Spring 2025 EE 445S Real-Time Digital Signal Processing Laboratory Prof. Evans

Homework #5 Solutions

A ***Hamming window*** is an even symmetric pulse about the midpoint with endpoints having value 0.08. The Matlab command for the Hamming window is hamming. The amplitude values of the causal Hamming window of length N samples is defined as

$$w[n] = 0.54 - 0.46 \cos(2 \pi n / (N - 1)) \text{ for } n = 0, 1, ..., N\text{-}1.$$

Due to even symmetry, group delay of a Hamming pulse of length N samples is $(N\text{-}1)/2$ samples.

Problem 5.1 Steepest Descent. 30 pts.

Johnson, Sethares & Klein, exercise 6.23, page 117. Explore the behavior of steepest descent by running **`polyconverge.m`** with different parameters, but use $J(x) = x^2 - 14x + 49$. Derive the adaptive equations for x to find the minimum value of $J(x)$. For part (a), use values of μ of -0.01, 0.00, 0.01, 0.1, 1.0, 10.0. For parts (b) and (c), use μ of 0.01.

Solution Prologue: *The objective function J(x) has the form of a least squares problem because we can factor the objective function* $J(x) = (x - 7)^2$. *That is, we're trying to find x that minimizes the squared distance to the desired answer of 7. This is also called a least squares problem. For the rest of the semester, we'll be making heavy use of the adaptive least mean squares approach in this problem to solving a wide variety of least squares problems, esp. as related to compensating for impairments experienced by propagating signals. Examples in a communication receiver include carrier frequency and phase recovery, symbol synchronization, automatic gain control and channel equalization. We'll explore those subsystems in homework assignments #6 and #7.*

The basic idea of steepest descent is illustrated in Figure 6.15 at the top of JSK page 116. We are trying to find x^* that minimizes a cost/objective function $J(x)$. We want to update x at each iteration so that we make progress toward the minimum value. The minimum value will be reached when $J'(x^*) = 0$. If our current value for $x > x^*$, then we want to decrease x. If our current value for $x < x^*$, then we want to increase x. We know which direction to update x based on the value of $J'(x)$. This is important because we generally don't know the value of x^*. The update equation for iteration k+1 to realize these goals is shown in JSK (6.5):

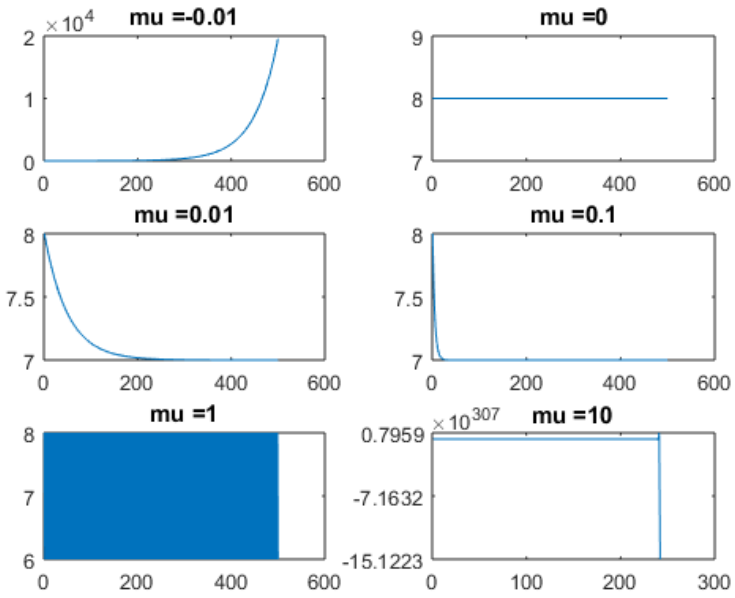
$$x[k+1] = x[k] - \mu \frac{d}{dx} J(x) \Big]_{x=x[k]}$$

Here, the step size μ is a constant positive value that controls by how much x is updated in each iteration. Generally, μ is kept small, e.g. 0.1 or 0.01 or even 0.001. When μ is zero, no update to the initial guess is made during the iterations. When $J'(x[k]) = 0$, the minimum value of the objective function is reached, and $x[k+1] = x[k]$.

a) Use values of μ of -0.01, 0.00, 0.01, 0.1, 1.0, 10.0. Can mu be too large or too small?

Solution: We modify "polyconverge.m" to evaluate the effect of step size μ on the convergence:

```
% polyconverge.m find the minimum of J(x)=x^2-14x+49 via steepest descent
N=500;                              % number of iterations
mu=[-.01 0 .01 .1 1 10];            % algorithm stepsize
x=zeros(size(1,N));                 % initialize x to zero
x(1)=8;                             % starting point x(1)
all = zeros(length(mu), N);
for i=1:length(mu)
    for k=1:N-1
      x(k+1)=(1-2*mu(i))*x(k)+14*mu(i);     % update equation
    end
    all(i,:)= x;
    subplot(3,2,i); plot(all(i,:)); title(strcat('mu = ',num2str(mu(i))));
end
```



In the update equation, a positive (negative) value of μ corresponds to finding the minimum (maximum) of the objective function. When μ is zero, no update to the initial guess is made.

From the plots, we see convergence when the step size μ is 0.01 or 0.1, divergence for μ of -0.01 or 10, oscillation for μ of 1, and trivial convergence for μ of 0. In fact, μ is used to estimate the next value $x[k+1]$ using the current value $x[k]$, and it should be $0 < \mu < 1$.

Given iteration $x_{i+1} = f(x_i)$, initial guess x_0, and interval $[a, b]$ that contains iterates $x_0, x_1, x_2, \ldots$, the fixed-point theorem says that the iterates $x_0, x_1, x_2, \ldots$, will converge to a fixed point $x^* = f(x^*)$ if $|f'(x)| < 1$ for all $x \in [a, b]$. With $f(x) = (1 - 2\mu)x + 14\mu$, we have $f'(x) = 1 - 2\mu$ and guarantee $|f'(x)| < 1$ for $0 < \mu < 1$ regardless of the initial guess.

The choice of μ has an analogy with the location of a single real-valued pole in a discrete-time LTI system. Consider $J(x) = (x - 7)^2$. Then, $J'(x) = 2x - 14$ and

$$x[k+1] = x[k] - \mu\,(2\,(x[k] - 7)) = (1 - 2\,\mu)\,x[k] + 14\,\mu$$

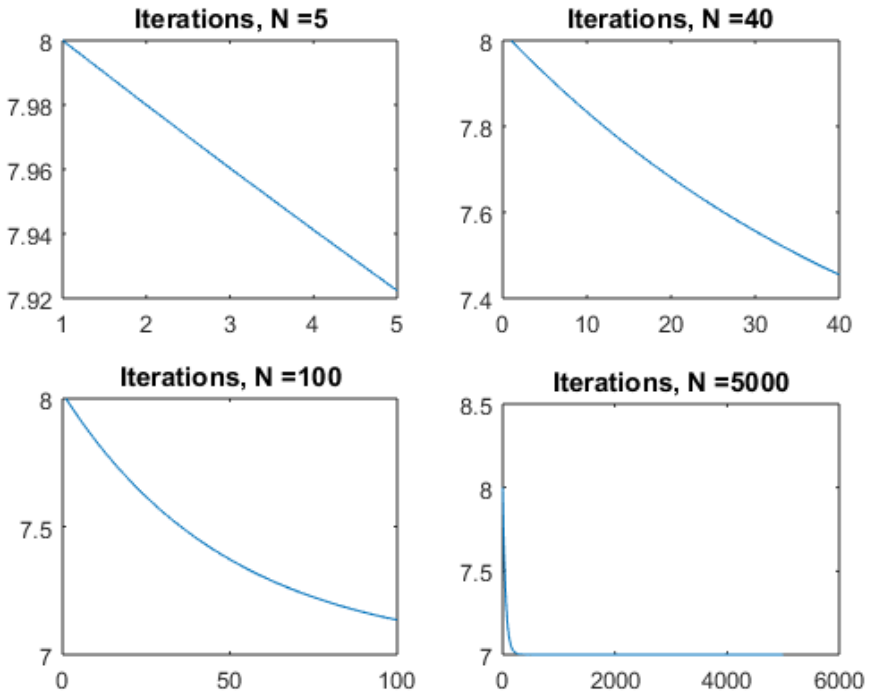
This is a recursive difference equation with a pole at 1 - 2 μ. When μ = 0, the pole is 1, which is BIBO unstable. When μ = 1, the pole is at -1, which is also BIBO unstable. The update is BIBO stable when $0 < \mu < 1$. For μ = 0.01, the pole location is at 0.98, which corresponds to a narrowband lowpass filter. The lowpass filter would smooth out updates to the value of $x[k]$ and hence $x[k]$ would not increase as much in each iteration. See JSK pages 117-118.

b) Use μ of 0.01, and try N = 5, 40, 100, 5000. Can N be too large or too small?

Solution: We modified "polyconverge.m" to see effect of number of iterations on convergence:

```
% polyconverge.m find the minimum of J(x)= x^2-14x+49 via steepest descent
N=[5 40 100 5000];      % number of iterations
mu=.01;                 % algorithm stepsize
x=zeros(1);             % initialize x to zero
x(1)=8;                 % starting point x(1)
y = zeros(4);
for i=1:length(N)
    for k=1:N(i)-1
      x(k+1)=(1-2*mu)*x(k)+14*mu;     % update equation
    end
    y(i)= min(x);
    subplot(2,2,i); plot(x); title(strcat('Iterations, N = ',
int2str(N(i))));
end

y % show minimum
```



The above plots show convergence vs. number of iterations for μ = 0.01.

Course Web site

With N = 5, 40 and 100, we do not have enough iterations to converge to 7 for a step size of $\mu = 0.01$. The minimum values we can achieve are tabulated below:

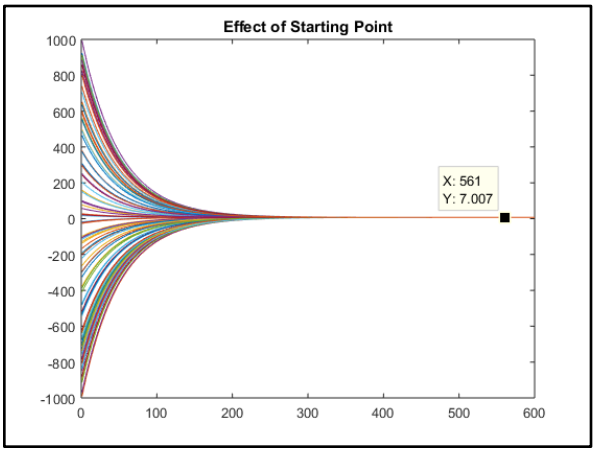
N	Min
5	7.9224
40	7.4548
100	7.1353
500	7.0000

If we use a step size of 0.1, we converge to x = 7 with about 200 iterations. In practice, the number of iterations required depends on the desired numerical precision.

c) Try a variety of values of $x(1)$. Can $x(1)$ be too large or too small?

Solution: Changing the initial guess for ***x*** did not affect the value for ***x*** after convergence. It will affect the speed of convergence, for a fixed step size, depending on how far the starting value is from the minimum. We used the following code to obtain our results:

```
% polyconverge.m find the minimum of J(x)= x^2-14x+49 via steepest descent
N=600;                          % number of iterations
mu=.01;                         % algorithm stepsize
x=zeros(size(1,N));             % initialize x to zero
for i=1:100
    x(1)=randi([-1000 1000],1,1); % starting point x(1)
    for k=1:N-1
      x(k+1)=(1-2*mu)*x(k)+14*mu;
    end          % update equation
    plot (x);
    hold on;
end
hold off;
title('Effect of Starting Point');
```



This plot shows 100 different starting points ranging from -1000 to 1000, all curves converge to our desired value of x = 7.

Epilog: The choice of objective function influences the run-time complexity and convergence behavior of the steepest descent algorithm. In this problem, which uses mean squared error for the objective function, the steepest descent algorithm converges regardless of the initial guess as long as mu is a small enough positive value. If the objective function had been $(x - 7)^4$ instead of $(x - 7)^2$, the per-iteration complexity would have tripled and convergence would have depended on the initial guess and mu (see Fall 2013 Midterm #2.1). Using a mean squared error objective function generally leads to low run-time complexity and good convergence behavior. These adaptive least mean squared (LMS) methods are commonly used in equalizers, echo cancellers, automatic gain control, and phase lock loops, which have applications in speech, audio, image, video and communication systems. Adaptation occurs at each sample to amortize run-time complexity.

Problem 5.2 Frame Synchronization. 40 pts.
Johnson, Sethares & Klein, exercise 9.6, page 177. For the marker in part (b), please use a pseudo-noise sequence of length 31 samples. *Problem relates to homework problems 4.2 & 4.3.*

Another idealized assumption made in **`idsys.m`** is that the receiver knows the start of each frame; that is, it knows where each four-symbol group begins. This is a kind of "frame synchronization" problem and was absorbed into the specification of a parameter `l` which appears in the code as `0.5*f1 + M`. With the default settings, `l` is 125. This problem poses the following question: "What if this is not known, and how can it be fixed?"

a) Verify, using **`idsys.m`**, that the message becomes scrambled if the receiver is mistaken about the start of each group of four symbols Add a random number of 4-PAM symbols before the message sequence, but do not "tell" the receiver that you have done so (i.e., do not change `l`). What value of `l` would fix the problem? Can `l` really be known beforehand?

Solution: The string to be transmitted is represented using eight-bit ASCII characters. Each eight-bit ASCII character is split into four 4-PAM symbols for transmission.

```
%TRANSMITTER
% encode text string as T-spaced 4-PAM sequence
random_symbols = [3 1 -1];
str='01234 I wish I were an Oscar Meyer wiener 56789';
m=[random_symbols letters2pam(str)]; N=length(m); % 4-level signal length N
```

Symbol amplitudes of 3, 1, and -1 were added before the message. Consequently, is a completely different message than the original message:

reconstructed_message '¢ÀÄÆÌÐ $ Ý¥Í $ Ý É" …¸ =Í …È 5 å È Ý¥ ¹ È ÔØÜà'

Optional: The same random symbols are applied to the same receiver but with time-varying fading channel plus automatic gain control (AGC), *idsysmod2.m*. The reconstruction message is scrambled in almost the same way but with slight variation since AGC does not perfectly restore power. reconstructed_message 'äÀÄÆÌÐ $ Ù¥™¥ d Ý É" …¸ =Í …É 5 å È Ý¥ ¹ È ÖØÜà'

The parameter `l` is the delay in samples through the transmitter and receiver in Fig. 9.1 on page 166 in JSK. The transmitter has a pulse shaping filter of length *M* samples, which has a group delay of (*M-1)*/2 samples; since *M* is even, we have a half-sample delay and we'll round up the group delay to *M*. The receiver has a lowpass demodulating filter, which is a linear phase finite impulse response filter with a group delay of f1/2 samples where f1 is the order. The receiver also has a pulse correlator filter (a.k.a. matched filter) of *M* samples, which has a group delay of (*M-1)*/2 samples; since *M* is even, we have a half-sample delay and we'll round up the group delay to *M*. Assuming that the receiver is synchronized with the transmitter with respect to symbol timing, the total delay `l` is `0.5*f1 + M` samples.

When the receiver is not frame synchronized, then the actual delay `l` is `0.5*f1 + M` samples plus an integer multiple of `M` samples, where the integer multiple is the number of symbols added before the message sequence. In practice, the receiver will have to estimate the actual delay because the receiver has different symbol clock circuitry than that in the transmitter.

b) Section 8.5 proposed the insertion of a marker sequence as a way to synchronize the frame. Add a 31-symbol marker sequence just prior to the first character of the text. In the receiver, implement a correlator that searches for the known marker. Demonstrate the success of this modification by adding random symbols at the start of the transmission. Where in the receiver have you chosen to put the correlation procedure? Why?

Solution: The **MATLAB** code was modified as shown below. The random symbol amplitudes, 3, 1 and -1, were added at the start of the transmission.

```
%% #5.2b
close all; clear all; clc;
% TRANSMITTER
% a) encode text string as T-spaced 4-PAM sequence
random_symbols = [3 1 -1];
marker = [1 1 -1 -1 1 1 -1 1 -1 -1 1 -1 -1 -1 -1 1 -1 1 -1 1 1 1 -1 1 1 -1 -1
-1 1 1 1];

str='01234 I wish I were an Oscar Meyer wiener 56789';
m=[random_symbols marker letters2pam(str)];
N=length(m); % 4-level signal of length N
% zero pad T-spaced symbol sequence to create upsampled
% T/M-spaced sequence of scaled T-spaced pulses (T=1)
M=100;                          % oversampling factor
mup=zeros(1,N*M);               % Hamming pulse filter with
mup(1:M:N*M)= m;                % T/M-spaced impulse response
p=hamming(M);                   % blip pulse of width M
x=filter(p,1,mup);              % convolve pulse shape with data
%figure(1), plotspec(x,1/M)     % baseband AM modulation
t=1/M:1/M:length(x)/M;          % T/M-spaced time vector
fc=20;                          % carrier frequency
c=cos(2*pi*fc*t);               % carrier
r=c.*x;                         % modulate message with carrier

% RECEIVER
% am demodulation of received signal sequence r
c2=cos(2*pi*fc*t);              % synchronized cosine for mixing
x2=r.*c2;                       % demod received signal
fl=50; fbe=[0 0.1 0.2 1];       % LPF parameters
damps=[1 1 0 0 ];
b=firpm(fl,fbe,damps);          % create LPF impulse response
x3=2*filter(b,1,x2);            % LPF and scale signal
% extract upsampled pulses using correlation implemented
% as a convolving filter; filter with pulse and normalize
y=filter(fliplr(p)/(pow(p)*M),1,x3);
% set delay to first symbol-sample and increment by M
z=y(0.5*fl+M:M:N*M);

% 1st method
corr=xcorr(marker, z);                % do cross correlation
[crccoef,index]=max(corr);            % location of largest correlation
headstart=length(z)-index+1;          % place where header starts
headstart = headstart + length(marker);   % place where message starts
z = z([headstart:end]);
```

```
figure(2), plot([1:length(z)],z,'.') % plot soft decisions
% decision device and symbol matching performance assessment
mprime=quantalph(z,[-3,-1,1,3])'; % quantize alphabet
cvar=(mprime-z)*(mprime-z)'/length(mprime); % cluster variance
lmp=length(mprime);
pererr=100*sum(abs(sign(mprime-m(1:lmp))))/lmp, % symbol error
% decode decision device output to text string
reconstructed_message=pam2letters(mprime)
```

The message was successfully recovered using the receiver correlator shown above. The correlator was put after the alphabet quantizer (constellation demapping). Since the marker is known at the receiver, the index of the peak of the correlation is found by correlating the received message and marker. The index corresponds to the place where the marker starts. The offset needed to properly recover the message is the sum of the received message length, marker length, negative value of location of largest correlation, and one.

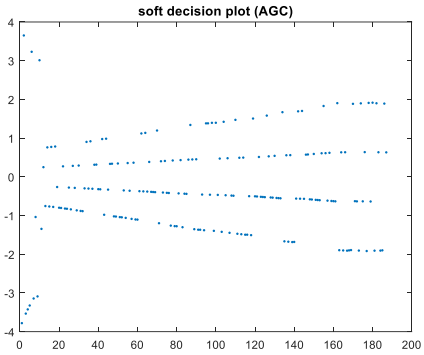
Another possible method implementing the receiver correlator is shown below. The correlator is put after extracting upsampled pulses and before quantization. Since upsampled pulses correlated to upsampled header can provide the location of the marker, the receiver correlator code is added after the convolving filter. Later, the message is downsampled to symbol rate, taking into account the total offset.

```
%RECEIVER
% am demodulation of received signal sequence r
c2=cos(2*pi*fc*t);              % synchronized cosine for mixing
x2=r.*c2;                       % demod received signal
fl=50; fbe=[0 0.1 0.2 1];       % LPF parameters
damps=[1 1 0 0 ];
b=firpm(fl,fbe,damps);          % create LPF impulse response
x3=2*filter(b,1,x2);            % LPF and scale signal
% extract upsampled pulses using correlation implemented
% as a convolving filter; filter with pulse and normalize
y=filter(fliplr(p)/(pow(p)*M),1,x3);
% set delay to first symbol-sample and increment by M

% 2nd method
mup1=zeros(1,N*M);                          % Hamming pulse filter with
mup1(1:M:length(marker)*M)= marker;         % T/M-spaced impulse response
correl = xcorr(mup1,y);                     % do cross correlation
[mp, index] = max(correl);                  % location of largest correlation
header = N*M - index+1 + M*length(marker); % calculate an offset
z=y(0.5*fl+header:M:N*M);                   % downsample to symbol rate

figure(2), plot([1:length(z)],z,'.') % plot soft decisions
% decision device and symbol matching performance assessment
mprime=quantalph(z,[-3,-1,1,3])'; % quantize alphabet
cvar=(mprime-z)*(mprime-z)'/length(mprime); % cluster variance
lmp=length(mprime);
pererr=100*sum(abs(sign(mprime-m(1:lmp))))/lmp, % symbol error
% decode decision device output to text string
reconstructed_message=pam2letters(mprime)
```

Optional: The **MATLAB** code shown below is the scenario when a receiver correlator with a known marker is applied to the transmitted signal with a time-varying fading channel + AGC. The value of stepsize mu in the AGC is changed from 0.0003 to 0.00001 to get more accuracy. As result, *reconstructed_message = 012feeYefifieYefere an Oscar Meyer wiener 5678*, which is not the same original message. However, AGC recovered significant portion of the signal. The transmitter would stay the same in this example.




```
% TIME-VARYING FADING CHANNEL + AGC
ds=pow(r);                  % desired average power of signal
lr=length(r);               % length of transmitted signal
fp=[ones(1,floor(0.2*lr)), 0.5*ones(1,lr-floor(0.2*lr))];  % flat fading
profile
r=r.*fp;                    % apply profile to transmitted signal
g=zeros(1,lr); g(1)=1;      % initialize gain
nr=zeros(1,lr);
mu=0.0001;                  % stepsize
for i=1:lr-1                % adaptive AGC element
    nr(i)=g(i)*r(i);        % AGC output
    g(i+1)=g(i)-mu*(nr(i)^2-ds); % adapt gain
end
r=nr;                       % received signal is still called r

%RECEIVER
% am demodulation of received signal sequence r
c2=cos(2*pi*fc*t);                    % synchronized cosine for mixing
x2=r.*c2;                             % demod received signal
fl=50; fbe=[0 0.1 0.2 1]; damps=[1 1 0 0 ];  % design of LPF parameters
b=firpm(fl,fbe,damps);                % create LPF impulse response
x3=2*filter(b,1,x2);                  % LPF and scale downconverted signal
% extract upsampled pulses using correlation implemented as a convolving
filter
y=filter(fliplr(p)/(pow(p)*M),1,x3);  % filter rec'd sig with pulse;
normalize
% set delay to first symbol-sample and increment by M
z=y(0.5*fl+M:M:N*M);                  % downsample to symbol rate

corr=xcorr(marker, z);                 % do cross correlation
[crccoef,index]=max(corr);             % location of largest correlation
headstart=length(z)-index+1;           % place where header starts
headstart = headstart + length(marker);   % place where message starts
z = z([headstart:end]);

%figure(2), plot([1:length(z)],z,'.')  % soft decisions
% decision device and symbol matching performance assessment
mprime=quantalph(z,[-3,-1,1,3])';     % quantize to +/-1 and +/-3 alphabet
cvar=(mprime-z)*(mprime-z)'/length(mprime),  % cluster variance
lmp=length(mprime);
pererr=100*sum(abs(sign(mprime-m(1:lmp))))/lmp,  % symb err
% decode decision device output to text string
reconstructed_message=pam2letters(mprime)  % reconstruct message
```

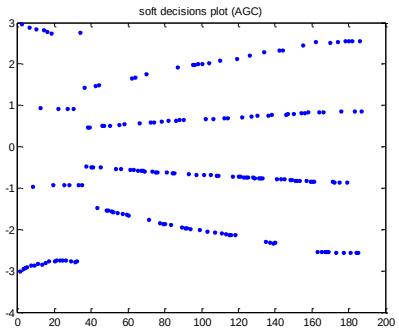
c) One quirk of the system (observed in the eye diagram in Figure 9.8 on page 173) is that each group of four begins with a negative number. Use this feature (rather than a separate marker sequence) to create a correlator in the receiver that can be used to find the start of the frames.

Solution: Different markers, -3 0 0 0 -3 0 0 0 … and -1 0 0 0 -1 0 0 0… , with various lengths, 20, 40, and 80, were used in this problem. As we increase the length, maximum correlation value increases since there are more non-zero numbers that get correlated. Also, if we use a lower absolute value of marker numbers, we would get a smaller value of maximum correlation coefficient. The result from the example with random_symbols = [3 1 -1] is shown below.

Marker length	Max. corr. coefficient (3's)	Max. corr. coefficient (1's)
40	71.83	23.93
60	98.76	32.92
80	119.71	39.90

```
% the receiver correlator comes after downsampling
s = 20; % 40,60,80 the marker with different length
marker = zeros(s,1); % fill in the marker with zeroes
% marker(1:4:s) = -3; % assign -3 for every -3 0 0 0 -3
marker(1:4:s) = -1; % assign -1 for every -3 0 0 0 -3
corr = xcorr(z,marker); % do correlation
[corrvalue index] = max(corr) % find the max correlation coef. and index
headstart = abs(length(z)-index)+1; % calculate an offset
z = z([headstart:end]); % truncate the message according to offset
% and before symbol matching assessment
figure(2), plot([1:length(z)],z,'.') % plot soft decisions
% decision device and symbol matching performance assessment
```

In all cases above, the message was recovered successfully. However, if we use smaller marker length such as 20, then the marker is not long enough to recover the message.



Optional: When the same Matlab code is applied to the receiver with time-varying fading channel +AGC, the *reconstructed_message = 01234 I vifieYefefeeejeZscar Meyer wiener 5678.* As mentioned before, AGC could restore significant power. However, there is still power loss and consequently distortion in received signal.

d) The previous two exercises showed two possible solutions to the frame synchronization problem. Explain the pros and cons of each method, and argue which is a "better" solution.

Solution: Communication delay: Frame synchronization method in part (b) has a delay in the transmitter and receiver that is equal to the marker sequence length (31 symbols) before the message data occurs. In part (c), there is no delay in the transmitter (because a training sequence isn't sent) but there is a delay in the receiver equal to the correlation sequence of 20, 40 or 80 symbols. The method in part (c) needs a correlation sequence of at least 20 symbols to work well. Hence, the receiver incurs a communication delay of 20 symbols.

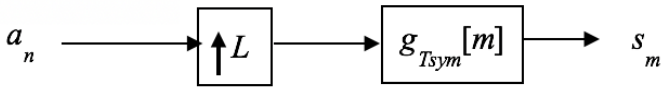
Communication throughput: Once the communication delay in the receiver has occurred as discussed above, each method will decode one symbol at the symbol rate.

Communication reliability: The method in part (b) is more accurate for a 31-symbol correlation sequence vs. a 20-symbol correlation in part (c). Part (b) is correlated to already known pseudorandom marker; hence, it has a higher correlation coefficient. It is especially true if the marker is pseudo-noise (PN) sequence that is robust to frequency distortion and additive noise in the communication channel. The accuracy of the method in part (c) depends on many factors including the length of a negative marker, the value of the marker, and the transmitted signal. However, in general, the chances of detecting 2 negative values spaced apart is high; therefore, this method is less accurate compared to the one in part (b).

Problem 5.3 Baseband PAM Transmitter. 30 pts.

For a baseband pulse amplitude modulation (PAM) transmitter, please compare the implementation complexity of two approaches for interpolation: (a) upsampling followed by a finite impulse response (FIR) pulse shaping filter and (b) polyphase FIR filter bank, by completing the table on slide 13-14. Draw block diagrams for both approaches. Lecture 13 slides 8 through 16 might also be helpful.

(a) The block diagram of the first implementation can be represented in the following form:



The upsampling by $\boldsymbol{L}$ block takes copies each input $\boldsymbol{a_n}$ to the output and then appends $\boldsymbol{L-1}$ zeros. The pulse shaping (FIR) filter replaces zero values with interpolated values depending on the shape of its impulses response. In the following, symbol rate is $\boldsymbol{f_{sym}}$, the number of samples per symbol is $\boldsymbol{L}$, and the duration of pulse shaping filter in symbol periods is $\boldsymbol{N_g}$.

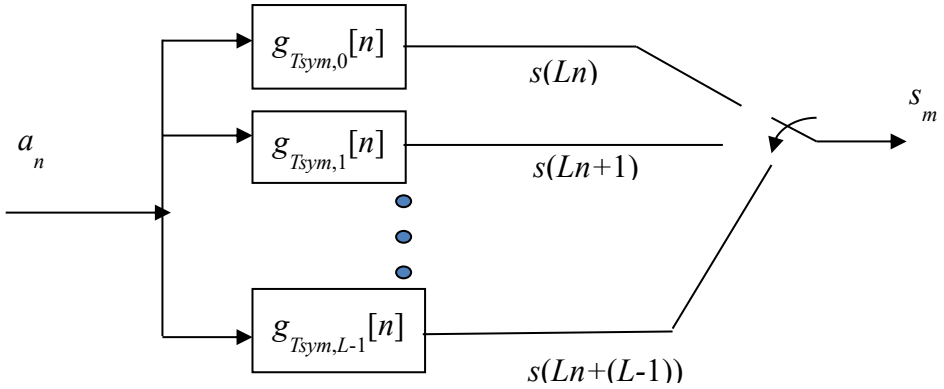
Computation in MACs/s: The pulse shaping filter in the direct form has $\boldsymbol{LN_g}$ coefficients and requires $\boldsymbol{LN_g}$ multiplications per input sample. The input samples arrive at the sampling rate, $\boldsymbol{Lf_{sym}}$. The upsampler does not require multiplication-addition operations. The multiplication-addition operations per second is thus $(\boldsymbol{LN_g})(\boldsymbol{Lf_{sym}})$.

Memory size in words: The pulse shaping (FIR) filter has $\boldsymbol{LN_g}$ coefficients and stores $\boldsymbol{LN_g}$ current and previous input values. The upsampling block has an internal memory of $\boldsymbol{L}$ words.

Memory reads in words/s: To produce an output sample $y[m]$, the pulse shaping (FIR) filter has to read $\boldsymbol{LN_g}$ coefficients and $\boldsymbol{LN_g}$ current and previous input values to compute the output $y[m] = g_0\, x[m] + g_1\, x[m\text{-}1] + \ldots$. The pulse shaping filter also has to read the index to the oldest sample in the circular buffer of the current and previous input values. The pulse shaping (FIR) filter runs at the sampling rate, $\boldsymbol{Lf_{sym}}$. The upsampler reads at the symbol rate $\boldsymbol{f_{sym}}$. Therefore, memory reads in words/s for the direct structure would be $(2\boldsymbol{LN_g+1})(\boldsymbol{Lf_{sym}}) + \boldsymbol{f_{sym}}$.

Memory writes in words/s: The pulse shaping (FIR) filter writes the output sample, updates the index into the oldest sample of the circular buffer of the current and previous input values, and saves the input value to the circular buffer. These two writes occur at sampling rate $\boldsymbol{Lf_{sym}}$. The upsampler will output values at sampling rate $\boldsymbol{Lf_{sym}}$. Memory writes in words/s would be $4\boldsymbol{Lf_{sym}}$.

(b) The block diagram of the second implementation can be represented in the following form:



Computation in MACs/s: Each of the L polyphase FIR filters has N_g coefficients and runs at symbol rate of f_{sym}. The commutator does not require any multiplication-addition operations. The number of multiplication-addition operations per second is $L\ N_g\ f_{sym}$.

Memory size in words: Each of the L polyphase (FIR) filters has N_g coefficients. Since all of the polyphase filters have the same input, they can share the same delay line, i.e. circular buffer of N_g current and previous input values. The commutator requires L words of memory to store the input values. The memory size in words is $LN_g + L + N_g$.

Memory reads in words/s: Each of the L polyphase (FIR) filters has to read N_g coefficients and N_g current and previous input values, and the index to the oldest sample in the circular buffer of current and previous input values. Each polyphase FIR filter runs at the symbol rate, f_{sym}. The commutator reads L inputs and runs at the symbol rate, f_{sym}. The total memory reads in words/s is $(2N_g + 1)\ L\ f_{sym} + L\ f_{sym}$.

Memory writes in words/s: Since the L polyphase filters will share the same delay line, i.e. the circular buffer of N_g current and previous input values, there would be two writes for the circular buffer at the symbol rate, f_{sym}. One write would be to update the current input value into the circular buffer, and the other would be to update the index to the oldest sample in the circular buffer. Each polyphase filter would write its output at the symbol rate, f_{sym}. The commutator writes at the sampling rate, $L\ f_{sym}$. Memory writes in words/s would be $(L+2)\ f_{sym} + L\ f_{sym}$.

	Multiplication additions/s	Memory size in words	Memory reads in words/s	Memory writes in words/s
Direct Structure	$L^2\ N_g\ f_{sym}$	$2\ L\ N_g + L$	$(2\ L\ N_g + 1)\ L\ f_{sym} + f_{sym}$	$4\ L\ f_{sym}$
Filter Bank Structure	$L\ N_g\ f_{sym}$	$(L+1)\ N_g + L$	$(2\ N_g + 2)\ L\ f_{sym}$	$(2L + 2)\ f_{sym}$
Savings	Factor of L	Factor of L in size of circular buffer	Factor of L approximately	None when $L = 1$ Factor of 2 for large L

For a sanity check, the run-time complexity should be identical for both structures when $L = 1$.

The filter bank structure is more efficient in all four implementation complexity measures above vs. the direct structure, and yet the filter bank structure computes the exact same values for the baseband pulse amplitude modulation signal with the same accuracy. This is an unusual situation in which there is a method with lower implementation complexity that gives the same signal quality. It's a rare win-win situation.

When using a platform that could execute polyphase FIR filters in parallel, such as a field programmable gate array (FPGA), the filter bank structure would experience an ***additional*** speedup by a factor of $\boldsymbol{L}$ in multiplication-addition operations per second.