EE 490/599: Communications Systems Topics
Homework 3, Spring 2010: Receiver Demodulation

Homework #3 is due on Thursday, February 11, 2010, at the beginning of class. You may either email your project writeup to me at sheryl.howard@nau.edu, or turn in a hard copy. Late homework is accepted for one week past the due date at a penalty of 20% off.

This homework asks you to simulate a receiver which demodulates a DSB-SC signal. The original signal uses 8-PAM (pulse amplitude modulation) symbols with rectangular pulse-shaping. In this homework, we do not consider any noise or interference from the channel. Your received signal is your transmitted signal. Simulation involves programming 4 blocks:

1. Downconvert by multiplying your received signal $r(t)$ by the carrier $c(t)$, which should be the same as the transmitter carrier. Your output is $w(t) = r(t)c(t)$.

2. Eliminate the high-frequency components of $w(t)$ with a low-pass filter to obtain a baseband sequence $\hat{x}(t)$.

3. Sample the filtered sequence $\hat{x}(t)$ once per symbol period (every $T_s$) to obtain an estimated 8-PAM value $\hat{m}_k$, $\forall k = 0, \ldots, L - 1$.

4. Undo the 8-PAM symbol mapping to get an estimate of your original bits $\hat{v}_i$, $\forall i = 0, \ldots, N - 1$.

These blocks will be combined with your transmitter blocks from HW 2 to simulate a transmitter and receiver.

Each block is described in more detail below.

1 Problem 1: Downconversion of DSB-SC Modulation

Multiply your received signal $r(t)$ by a replica of the transmitter carrier $c(t)$. For now, we’ll assume that the receiver carrier is synchronized to the transmit carrier; thus there is no phase or frequency shift in our receiver carrier. In another homework, we’ll explore the effect of carrier phase and frequency offset.

1. Your Matlab function for downconversion should input your received signal $r(t)$ - which should be your transmitted signal $s(t)$ from HW 2 (you can run a new $s(t)$ for this HW) - and output the downconverted signal $w(t)$.

2. Generate a time vector $t$ that starts at $t = 0$, ends at $t = 159.9$ ms, with time step $\Delta t = 100 \mu$s.

3. Generate a carrier wave $c(t) = \cos(2\pi 1000t)$.

4. Multiply the received sequence $r(t)$ by the carrier wave $c(t)$ above to create the downconverted signal $w(t) = r(t)c(t)$. 

5. Remember that the process of downconversion reduces the amplitude of the baseband component (as well as the high-frequency components) by 1/2. Multiply your \( w(t) \) from the previous step by 2 to account for this attenuation, so that \( w(t) = 2w(t) \).

6. Plot \( w(t) \) versus \( t \). Can you see the 8-PAM values as rectangular pulses of time \( T_s \) in your plot of \( w(t) \)?

2 Problem 2: Low-Pass Filtering

The goal of this section is to filter out the high-frequency \((2f_c)\) components of \( w(t) \), to leave the baseband signal \( \hat{x}(t) \). The Matlab program \texttt{firpm} can be used to design a filter. \texttt{firpm(N,F,A)} designs a linear-phase FIR filter of length \( N + 1 \) that has amplitude described by the vector \( A \) between the frequency bands (in order of increasing frequency) of the vector \( F \). \( F \) is a normalized frequency vector (normalized with respect to the Nyquist frequency \( f_s/2 = 1/(2 \times T_s) \)), that has values between 0 and 1. \( F = 1 \) corresponds to \( f = f_s/2 \), and \( F = 0 \) to \( f = 0 \). The filter will have symmetric frequency response, so \( |H(f)| = |H(-f)| \).

So if you want a low-pass filter \( H(f) = 1 \), \( f < f_0 \) and \( H(f) = 0 \), \( f > f_1 \), then \( A = [1 \ 1 \ 0 \ 0] \). The normalized values of \( f_0 \) and \( f_1 \) are found as \( \hat{f}_0 = f_0/(f_s/2) \) and \( \hat{f}_1 = f_1/(f_s/2) \). \( F = [0 \ \hat{f}_0 \ \hat{f}_1 \ 1] \). The filter response \( H(f) = A[1] \) at \( f = 0 \), = \( A[2] \) at \( f_0 \), and \( A[3]=0 \) at \( f_1 \). The response between \( f_1 \) and the highest frequency \( f_s/2 \) will be \( A[3] \) at \( f_1 \) and \( A[4] \) at \( f_s/2 \); in this case of a low-pass filter, \( H(f) = 0 \) in that frequency range.

In more general terms, the filter will have the response specified by a line connecting the points of the frequency/amplitude pairs \((F(k),A(k))\) and \((F(k+1),A(k+1))\) for \( k \) odd. Response between \((F(k+1),A(k+1))\) and \((F(k+2),A(k+2))\) for \( k \) odd is not specified, although the filter will generally decrease or increase between the two as appropriate.

Note that if your filter length \( N \) is very short (say less than 10), you will not get a very good filter and \texttt{firpm} may be unable to satisfy your requirements. In that case, it will design the best filter it can given the constraints, by minimizing the maximum error between its filter response and your specified filter response. Consider using a filter of at least length 100 to ensure good filter response.

If you type \texttt{help firpm} in Matlab, a good explanation of the command with some examples of lowpass and highpass filters appears.

Remember that linear-phase filters introduce delay in the time-domain output response. This delay will be proportional to the filter length \( N \), that is \( N/2 \). You will have to account for this delay in your output \( \hat{x}(t) \).

1. Design a low-pass filter to remove the high-frequency components from \( w(t) \). The magnitude of your filter frequency response \( H(f) \) should be given as close to the response below as
possible:

\[
|H(f)| = \begin{cases} 
1 & 0 \leq |f| \leq f_0; \\
0, & f_1 \leq |f| < \infty, 
\end{cases}
\]  

(1)

and should decrease from 1 to 0 in the frequency range \(f_0 \leq f \leq f_1\). I recommend you use the Matlab function \texttt{firpm}, but you may use another function or design your own if you choose. You choose what frequencies \(f_0\) and \(f_1\) should be. Ensure that \(f_1 < 2f_c\) so as to filter out the high-frequency components of \(w(t)\). If \(f_0\) is too close to \(f = 0\) or DC, you will filter out part of the pulse-shaped symbol sequence as well. We haven’t considered the bandwidth of \(p(t)\) or \(x(t)\) yet, but they have non-zero bandwidth that will be proportional to \(1/T_s\). Remember that \(p(t)\) is a rectangular pulse, which has a sinc frequency response. Obviously we have to filter out some of the sinc response, but most of the sinc power is contained in the first 3 - 4 sidelobes (with the main lobe containing the peak value). Each lobe of the sinc has width \(1/T_s\).

2. Plot your filter frequency response using the Matlab command \texttt{freqz}. The command \([Hf,f] = \texttt{freqz(h,1,M,fs})\) returns the M-point complex frequency response of the FIR filter with impulse response \(h\), as well as the M-point frequency vector \(f\) in Hz, when \(fs\) is the sampling frequency \(f_s\) in Hz. Then you can easily \texttt{plot(f,abs(Hf))} to display the frequency response of your filter \(h\). Label the axes and title your plot.

3. Write a Matlab function that takes as input the downconverted signal \(w(t)\), and outputs the baseband signal \(\hat{x}(t)\). Your function should filter \(w(t)\) using the filter designed above, and account for any filter delay.

4. Run your Matlab function with your downconverted signal \(w(t)\) from Problem 2. Plot your output baseband signal \(\hat{x}(t)\). Label and title your plot. Can you see the filter delay? Does the delay appear to be about \(N/2\) points in length? Note that because \texttt{filter} generates an output vector that is exactly the same length as your input vector, some symbols are now missing due to the filter delay. How many 8-PAM symbols are missing from your plot, out of the 16 that should be displayed?

5. Use the Matlab command \texttt{conv} instead of \texttt{filter} in your Matlab program. \texttt{conv(h,x)} convolves vectors \(h\) and \(w\), and generates an output that is of length \texttt{length(h) + length(w)} - 1. (\texttt{filter(h,1,w}) also does convolution, but it stops when the output reaches the length of the input \(w\).) Run this Matlab function with your downconverted signal \(w(t)\) from Problem 2.

6. Plot your new output baseband signal \(\hat{x}(t)\). You will need to generate a new time vector that is long enough to include the extra points due to the longer length of the convolution output. Can you see all 16 of the 8-PAM pulse-shaped symbols now? Label and title your plot.

7. Looking at your plot, are the last \(N/2\) points close to zero in value? This is because the convolution operation is running the last \(N\) points of your signal \(w(t)\) through the filter.
response $h$. ($N$ points, rather than $N/2$, due to the filter delay of $N/2$). For the last values of $\hat{x}$, there are not enough values of $w$ left to multiply all $N$ elements of $h$, and several are multiplied by 0, reducing the output. For example, if $w$ has length $M$, 
$$\hat{x}[M + N/2] = \sum_{k=0}^{N-1} h[k]w[M + N/2 - k].$$
But there are no points $w[M + N/2]$ to $w[M + 1]$, so all those terms are zero. There are only values for $w[M - N/2 + 1]$ to $w[M]$, corresponding to $k = N - 1$ down to $k = N/2$. The fact that so many $w$ terms do not exist to calculate the last $N/2$ points reduces the output values for $\hat{x}$ in that region to near zero. By that time, all the 8-PAM symbols have been filtered through $h$. Check this by noting that the last symbol ends $N/2$ points before the end of your plotted sequence $\hat{x}$, assuming your time vector is long enough to display the entire sequence.

8. Remove the delay by eliminating the first $N/2$ points in your output $\hat{x}(t)$. Also remove the last $N/2$ points in your output, as they are just the last vestiges of the convolution operation.

9. Plot your new sequence $\hat{x}$, after the delay and final $N/2$ points have been removed. Label and title your plot. Can you see all 16 of the 8-PAM pulse shaped symbols, with no extraneous signal at the beginning or end?

### 3 Problem 3: Sampling

The purpose of sampling is to convert the analog signal $\hat{x}(t)$ to a discrete sequence $\hat{m}$ that (hopefully) corresponds to our original symbol sequence $m$. If we sample at the correct time (at the peak of each symbol period’s pulse) and with the correct time spacing (every $t = T_s$, where $T_s$ is one symbol period), we should obtain a good match for $m$ - assuming low noise, correct carrier synchronization, and proper filtering. We obtain a discrete sampled sequence $d$ as

$$d[k] = \hat{x}(kT_s + T_s/2), \forall k = 0, \ldots, L - 1,$$

where we are sampling at the center ($T_s/2$) of each pulse.

Note that the values of $d[k]$ are real, because they are the exact value of the sampled $x(t = kT_s + T_s/2)$. Thus $d[k]$ is most likely not exactly an 8-PAM value. Your second task in the sampling problem is to convert the sampled value $d[k]$ to a corresponding 8-PAM value $\hat{m}[k]$. The 8 possible 8-PAM values are $\hat{m}[k] \in \{\pm 1, \pm 3, \pm 5, \pm 7\}$. You will design a mapping from the real value of $d[k]$ to the appropriate 8-PAM value $\hat{m}[k]$. The decision boundaries will be halfway between symbols. For example, if $2 < d[k] < 4$, then $\hat{m}[k] = 3$; if $d[k] > 6$, $\hat{m}[k] = 7$; if $d[k] < -6$, $\hat{m}[k] = -7$, etc.

1. Write a Matlab function that will sample your baseband sequence $\hat{x}(t)$ by sampling every $T_s$ seconds. The original pulse $p(t)$ had width $T_s = 10$ ms, so that is the time between symbols, and should be your sampling time. Ideally, for a rectangular pulse, you can sample at any time during the pulse symbol time to accurately extract the symbol value. However, due to filtering, we do have some ripple and changing from one symbol value to another takes some time. Therefore, I recommend you sample as close to the center of each pulse as possible.
Your function should input the baseband sequence \( \hat{x}(t) \), and output your sampled sequence \( d \), where \( d[k] = \hat{x}(kT_s + T_s/2) \), \( \forall k = 0, \ldots, L - 1 \).

2. Also write a function that will convert the real-valued discrete sequence \( d[k] \) to an 8-PAM symbol sequence \( \hat{m}[k] \). Assuming the delay of \( N/2 \) has been removed, if you sample at \( t = kT_s + T_s/2 \), \( \forall k = 0, \ldots, L - 1 \) to obtain \( d \), you should be able to extract all \( L \) 8-PAM symbol values correctly as \( \hat{m} \). Your function should input the sampled real-valued sequence \( d \), and output your 8-PAM symbol sequence \( \hat{m} \).

3. Plot your 16 output sampled 8-PAM symbol values \( \hat{m} \) versus \( k \). Label and title your plot.

4. Compare \( \hat{m} \) with \( m \) by taking the difference \( D = m - \hat{m} \). Show that \( D \) is an all-zeros vector. You don’t have to print \( D \) out to show this; the \texttt{find} command can be used judiciously to show this. However you choose to show that \( D \) is an all-zeros vector, include your command(s) and their result in your homework file.

4 Problem 4: Symbol-to-Binary Mapping

Now we need to convert the sampled symbol values \( \hat{m} \) to the corresponding binary grouping \( \hat{v} \). We used the following 8-PAM symbol mapping at the transmitter:

<table>
<thead>
<tr>
<th>Binary Group</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-PAM Symbol</td>
<td>-7</td>
<td>-5</td>
<td>-3</td>
<td>-1</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

The receiver needs to reverse this mapping, and go from an 8-PAM symbol value to its group of 3 bits.

1. Write a program that will convert each 8-PAM symbol \( \hat{m} \) to a 3-bit group of the estimated binary sequence \( \hat{v} \) using the mapping above. Your program should input the estimated 8-PAM symbol sequence \( \hat{m} \) and output the binary sequence \( \hat{v} \), which will have length \( N \).

2. Include your program.

3. Test your program by running a test case:
   Use the input symbol values \( \hat{m} = [3 -5 7 1 -3 5 -1] \). What is your output binary sequence \( \hat{v} \)? Show the result from your program.

4. Now convert your sampled 8-PAM symbol vector \( \hat{m} \) of Problem 3 to a binary sequence \( \hat{v} \).
   Include the first 12 and last 12 output values of \( \hat{v} \) in your HW writeup. You do not have to print out the entire vector \( \hat{v} \).

5. Compare your estimated binary sequence \( \hat{v} \) with the original binary sequence at the transmitter, \( v \) by generating the difference vector \( E = v - \hat{v} \). Are there any errors?