Spring 2024 EE 445S Real-Time Digital Signal Processing Laboratory Prof. Evans

Homework #7 Solutions

Prologue: Problems 7.1 and 7.2 concern equalization. An equalizer compensates distortion experienced by a signal. Examples of distortion:

- Audio– distortion introduced by a microphone or an audio speaker.
- Communications– distortion experienced by the baseband discrete-time signal in the transmitter as it passes through the transmitter analog/RF front end, communication channel, and receiver analog/RF front end to become the baseband discrete-time signal in the receiver.
- Image/video– blur from an out-of-focus camera or artifacts from compression/decompression

We can model the LTI distortion as an FIR filter and the additive thermal noise as a Gaussian random signal. When an equalizer is placed before the distortion, it is called a pre-distorter.

Equalizing frequency distortion has many applications, including pre-distorting an image prior to display/printing, calibrating biomedical instrumentation, and compensating phase distortion in an analog-to-digital converter. In a communication receivers. the equalizer is usually an FIR filter.

Please watch the video lecture from spring 2014 entitled Digital Quadrature Amplitude Modulation Receivers Part 2 on channel equalization from 34:04 to 50:03 (end).

The "best" FIR equalizer may be of any length, including longer than the channel impulse response length. To reduce the computational complexity of your simulations, please limit your search to FIR equalizers having 4 to 41 coefficients.

The bit error rate ***may appear to be zero*** if enough bits aren't used in the simulation. For example, if one expects a bit error rate (BER) of 10^{-2}, i.e. one bit error occurs every 100 bits on average, then one would generally have to run 100/BER bits (i.e. 10,000 bits) to have confidence in the measurement of this bit error rate. It is okay if you cannot drive the number of bit errors to zero.

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. $2^{10} - 1 = 1023$. One realization is a connection polynomial with connections at stages 3 and 10. Matlab code to generate the maximal length PN sequence of length 1023 using version 6.0 of the Communications Toolbox is the following:

```
pn1023gen = commsrc.pn('GenPoly',        [10 3 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);
```

The PN sequence has a period of 1023 values. We can repeat one period of the generated PN sequence ten times to obtain a sequence of 10,230 values, or set the ‘NumBitsOut’ parameter to 10230. Or you can use this maximal length PN sequence with period of 1023 samples.

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 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 J_{min}?
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 `1023` concatenated 10 times and the channel impulse response

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

Plot the 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*+1) coefficients.

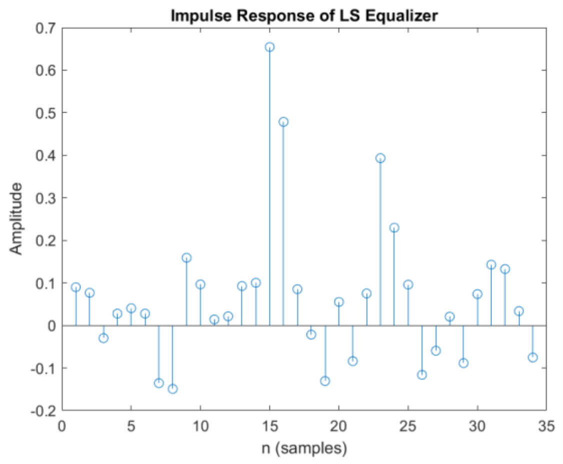
Solution: The code obtains the equalizer order *n* and **`delta`** values for the equalizer that produces the least square error between delayed samples from the sender and the actual received samples:

```
% Prob 13.3 of Johnson, Sethares & Klein
% Modified from LSequalizer.m
% Find a least-squares equalizer f for channel impulse response 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=1023;                                    % binary source length

% With Communications Toolbox installed, use these
% 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

% Alternatively, if Communications Toolbox is not installed
load pn1023seq
s = pn1023seq';
s = [s s s s s s s s s s];                  % duplicate to extend 10 times
r=filter(b,1,s);                            % output of channel
errmin_struct.err = 100000;           % initialize to large value
errmin_struct.Jmin = 100000;          % initialize to large value
errmin_struct.f = 0;
errmin_struct.n = 0;
errmin_struct.delta = 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_struct.err )
            errmin_struct.err = err;
            errmin_struct.J = Jmin;
            errmin_struct.f = f;
            errmin_struct.n = n;
            errmin_struct.delta = delta;
        end
    end
end
eq_channel = conv(errmin_struct.f, b);
% Print results of search for best equalizer length and delay
errmin_struct
```

At the end of simulation, the following Matlab struct contains the FIR LS equalizer filter order `n`, delays delta, and J_{min} when the error has been minimized to zero:

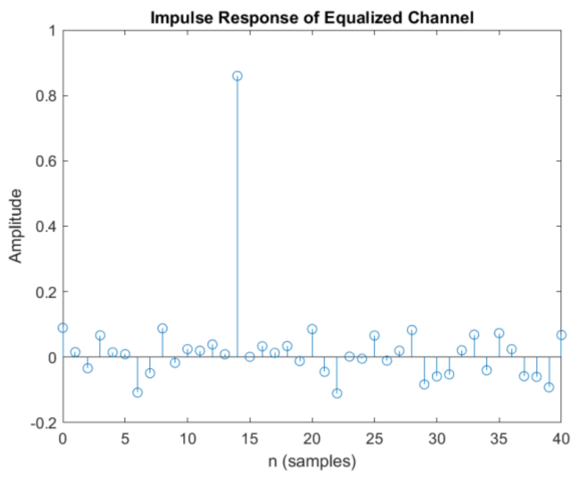
```
errmin_struct =

  struct with fields:

      err: 0
     Jmin: 100000
        f: [34×1 double]
        n: 33
    delta: 14
        J: 1.435384376655984e+03
```

(a) As the number of bit errors approaches zero, the eye diagram becomes more open. The finite impulse response (FIR) equalizer that gave zero bit errors over the training sequence has a filter order of 33 (i.e. `n` in the `errmin_struct` is 33). Hence, the FIR equalizer has 34 coefficients.



[Optional]: The left figure shows the impulse response of the LS equalizer that gives zero bit errors.

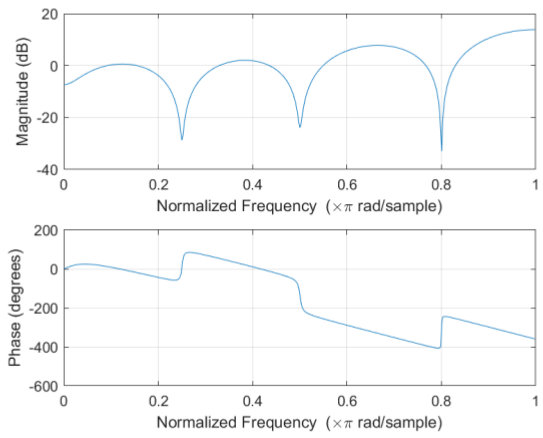
(b) The resulting delays delta is 14 samples. The delays delta is the combined delay due to the channel and the equalizer. The equalizer must be long enough to compensate effects of the channel, but not too long to overly model the channel. A much higher order equalizer without much better improvement on bit errors is an example of diminishing return.



[Optional]: The left figure shows the impulse response of the overall system with the cascade of the channel and the channel equalizer. The coefficients are not symmetric around the peak value. At the receiver, the equalized sample will have a 14-sample delay.

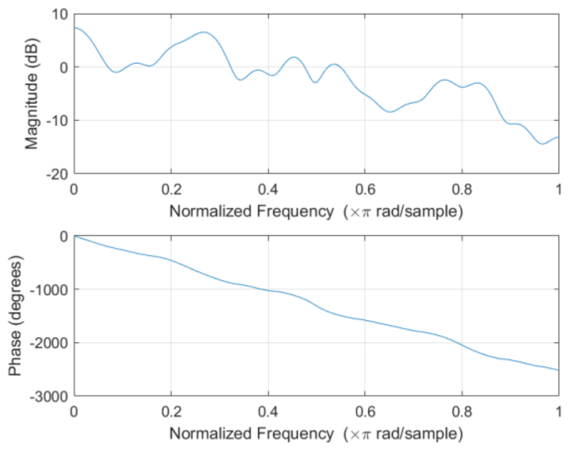
(c) $J_{\min}$ for an equalizer length of 34 coefficients and delay delta of 14 samples is 1435.3843.

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



The channel distorts the transmitted signal. The equalizer will have difficulty in recovering frequencies with attenuation of -20 dB around frequencies 0.25π, 0.5π, and 0.8π rad/sample. The attenuation at the DC is about -8 dB. The channel phase response deviates from linear in small neighborhoods around 0, 0.25π, 0.5π, and 0.8π rad/sample.

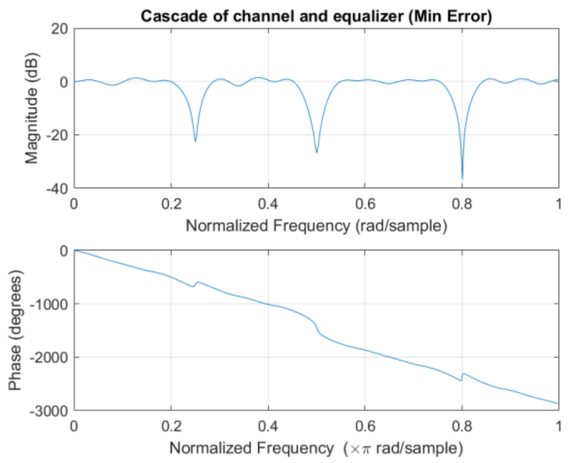
(e) The frequency response of the 34-tap equalizer is shown below:



The equalizer passbands are centered near 0π, 0.25π, 0.45π, and 0.54π rad/sample. The passband at DC has the strongest gain at 7.3 dB, while the gain at the 0.25π rad/sample is 6.4 dB. The gain at 0.45π rad/sample is 1.7 dB and only 0.43 dB gain at 0.54π rad/sample.

The equalizer phase response is nearly linear, and hence, the equalizer filter coefficients are nearly symmetric or nearly anti-symmetric about the group delay term.

(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 ideally 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 close to 0 dB (unity gain with 1 dB ripple) except for the nulls at 0.25π, 0.5π, and 0.8π rad/sample. The phase response is nearly linear, except for the deviations at 0.25π, 0.5π, and 0.8π rad/sample.

The Matlab code below is used to generate and save all the above plots at the end of the simulation:

```
figure; stem(errmin_struct.f); title('Impulse Response of LS Equalizer');
ylabel('Amplitude');
xlabel('n (samples)');
saveas(gcf, 'impulse-res-LS-EQ.png');

figure; [h1,w]=freqz(errmin_struct.f,1); freqz(errmin_struct.f,1);
saveas(gcf, 'freq-res-LS-EQ.png');

figure; [h2,w]=freqz(b,1); freqz(b,1);
saveas(gcf, 'freq-res-channel.png');

figure; freqz(eq_channel,1);
xlabel('Normalized Frequency (rad/sample)');
ylabel('Magnitude (dB)');
title('Cascade of channel and equalizer (Min Error)');
saveas(gcf, 'freq-LS-EQ-channel.png');

figure; stem([0:length(eq_channel)-1], eq_channel);
title('Impulse Response of Equalized Channel');
ylabel('Amplitude'); xlabel('n (samples)');
saveas(gcf, 'impulse-res-LS-EQ-channel.png');
```

Computational Complexity. For an equalizer with (n+1) coefficients, training sequence of m samples, and delay Δ, computing LS equalizer coefficients uses `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 q x 1. Here, $q = m + \text{length}(b) - \Delta - n$. In this problem, $\Delta \leq n$, $n \leq 40$ 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+1) and R' R is (n+1) x (n+1). Computing $R' R$ takes $m(n+1)^2$ multiplication-accumulate (MAC) operations. $R' S$ takes m (n+1) MACs, the matrix inverse takes $2(n+1)^3$ MACs, and the final product takes $(n+1)^2$ MACs, for a total of $\boldsymbol{m(n+1) + (n+1)^2 + (2/3)(n+1)^3 + m(n+1)^2}$ MACs. (The matrix inverse is really used here to solve a linear system of equations. With vector **x** known, we rewrite $\mathbf{y} = A^{-1}\mathbf{x}$ as the solution for **y** in $A\,\mathbf{y} = \mathbf{x}$. An n by n system of linear equations can be solved with $(2/3)n^3$ MAC operations using Gaussian elimination. The MATLAB command change from `f=inv(R'*R)*R'*S` to `f=(R'*R) \ (R'*S)`.)

Course Web site

For *n* = 33 and *m* = 10230, computational complexity is 12.22 MFLOPS.

Here are several hints on calculating computational complexity:

- A scalar element multiplies with a vector of *N* elements takes *N* multiplication operations.
- The dot/inner product of two vectors of length *N* takes *N* multiplications and *N-1* additions, or equivalently *N* MAC operations.
- The product of an *N* by *M* matrix and a column vector of *M* elements (i.e. an *M* x 1 matrix) requires *N* dot/inner products of two vectors of length *M*, which requires *N M* MAC operations.
- The product of an *N* by *M* matrix with an *M* by *N* matrix produces an *N* by *N* matrix with N^2 entries. Each entry in the resulting matrix involves a dot/inner product of two vectors of length *M*. Therefore, the product of the two matrices requires $M\,N^2$ MAC operations.
- Matrix inverse are often avoided to solve a linear system of equations. That is, with vector **x** known, we can rewrite

  $$\mathbf{y} = \mathbf{A}^{-1}\,\mathbf{x}$$

  as solving the following linear system of equations for **y**:

  $$\mathbf{A}\,\mathbf{y} = \mathbf{x}$$

  An *N* by *N* system of linear equations can be solved with (2/3) N^3 MAC operations, as described by the article "Gaussian elimination".

Memory usage. The LS equalizer stores matrices *R*, *R*'*R*, and vector *S*, which would take $m(n+1) + (n+1)^2 + m$ words of memory. **For *n* = 33 and *m* = 10230, memory usage is 359,206 words.**

7.2 Channel Equalization Using An Adaptive FIR Design.

Johnson, Sethares & Klein, problem 13.9, on page 287:

"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 `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.

Prologue: Please see the prologue on the first page of this solution set concerning equalization. For the adaptive least mean squares (LMS) equalizer in this problem, a step-by-step derivation is available on page 3 in the Fall 2020 QAM lecture notes part 1. Also, the section on the Ideal Channel section (page 2) might also be helpful in understanding the derivation.

Please see the following midterm #2 problems and their solutions involving adaptive FIR channel equalizer design: fall 2021 problem 2.3, fall 2019 problem 2.3, fall 2017 problem 2.3, spring 2017 problem 2.3, spring 2017 problem 2.4, spring 2014 problem 2.1, fall 2013 problem 2.1, spring 2013 problem 2.1, and fall 2012 problem 2.2.

Please see the solution for fall 2011 midterm #2 problem 2.4(a) in the course reader for a qualitative comparison between the least squares and adaptive least mean squares methods.

Please augment the code for problem 7.2 by copying the code to calculate and track the minimum error (errmin) from the code for problem. Reduce the step size parameter, mu, to be on the order of 0.001 so that the iterations will converge to a reasonable error value. There are two different error calculations in this problem, and they are easy to confuse.

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

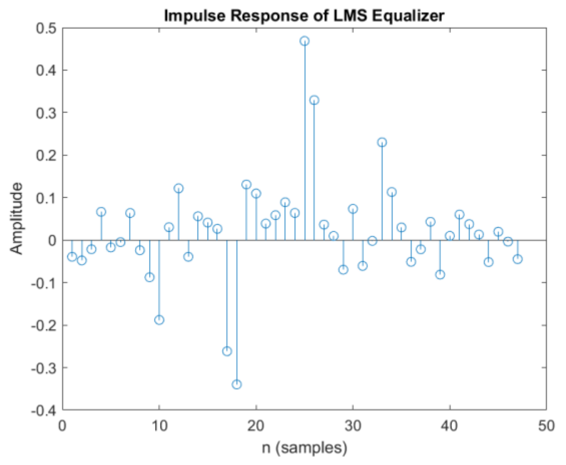
```
% Prob 13.9 of Johnson, Sethares & Klein
% Modified from LMSequalizer.
% Find a least mean squared equalizer f for channel impulse b
clear all; close all; clc;
b = [1 -0.68 0.54 -0.25 0.32 -0.42 0.82 -0.9];       % define channel
m=1023;                                       % binary source length

% With Communications Toolbox installed, use these
% 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

% Alternatively, if Communications Toolbox is not installed
load pn1023seq
s = pn1023seq';
s = [s s s s s s s s s s];                     % duplicate to extend 10 times
r=filter(b,1,s);                              % output of channel
errmin_struct.err = 100000;
errmin_struct.n = 0;
errmin_struct.delta = 0;
errmin_struct.f = 1;
for n=3:50
    for delta=1:n
        f=zeros(n,1);                 % initialize equalizer at 0
        mu=.0035;                      % 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_struct.err)
            errmin_struct.err = err;
            errmin_struct.n = n;
            errmin_struct.delta = delta;
            errmin_struct.f = f;
        end
    end
end
```

Course Web site

```
eq_channel = conv(errmin_struct.f, b);
% Print results of search for best equalizer length and delay
errmin_struct
```

At the end of simulation, the following Matlab struct contains the FIR LMS equalizer filter order n, and delays delta when the error has been minimized to zero:

```
errmin_struct =
  struct with fields:
      err: 0
        n: 47
    delta: 24
        f: [47×1 double]
```

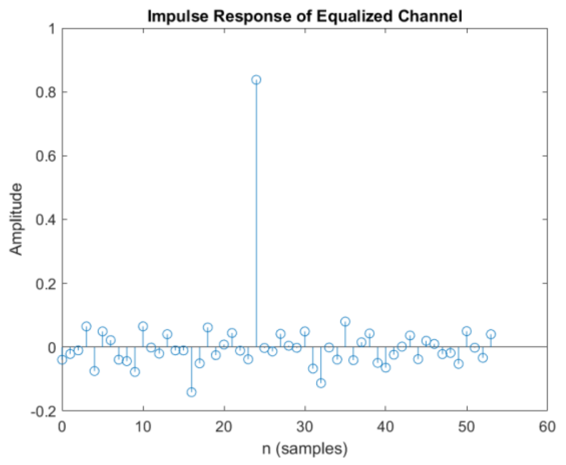


The value of step size, μ = 0.0035 is used to make sure that the adaptive LMS equalizer converges and produces no bit errors over the training sequence.

(a) As the number of bit errors approaches zero, the eye becomes more open. The finite impulse response (FIR) equalizer that gave the lowest number of bit errors over the training sequence has a length of 47 coefficients (i.e. n in the errmin_struct is 47).

[Optional] Figure shows the impulse response of the LMS equalizer that gives 0 bit errors.

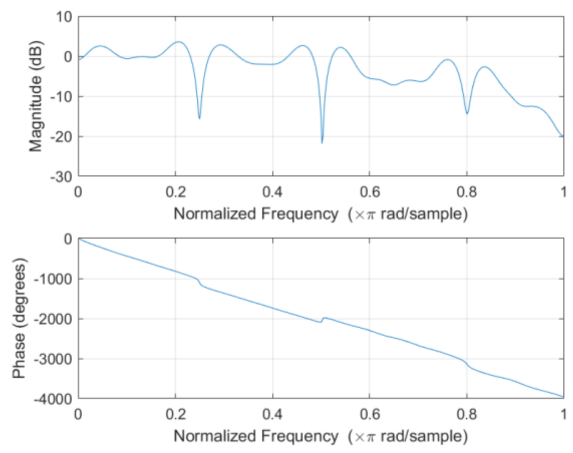
(b) With an adaptive LMS equalizer of length of 47 and resulting delta value of 24 samples, the resulting bit error is zero. The same phenomenon is observed on the Least Squares FIR case.



[Optional]: The left figure shows the impulse response of the overall system of the cascade of the channel and the channel equalizer. The coefficients are not symmetric around the peak value. At the receiver, the equalized sysmte will have a primary delay of 24 samples.

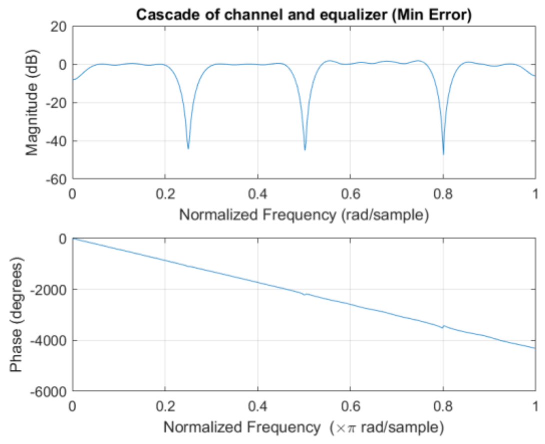
(c) We will compare the two equalizers in several ways.

Adaptive LMS equalizer frequency response: Adaptive LMS equalizer frequency response is very different from the LS equalizer. The passband frequency (above 0dB) is between DC and 0.2π rad/sample. Other passband frequencies that are emphasized include narrow bands at 0.28π, 0.45π, and 0.54π rad/sample**.**



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 μ.

Transmission delay: The delays through the cascade of the equalizer and channel was 24 samples (LMS FIR equalizers) and 14 samples (LS FIR equalizers). The actual delay to minimize the number of bit errors will vary with the channel impulse response. The transmission delay tells us how many initial samples to discard in a received signal.



Equalized channel for adaptive LMS equalizer: The passband ripple and the nulls occur at 0.25π, 0.5π, and 0.8π rad/sample of the equalized channel are same as those of the LS equalizer. However, the DC (-8 dB) and Nyquist frequency (-5 dB) in the LMS equalized channel is worse than the LS equalized channel. The phase response is similar as those obtained by using LS equalizer.

The following Matlab code generates and saves all the above plots at the end of the simulation:

```
figure; stem(errmin_struct.f); title('Impulse Response of LMS Equalizer');
ylabel('Amplitude');
xlabel('n (samples)');
saveas(gcf, 'impulse-res-LMS-EQ.png');

figure; [h1,w]=freqz(errmin_struct.f,1); freqz(errmin_struct.f,1);
saveas(gcf, 'freq-res-LMS-EQ.png');

figure; freqz(eq_channel,1);
xlabel('Normalized Frequency (rad/sample)');
ylabel('Magnitude (dB)');
title('Cascade of channel and equalizer (Min Error)');
saveas(gcf, 'freq-LMS-EQ-channel.png');

figure; stem([0:length(eq_channel)-1], eq_channel);
title('Impulse Response of Equalized Channel');
ylabel('Amplitude'); xlabel('n (samples)');
saveas(gcf, 'impulse-res-LMS-EQ-channel.png');
```

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. At each iteration, training requires n multiplications to compute one equalizer output sample, 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 **(2*n*+2)*m* multiply-add** operations. With m = 10230 and n FIR filter coefficients, the LS equalizer requires $mn+n^2+(2/3)n^3+mn^2$ multiply-adds (please note that n in problem 7.1 is the FIR filter order). Computational complexity is **12.22 MFLOPS for LS** and **0.982 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 m = 10230, memory usage is **359,206 words** for the **LS equalizer (*n*=33)** and **141 words for the adaptive LMS equalizer (*n*=47)**. Memory usage for the LS equalizer is too high to fit into on-chip memory for many processors. Advantage: adaptive LMS equalizer.

Note: Sometimes, the least squares method may have issues in the numeric precision of the inv(R' R) calculation because R' R may not be in full rank or have a high condition number.

Summary. When compared to the LS equalizer in this simulation, the adaptive LMS equalizer (1) has same communication performance for a time-invariant channel, (2) has better performance for a time-varying channel (as would occur in practice) because it tracks changes in the channel over time, (3) requires an order of magnitude lower computational complexity and several orders of magnitude lower 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, μ.

Although not investigated here, the adaptive LMS equalizer will benefit substantially having a longer training sequence for the steepest descent algorithm to converge to a better answer.

7.3. QAM Carrier Recovery

Johnson, Sethares & Klein, problem 16.15, on page 371.

Prolog: You had evaluated the impact of carrier frequency and phase offset in QAM receivers in homework problem 3.2 (JSK problem 5.14) and the impact of phase offset in upconverted PAM receivers in homework problem 6.1 (JSK problem 10.21). Estimating and compensating for carrier frequency and phase offset is known as carrier recovery. For QAM Carrier Recovery, as mentioned in JSK Sections 16.3 and 16.4, in order to reconstruct the message symbols from a demodulated QAM signal, we will need to identify the phase and frequency of the carrier. From Equation (16.7) below, we know how to recover the message m(t) from the output of the receiver s(t). In the equation, we can see our modulating frequency f_c and our modulating phase φ, as well as our demodulating frequency f_0 and our demodulating phase θ.

$$\mathrm{s(t)}=\frac{1}{2}e^{-j2\pi(f_0-f_c)t+j(\varphi-\theta)}m(t) \tag{16.7}$$

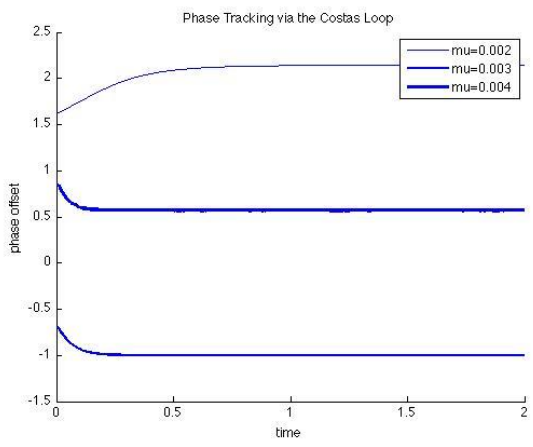
To begin carrier recovery, the receiver can adjust the demodulating phase θ and demodulating frequency f_0 to any desired values. We use this idea to determine what our f_c and φ should be. The QAM Costas loop performs carrier recovery by tracking a time-varying phase which includes differences in transmit and receive carrier frequencies as well as transmit and receive phase offsets. The objective function in JSK (16.10) is periodic so that all local maxima are global maxima.

Problem: Use the preceding code to "play with" the Costas loop for QAM.

a. How does the stepsize mu affect the convergence rate?
b. What happens if mu is too large (say mu=1)?
c. Does the convergence speed depend on the value of the phase offset?
d. When there is a small frequency offset, what is the relationship between the slope of the phase estimate and the frequency difference?

```
% qamcostasloop.m simulate costas loop for QAM
% input vreal from qamcompare.m

r=vreal;                                   % vreal is from qamcompare.m
fl=100; f=[0 .2 .3 1]; a=[1 1 0 0];        % filter specification
h=firpm(fl,f,a);                           % LPF design
mu=.003;                                   % algorithm stepsize
f0=1000; q=fl+1;                           % assumed freq. at receiver
th=zeros(1,length(t)); th(1)=randn;        % initialize estimate vector
z1=zeros(1,q); z2=zeros(1,q);              % initialize buffers for LPFs
z4=zeros(1,q); z3=zeros(1,q);              % z's contain past fl+1 inputs
for k=1:length(t)-1
 s=2*r(k);
 z1=[z1(2:q),s*cos(2*pi*f0*t(k)+th(k))];
 z2=[z2(2:q),s*cos(2*pi*f0*t(k)+pi/4+th(k))];
 z3=[z3(2:q),s*cos(2*pi*f0*t(k)+pi/2+th(k))];
 z4=[z4(2:q),s*cos(2*pi*f0*t(k)+3*pi/4+th(k))];
 lpf1=fliplr(h)*z1'; lpf2=fliplr(h)*z2';  % new output of filters
 lpf3=fliplr(h)*z3'; lpf4=fliplr(h)*z4';  % new output of filters
 th(k+1)=th(k)+mu*lpf1*lpf2*lpf3*lpf4;    % algorithm update
end

plot(t,th),
title('Phase Tracking via the Costas Loop')
xlabel('time'); ylabel('phase offset')
```

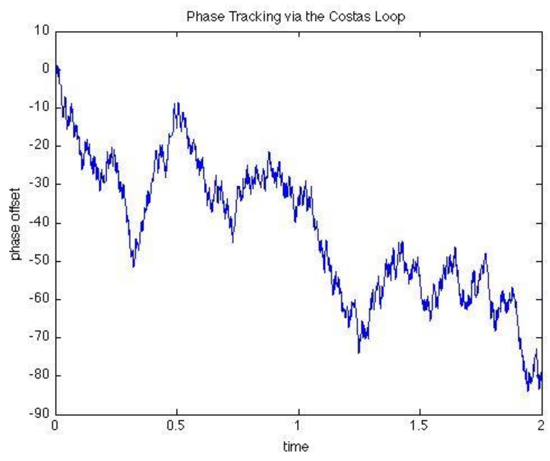
Solution for part a: Effect of step-size mu on the convergence rate.



Graphically an increase in mu increases the convergence rate. In the case of mu = 0.004, we see that the slope of the tracked phase offset is much steeper than for when mu = 0.002, demonstrating that a larger mu increases convergence rate.

Solution for part b: What if mu is too large? (say mu = 1).

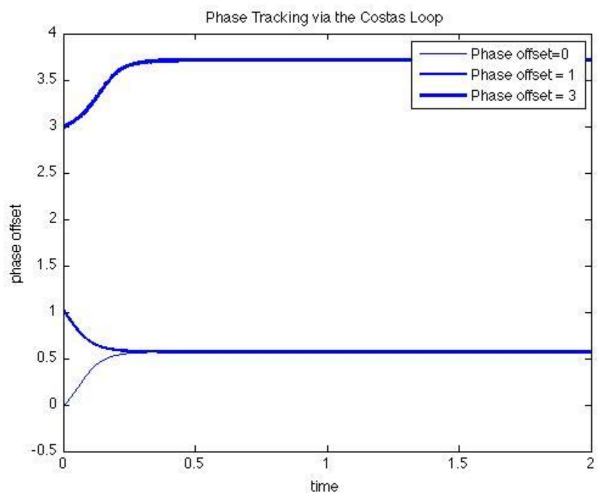
However, if mu is too large (for instance mu=1), then we may not achieve convergence:



Solution for part c: Effect of the phase offset on convergence speed.

The phase offset only slightly affects the convergence rate.

Course Web site



Solution for part d: When there is a small frequency offset, what is the relationship between the slope of the phase estimate and the frequency difference?

If there is a small frequency offset, then the frequency of the carrier is unknown at the receiver, so the phase estimates “converge” to lines. The slope of the lines is proportional to the difference in frequency between the carrier frequency and the frequency at the receiver. This slope can be used to estimate the frequency difference.