Homework #7 Solutions

Prolog: Problems 7.1 and 7.2 require a maximal length pseudo-noise sequence of length 1023 bits. Length 1023 sequence would require 10 stages, i.e. $2^{10} - 1 = 1023$. The following Web site recommends a connection polynomial with connections at stages 7 and 10:

http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_lfsr/10stages.txt

Matlab code to generate the maximal length PN sequence of length 1023 using version 4.3 of the Communications Toolbox is the following:

```
pn1023gen = commsrc.pn('GenPoly',       [10 7 0], ...
                       'InitialStates', [0 0 0 0 0 1 0 0 0 0], ...
                       'CurrentStates', [0 0 0 0 0 1 0 0 0 0], ...
                       'Mask',          [0 0 0 0 0 1 0 0 0 0], ...
                       'NumBitsOut',    1023);
pn1023seq = round(2 * generate(pn1023gen) - 1);
```

We can generate 10 cycles of the maximal length sequence to generate 10,230 values by changing the value of NumBitsOut from 1023 to 10230. (In version 3.5 of the Communications Toolbox, `commsrc.pn` is called `seqgen.pn`, which has the same arguments.)

7.1 Channel Equalization Using a Least Squares FIR Design.
Johnson, Sethares & Klein, problem 13.3, on page 279:

"Use LSequalizer.m to find an equalizer that can open the eye for the channel b= [1 1 -0.8 -0.3 1 1].

a. What equalizer length n is needed?
b. What delays delta give zero error at the output of the quantizer?
c. What is the corresponding Jmin?
d. Plot the frequency response of this channel.
e. Plot the frequency response of your equalizer.
f. Calculate and plot the product of the two."

Changes: Use a training signal `s` that is a pseudo-noise sequence of length `m=1023` and the channel impulse response

```
b = [1 0.68 0.54 0.25 0.32 0.42 0.82 0.9];
```

Plot magnitude and phase of the channel frequency response using the freqz command. The equalizer will seek to compensate the magnitude and phase response of the channel so that the cascade of the channel and equalizer would give (approximately) an ideal channel of a cascade of gain and delay.

Estimate the computational complexity and memory usage to design the channel equalizer coefficients when using a training sequence of *m* samples and an FIR equalizer of *n* coefficients.

Solution:
The following code obtains the smallest equalizer length *n* and smallest delta required for that *n*.

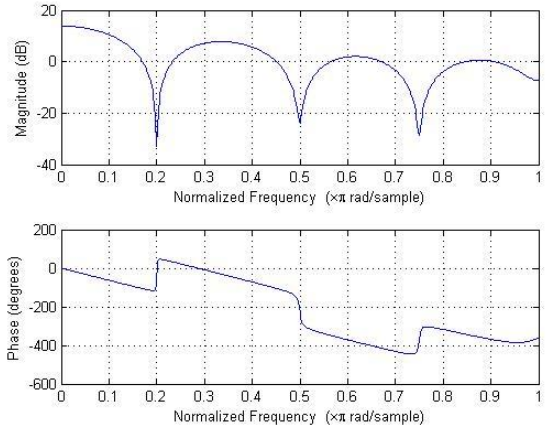
```
% Prob 13.3 of Johnson, Sethares & Klein
% Modified from LSequalizer.m
% LSequalizer.m find a LS equalizer f for the channel b

clear all; close all; clc;

b = [1 0.68 0.54 0.25 0.32 0.42 0.82 0.9];    % define channel
m=10230;                                       % binary source length
pn1023gen = commsrc.pn('GenPoly',      [10 7 0], ...
                       'InitialStates', [0 0 0 0 0 1 0 0 0 0], ...
                       'CurrentStates', [0 0 0 0 0 1 0 0 0 0], ...
                       'Mask',          [0 0 0 0 0 1 0 0 0 0], ...
                       'NumBitsOut',    m);
s=round(2 * generate(pn1023gen) - 1)';  % binary source of length m
r=filter(b,1,s);                        % output of channel

errmin = 100000;
Jerrmin = 100000;
ferrmin = 0;
nerrmin = 0;
deltaerrmin = 0;

for n=3:40                              % length of equalizer - 1
  for delta=1:n                         % use delay <= n * length(b)
    p=length(r)-delta;
    R=toeplitz(r(n+1:p),r(n+1:-1:1));   % build matrix R
    S=s(n+1-delta:p-delta)';           % and vector S
    f=inv(R'*R)*R'*S;                  % calculate equalizer f
    Jmin=S'*S-S'*R*inv(R'*R)*R'*S;     % Jmin for this f and delta
    y=filter(f,1,r);                   % equalizer is a filter
    dec=sign(y);                       % quantize and find errors
    err=0.5*sum(abs(dec(delta+1:end)-s(1:end-delta)));
    if ( err < errmin )
      close all; clear h1;
      errmin = err;
      Jerrmin = Jmin;
      ferrmin = f;
      nerrmin = n;
      deltaerrmin = delta;
      figure;
      [h1,w]=freqz(f,1);
      freqz(f,1);
    end
  end
end
% Print results of search for best equalizer length and delay
nerrmin
deltaerrmin
errmin
Jerrmin
ferrmin
figure; [h2,w]=freqz(b,1);
freqz(b,1);

figure;
freqz(conv(f,b),1);
xlabel('Normalized Frequency (rad/sample)');
ylabel('Magnitude (dB)');
title('Cascade of channel and equalizer');
```

(a) An equalizer length of 36 gave the lowest error (i.e. `nerrmin` was 35). As the number of errors approaches zero, the eye is open and the equalizer will function appropriately.

(b) With an equalizer length of 36, a delay of 17 samples gave zero bit errors. The delay, delta, is the combined delay through the channel and the equalizer. A longer equalizer is helpful in choosing a delta that is more robust. The equalizer must be long enough to erase the effects of the channel, but not too long so as to avoid overmodeling the channel.

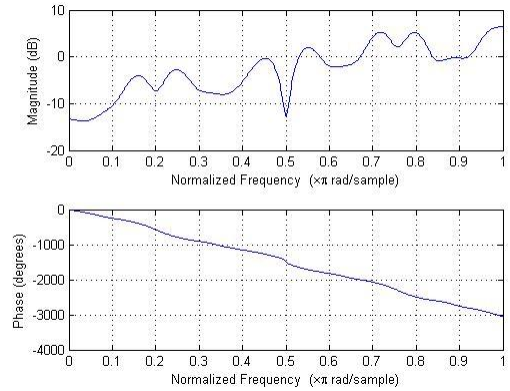
(c) $J_{\min}$ for an equalizer length of 36 coefficients and delay delta of 17 samples is 1393.2.

(d) The frequency response of the channel is plotted below (using freqz):



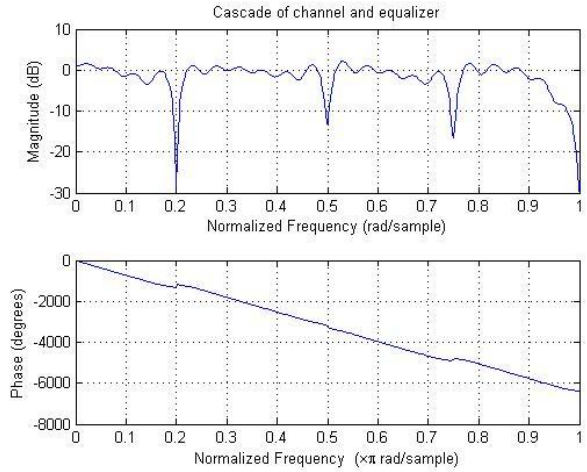
The channel corrupts the transmitted signal. The equalizer will have difficulties recovering frequencies near the nulls in magnitude response around frequencies 0.2π, 0.5π, and 0.75π rad/sample. The channel phase response deviates from linear in small neighborhoods around 0.2π, 0.5π, and 0.75π rad/sample.

(e) The frequency response of the 36-tap equalizer is shown below:



The equalizer passbands are roughly centered at 0.15π, 0.25π, 0.45π, 0.55π, 0.7π, 0.8π and π rad/sample. The equalizer phase response is nearly linear, and hence, the equalizer filter coefficients are nearly symmetric or nearly anti-symmetric about the midpoint. The equalizer design depends on the channel, which is nearly linear in our case. (This is unusual in practice.)

(f) Here is the frequency response of the cascade of the channel with the equalizer plotted by applying `freqz` to the convolution of impulse responses of the channel and equalizer:



The cascade of channel and equalizer should pass all frequencies with unity gain and delay the transmitted signal by a constant amount. A constant delay corresponds to a linear phase response.. A constant fixed delay corresponds to a linear phase response. In this case, the equalized channel has a magnitude response that is fairly close to 0 dB (unity gain) except for the large dips at 0.2π, 0.5π, 0.75π, and π rad/sample. The phase response is nearly linear, except for the deviations at 0.2π, 0.5π, and 0.75π rad/sample.

Computational Complexity. For equalizer with n coefficients, training sequence of m samples, and fixed delay Δ, computing LS equalizer coefficients is by `f=inv(R'*R)*R'*S` where R is q x (n+1), R' is (n+1) x q, R' R is (n+1) x (n+1) and S is a q x 1. Here, $q = m$ + length(b) – Δ – n.

In this problem, $\Delta \le n$, $n \le 40$ and length(b) = 8. Since m = 10230, we'll use $q \approx m$.

Vector S is composed of training sequence samples and is m x 1. Matrix R is composed of received samples and is m x n. R' R takes mn^2 multiplication-accumulate (MAC) operations. R' S takes mn MACs, the inverse takes $2n^3$ MACs, and the final product takes n^2 MACs, for a total of $mn + n^2 + 2n^3 + mn^2$ MACs. (The matrix inverse is really used here to solve a linear system of equations. With vector $\mathbf{x}$ known, we rewrite $\mathbf{y} = A^{-1}\mathbf{x}$ as the solution for $\mathbf{y}$ in $A\,\mathbf{y} = \mathbf{x}$. An n by n system of linear equations can be solved with $2n^3$ MAC operations, as described by the article "Gaussian elimination".) For n = 36 and m = 10000, computational complexity is 13.4 MFLOPS.

7.2 Channel Equalization Using An Adaptive FIR Design.

Johnson, Sethares & Klein, problem 13.9, on page 289:

"Use LMSequalizer.m to find an equalizer that can open the eye for the channel b=[1 1 -0.8 -0.3 1 1].

a. What equalizer length n is needed?
b. What delays delta give zero error in the output of the quantizer?
c. How does the answer compare with the design in Exercise 13.3?"

Changes: Use a training signal `s` that is a pseudo-noise sequence of length `m=1023` and the channel impulse response

```
b = [1 0.68 0.54 0.25 0.32 0.42 0.82 0.9];
```

Estimate the computational complexity and memory usage to design the channel equalizer coefficients when using a training sequence of *m* samples and an FIR equalizer of *n* coefficients.

Solution: I used the following piece of code to obtain the least n and delta required for that n.

```
% Prob 13.9 of Johnson, Sethares & Klein
% Modified from LMSequalizer.
% LMSequalizer.m find a LMS equalizer f for the channel b

clear all; close all; clc;

b = [1 0.68 0.54 0.25 0.32 0.42 0.82 0.9];       % define channel
m=10230;                                  % binary source length
pn1023gen = commsrc.pn('GenPoly',      [10 7 0], ...
                       'InitialStates', [0 0 0 0 0 1 0 0 0 0], ...
                       'CurrentStates', [0 0 0 0 0 1 0 0 0 0], ...
                       'Mask',          [0 0 0 0 0 1 0 0 0 0], ...
                       'NumBitsOut',    m);
s=round(2 * generate(pn1023gen) - 1)'; % binary source of length m
r=filter(b,1,s);                          % output of channel

errmin = 100000;
nerrmin = 0;
deltaerrmin = 0;

for n=3:50
    for delta=1:n
        f=zeros(n,1);              % initialize equalizer at 0
        mu=.002;                   % stepsize
        for i=n+1:m                % iterate
            rr=r(i:-1:i-n+1)';     % vector of received signal
            e=s(i-delta)-f'*rr;    % calculate error
            f=f+mu*e*rr;           % update equalizer coefficients
        end
        y=filter(f,1,r);           % equalizer is a filter
        dec=sign(y);               % quantization

        err=0.5*sum(abs(dec(delta+1:end)-s(1:end-delta)));

        if (err < errmin)
            errmin = err;
            nerrmin = n;
            deltaerrmin = delta;
        end
    end
end

errmin
deltaerrmin
nerrmin
```

I used a mu (step size) of 0.002 to make the adaptive LMS equalizer converge to give zero bit errors over the training sequence.

(a) An equalizer length of 37 gave the lowest error (i.e. `nerrmin` was 36). As the number of errors approaches zero, the eye is open and the equalizer will function appropriately.

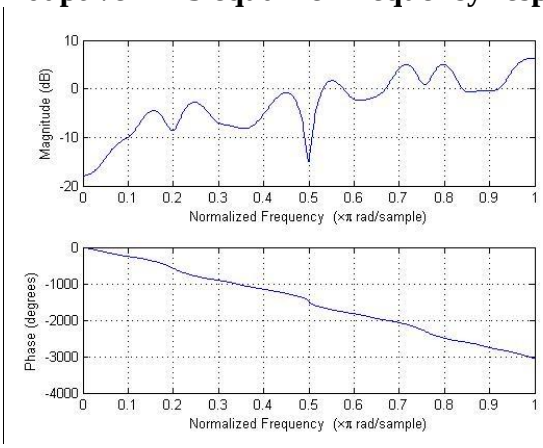
(b) With an adaptive LMS equalizer of length of 37, a delta of 17 samples gave 0 bit errors. A longer equalizer is helpful in choosing a delta that is more robust. The equalizer must be long enough to erase the effects of the channel.

(c) We will compare the two equalizers in several ways.

Number of bit errors. Both equalizers gave zero bit errors over the training sequence. However, the number of bit errors for the adaptive LMS equalizer depends on mu.

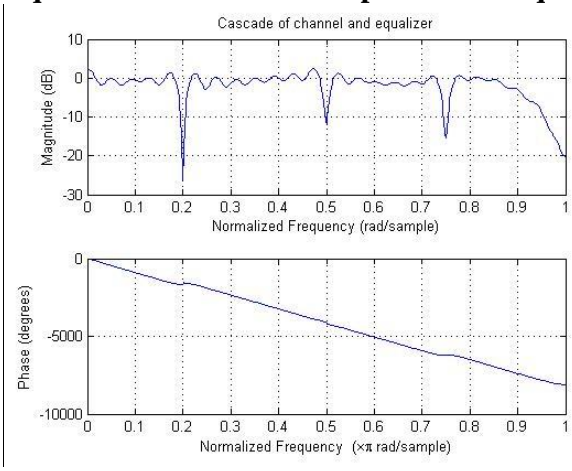
Transmission delay: Both the LMS and LS equalizer had zero bit errors when the delay through the cascade of the equalizer and channel was 17 samples. The actual delay values to minimize the number of bit errors will vary with the channel impulse response.

Adaptive LMS equalizer frequency response:



The adaptive LMS equalizer frequency response is similar to that of the LS equalizer.

Equalized channel for adaptive LMS equalizer:



The cascade of channel and equalizer should pass all frequencies with unity gain and delay the transmitted signal by a constant amount. A constant delay corresponds to a linear phase response. In this case, the equalized channel has a magnitude response that is fairly close to 0 dB (unity gain) except for the large dips at 0.2π, 0.5π, 0.75π, and π rad/sample. The phase response is nearly linear, except for the deviations at 0.2π, 0.5π, and 0.75π rad/sample.

Channel tracking. An adaptive LMS equalizer tracks changes in the channel over the training sequence, whereas the LS equalizer does not. Advantage: adaptive LMS equalizer in practice.

Computational complexity. For an adaptive LMS equalizer, we assume fixed equalizer length n, delay Δ and gain g. There are m training samples/iterations. In an iteration, training requires n multiplications to compute one output sample of the equalizer, scalar multiplications to compute

$e[k]$ and $\mu\, e[k]$, multiplication of scalar $\mu\, e[k]$ and n equalizer coefficients, and addition of two vectors of length n. Training requires $(2n+2)m$ multiply-add operations. The LS equalizer requires $mn+n^2+2n^3+mn^2$ multiply-adds. Computational complexity is 13.4 MFLOPS for the LS equalizer and 0.74 MFLOPS for adaptive LMS equalizer. Advantage: adaptive LMS equalizer.

Because the LS equalizer performs an inversion of the n x n matrix resulting from the calculation of $R'R$, the LS equalizer must be performed in floating point arithmetic for large values of n (e.g. $n > 15$). Matrix inversion requires n^2 divisions. The adaptive LMS equalizer does not require any division operations and hence can more easily be implemented in fixed-point arithmetic.

Memory usage. The LS equalizer stores matrices R and $R'R$, and vector S, using $mn+n^2+m$ words of memory. In the adaptive LMS equalizer, only n training signal samples would need to be available at a given time. The algorithm stores three vectors of length n which uses $3n$ words of memory. For $n = 36$ and $m = 10000$, memory usage is 371,396 words for the LS equalizer and 105 words for the adaptive LMS equalizer. Memory usage for the LS equalizer is too high to fit into on-chip memory for many digital signal processors. Advantage: adaptive LMS equalizer.

Summary. When compared to the LS equalizer, the adaptive LMS (1) has same communication performance for a time-invariant channel, (2) has better performance for a time-varying channel (as would occur in practice), (3) requires orders of magnitude lower computational complexity and memory usage, and (4) can be implemented in fixed-point arithmetic. The only drawback in the adaptive LMS equalizer is the proper choice of the step size.

The least squares method may have issues in the numeric precision of the inv(R' R) calculation. When re-running problem 7.1 with $m = 60$, $n = 20$ and $\Delta = 20$, the matrix R' R is not full rank. It's rank is 20 instead of 21. Its condition number is 5.4608e+35.

7.3 Automatic Gain Control

Johnson, Sethares & Klein, problem 6.29, on page 126.

"Use agcgrad.m to investigate the AGC algorithm.

a. What range of stepsize mu works? Can the stepsize be too small? Can the stepsize be too large?
b. How does the stepsize mu affect the convergence rate?
c. How does the variance of the input affect the convergent value of a?
d. What range of averages lenavg works? Can lenavg be too small? Can lenavg be too large?
e. How does lenavg affect the convergence rate?"

Changes: Replace the Gaussian noise signal generated by the `randn` command with a signal that is BPSK (2-PAM) transmission plus Gaussian noise. For the BPSK transmission, randomly generate symbol amplitudes using $d = 2$; i.e., amplitudes are -1 and +1. Line 3 in `agcgrad.m` would change as follows:

```
r=(sqrt(vr)/sqrt(2))*(randn(n,1) + sign(randn(n,1)));  % generate BPSK plus
noise
```

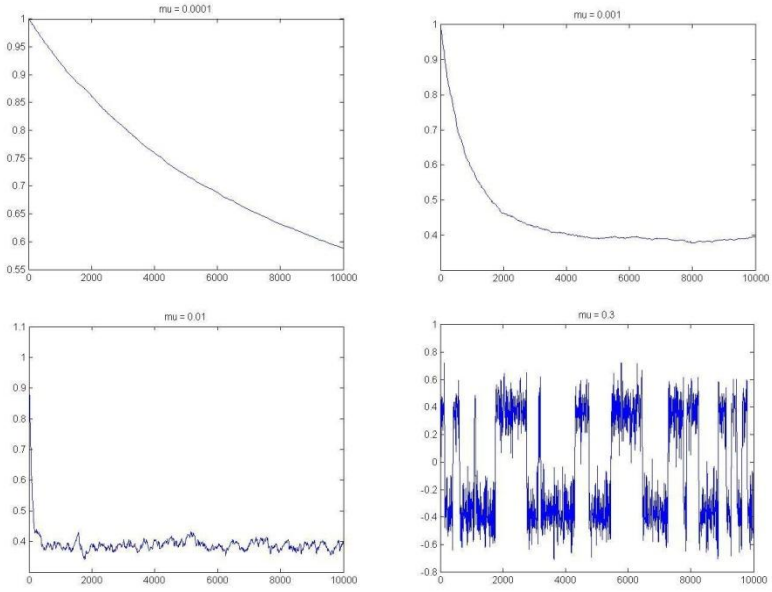
Plot the error surface using `agcerrorsurf.m`. with the following definition of `r`:

```
r=randn(n,1) + sign(randn(n,1));  % generate BPSK plus noise
```

Describe the error surface and any issues with the initial guess for the gain `a`.

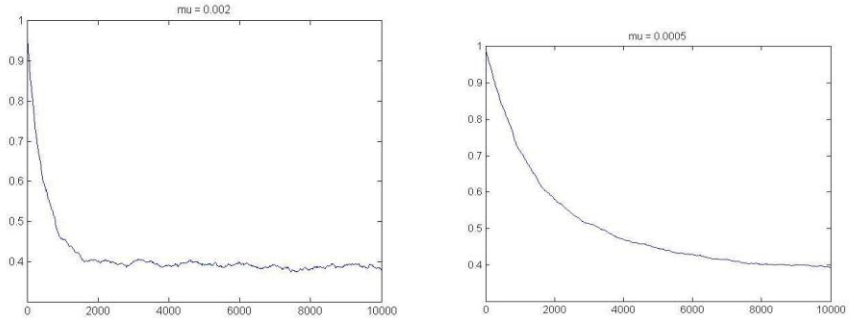
Solution:

a) Restrictions exist on mu. First, mu cannot be negative; otherwise the adaptive algorithm will be searching for a maximum of the cost function J. To find mu values that work, we plot a few:



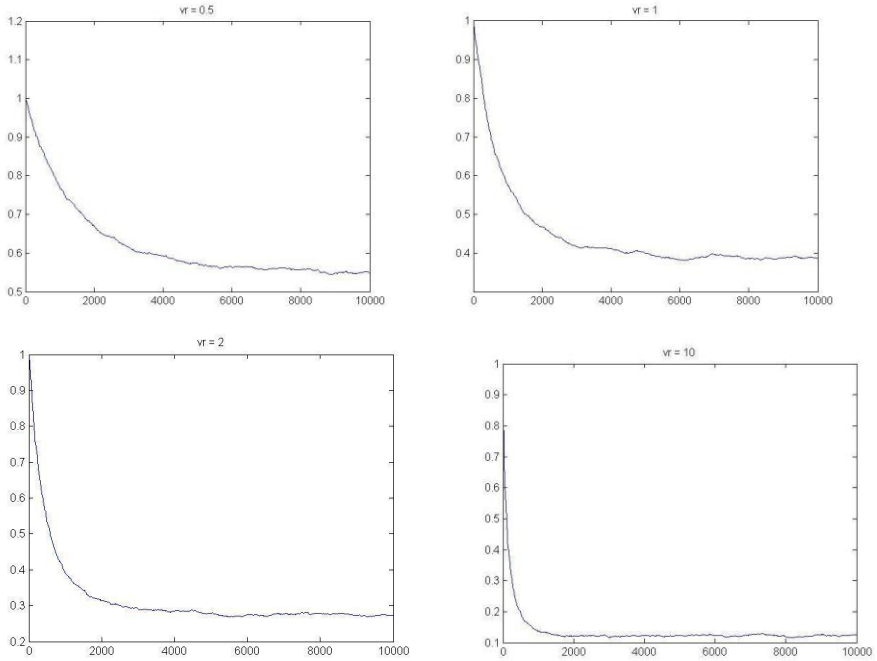
If the magnitude of mu is too small, values for *a* will not converge within the 10000 iterations. We see the effects at each step of slightly overshooting when mu = 0.01 and greatly overshooting when mu = 0.3. The large stepsize sometimes causes the algorithm to overshoot and aim for the other local minima (-0.4), which is incorrect. In short, yes, the stepsize for mu can be too small or too large, and the adaptive algorithm will fail to converge to the correct gain value.

b) Within the range of values for mu that work, as mu is increased, the algorithm converges much more quickly. A smaller mu will give a smoother but slower convergence.



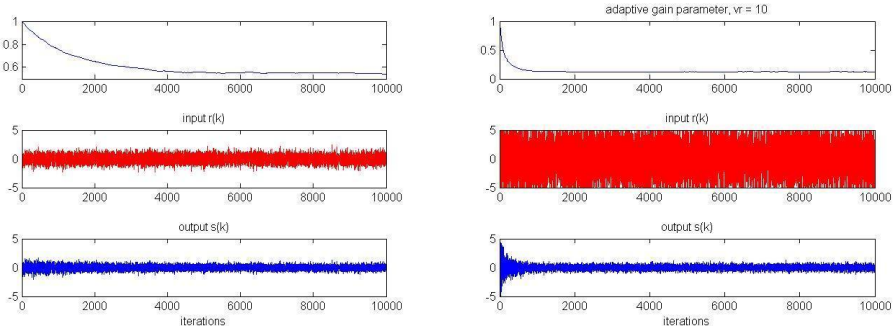
From the figures, when mu is smaller (0.0005), number of iterations for convergence increases.

c) Variance corresponds to the dynamic range of the signal. If the variance is too small, the AGC should increase the dynamic range, and if the variance is too large, the AGC should attenuate it.



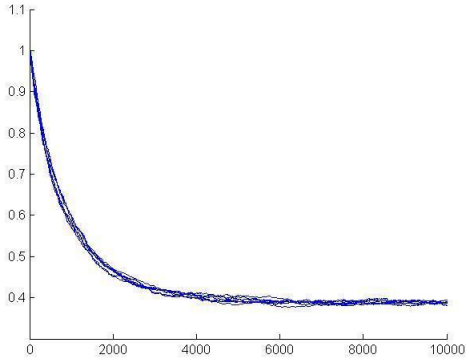
When the variance is larger (say, 10), the dynamic range of the signal is larger, so the AGC converges to a much smaller number, which means the signal is attenuated more. If we take a look at the signal before and after the AGC,

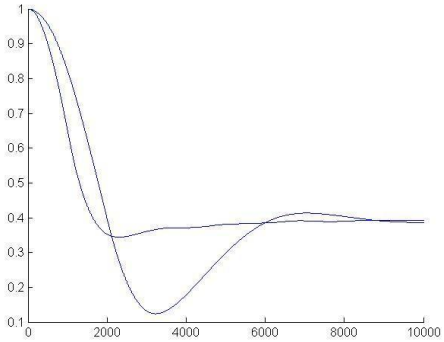
Variance = 0.5 Variance = 10



We can see that the signal with a variance of 10 requires much more aggressive attenuation and, therefore, a smaller gain parameter a.

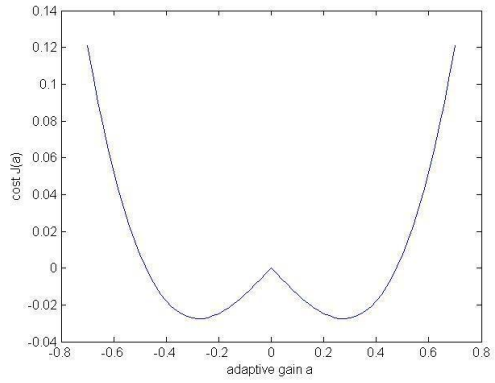
d) If we plot the AGC parameter using different values for lenavg:
lenavg between 1 and 100 lenavg = 1000 and 2000





We can see that the value for lenavg is not very important. We should be careful of letting lenavg get too large, though, because our input signal is a predetermined length (10000). Therefore, there exists an upperbound for the length we can average over and still get a good convergence of the AGC parameter.

e) From the above figure with lenavg between 1 and 100, all good values for lenavg, we can see that varying the value of lenavg doesn't have an effect on the convergence rate.



The error surface plotted to the right shows two minima. As long as our initial guess of *a* is positive, we will converge to the correct minimum.

MATLAB Scripts from in Johnson, Sethares and Klein's *Software Receiver Design* textbook

The Matlab scripts should run "as is" in MATLAB or LabVIEW Mathscript facility.

1. Copy the .m files on your computer from the "SRD - MatlabFiles" folder on the CD ROM: http://users.ece.utexas.edu/~bevans/courses/rtdsp/homework/SRD-MatlabFiles.zip
2. Add the folder containing the .m files from the book to the search path.:
    - In MATLAB, use the addpath command
    - In LabVIEW, open the Mathscript window in LabVIEW by going to the Tools menu and select "Mathscript Window" (third entry), go the File menu, select "LabVIEW MathScript Properties" and add the path.

Johnson, Sethares and Klein intentionally chose not to copyright their programs so as to enable their widespread dissemination.

Discussion of this solution set will be available online soon.