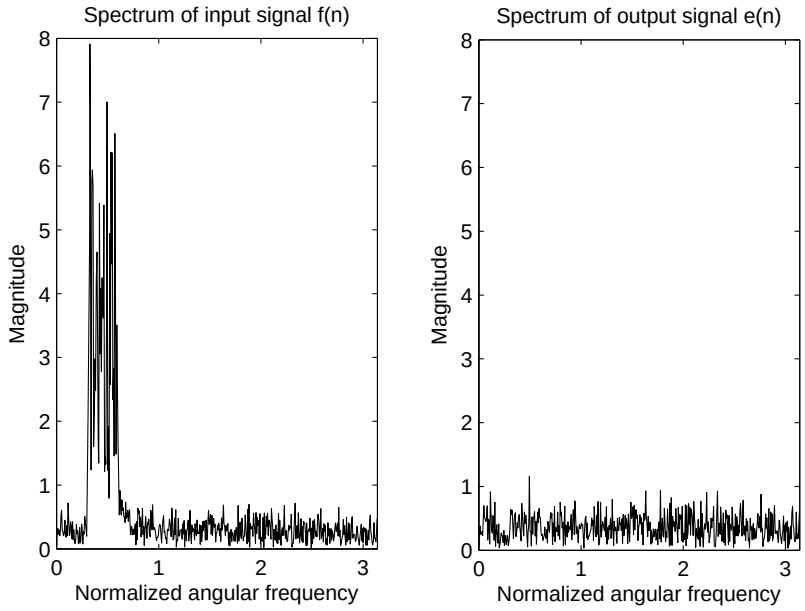

Figure 5: Input and output spectra from a RLS suppression filter with interference in the band $0.3 < \Omega < 0.6$.

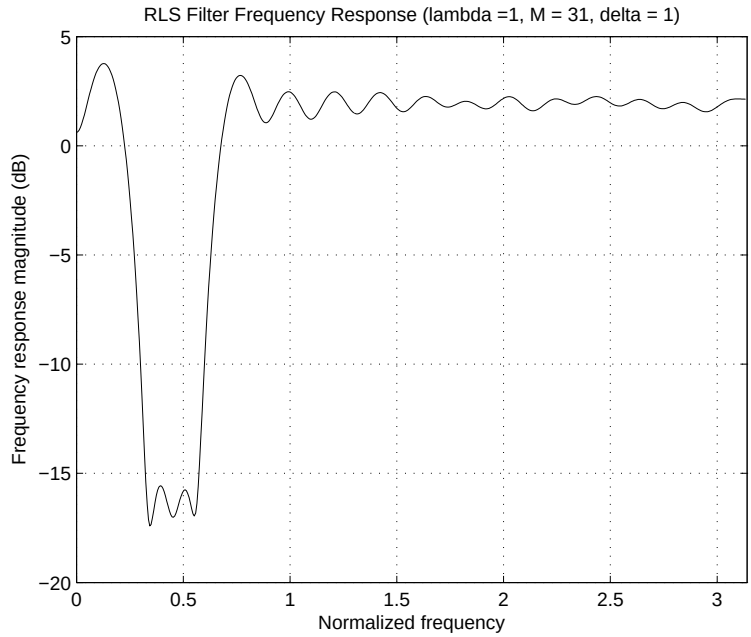

Figure 6: Frequency response magnitude plot of a RLS suppression filter with interference in the band $0.3 < \Omega < 0.6$.

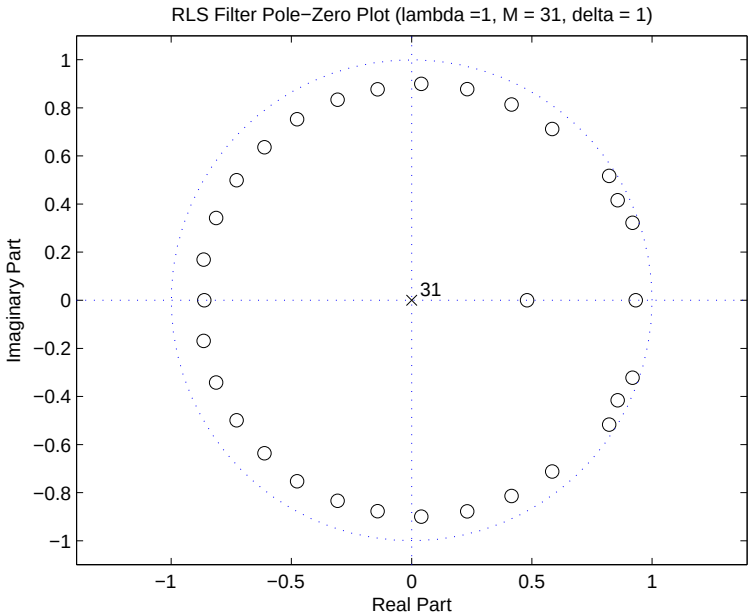

Figure 7: Pole-zero plot of a RLS suppression filter with interference in the band $0.3 < \Omega < 0.6$.

Appendix: The Matrix Inversion Lemma

The *matrix inversion lemma*, also known as the *Woodbury matrix identity* states that for a square $n \times n$ matrix $\mathbf{A}$ with a known inverse $\mathbf{A}^{-1}$, a new updated inverse may be formed

$$\left(\mathbf{A} + \mathbf{BCB}^T\right)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{C}^{-1} + \mathbf{B}^T\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{B}^T\mathbf{A}^{-1}$$

where $\mathbf{B}$ is $n \times k$ and $\mathbf{C}$ is $k \times k$ and non-singular.

First, using the identity $(\mathbf{PQ})^{-1} = \mathbf{Q}^{-1}\mathbf{P}^{-1}$ the right-hand side may be written

$$\mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{C}^{-1} + \mathbf{B}^T\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{B}^T\mathbf{A}^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{I} + \mathbf{CB}^T\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{CB}^T\mathbf{A}^{-1}.$$

Now form the product

$$\begin{aligned}
&\left(\mathbf{A} + \mathbf{BCB}^T\right)\left(\mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{I} + \mathbf{CB}^T\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{CB}^T\mathbf{A}^{-1}\right) \\
= \;& \mathbf{I} + \mathbf{BCB}^T\mathbf{A}^{-1} - \mathbf{B}\left(\mathbf{I} + \mathbf{CB}^T\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{CB}^T\mathbf{A}^{-1} - \mathbf{BCB}^T\mathbf{A}^{-1}\mathbf{B}\left(\mathbf{I} + \mathbf{CB}^T\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{CB}^T\mathbf{A}^{-1} \\
= \;& \mathbf{I} + \mathbf{BCB}^T\mathbf{A}^{-1} - \mathbf{B}\left(\mathbf{I} + \mathbf{CB}^T\mathbf{A}^{-1}\mathbf{B}\right)\left(\mathbf{I} + \mathbf{CB}^T\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{CB}^T\mathbf{A}^{-1} \\
= \;& \mathbf{I} + \mathbf{BCB}^T\mathbf{A}^{-1} - \mathbf{BCB}^T\mathbf{A}^{-1} \\
= \;& \mathbf{I},
\end{aligned}$$

thus proving the lemma.

Notice that if $k = 1$, as in the case of the RLS filter where $\mathbf{C} = \lambda^{-1}$, the updated inverse does not require explicit matrix inversion.