Homework #6 Solutions

**6.1 Modified Phase Locked Loop (PLL).**

**Prolog:** The received signal $r(t)$ with carrier frequency $f_c$ passes through a pre-emphasis step consisting of a squaring block followed by a bandpass filter with center frequency of 2 $f_c$, as shown in Fig. 10.3 of Johnson, Sethares and Klein. The output of the bandpass filter is $r_p(t) \sim \cos(4\,\pi\, f_c\, t + 2\phi + \psi)$. [JSK, page 197] The PLL modulates "the (processed) received signal $r_p(t)$ of Figure 10.3 down to DC, using a cosine of known frequency 2 $f_0$ and phase $2\theta + \psi$. After filtering to remove the high-frequency components, the magnitude of the DC term can be adjusted by changing the phase. The value of $\theta$ that maximizes the DC component is the same as the phase $\phi$ of $r_p(t)$." [JSK, page 203]. We assume $f_0$ is close in value to $f_c$. [JSK, page 204] In code in `pllconverge.m` to implement the PLL, "the firpm filter creates an h with a zero phase at the center frequency and so $\psi$ is set to zero." [JSK, page 204]

Johnson, Sethares and Klein, exercise 10.17, pages 205-206. ***Errata:*** In the problem statement, the equation for $u(k\,T_s)$ should have been $u(k\,T_s) = r_p(k\,T_s)\cos(4\,\pi\, f_0\, k\,T_s + 2\,\theta)$. That is, there should have been a factor of "2" before "$\theta$". The rest of the problem statement is correct. In repeating the problem statement below, we have fixed the equation for $u(k\,T_s)$.

Many variations on the basic PLL theme are possible. Letting

$$u(k\,T_s) = r_p(k\,T_s)\cos(4\,\pi\, f_0\, k\,T_s + 2\,\theta)$$

the preceding PLL corresponds to a performance function of

$$J_{\text{PLL}}(\theta) = \text{LPF}\{\, u(k\,T_s)\,\}$$

Consider the alternative

$$J(\theta) = \text{LPF}\{\, u^2(k\,T_s)\,\}$$

which leads directly to the algorithm

$$\theta[k+1] = \theta[k] - \mu\, LPF\left\{ u(k\,T_s)\left.\frac{du(k\,T_s)}{d\theta}\right|_{\theta=\theta[k]} \right\}$$

which is

$$\theta[k+1] = \theta[k] - \mu\,\text{LPF}\{\, r_p(k\,T_s)\sin(4\,\pi\, f_0\, k\,T_s + 2\,\theta[k])\cos(4\,\pi\, f_0\, k\,T_s + 2\,\theta[k])\,\}$$

(a) Modify the code in pllconverge.m to "play with" this variation on the PLL. Try a variety of initial values theta(1). Are the convergent values always the same as with the PLL?

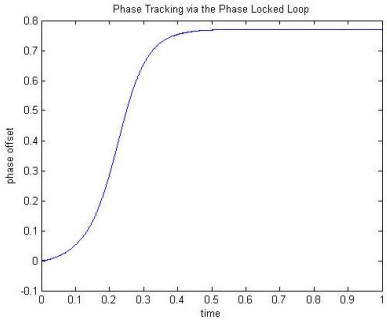
**Solution:** The following code updates the initial equation of $J_{PLL}(\theta) = \text{LPF}\{u(kT_s)\}$ to the modified version: $J(\theta) = \text{LPF}\{u^2(kT_s)\}$. This changes the update equation to

$$\theta[k+1] = \theta[k] - \mu\,\text{LPF}\{\, r_p^2(kT_s)\sin(4\pi\, f_0\, k\,T_s + 2\,\theta[k])\cos(4\pi\, f_0\, k\,T_s + 2\,\theta[k])\,\}$$

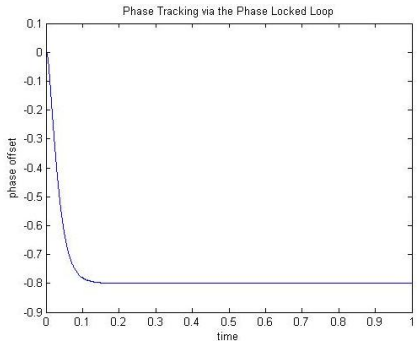
```
% pllconverge.m simulate Phase Locked Loop

Ts=1/10000; time=1; t=Ts:Ts:time;     % time vector
```

```
fc=1000; phoff=-0.8;                    % carrier freq. and phase
rp=cos(4*pi*fc*t+2*phoff);              % simplified rec'd signal
fl=100; ff=[0 .01 .02 1]; fa=[1 1 0 0];
h=firpm(fl,ff,fa);                      % LPF design
mu=.003;                                % algorithm stepsize
f0=1000;                                % freq at receiver
theta=zeros(1,length(t)); theta(1)=0; % initialize estimates
z=zeros(1,fl+1);                        % initialize LPF
for k=1:length(t)-1                     % z contains past inputs
  z=[z(2:fl+1), (rp(k).^2)*sin(4*pi*f0*t(k)+2*theta(k))*cos(4*pi*f0*t(k)+2*theta(k))];
  update=fliplr(h)*z';                  % new output of LPF
  theta(k+1)=theta(k)-mu*update;        % algorithm update
end
plot(t,theta)
title('Phase Tracking via the Phase Locked Loop')
xlabel('time'); ylabel('phase offset')
```

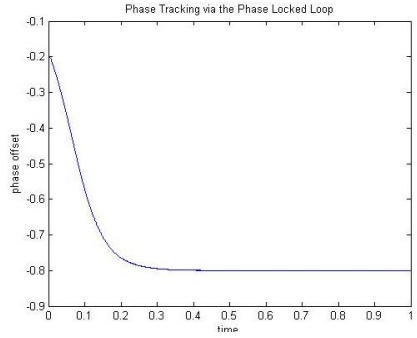
Using an initial value of $\theta[1] = 0$, the following plot shows that it converges to about 0.77 rad.




Many phase locked loop algorithms can make a 90° mistake. The correct phase offset is -0.8 rad. Adding $\pi/2$, or 1.57 rad, and one gets 0.77 rad, which is the value tracked above. For the original PLL algorithm converges more quickly to the correct phase offset for an initial value of $\theta[1] = 0$.



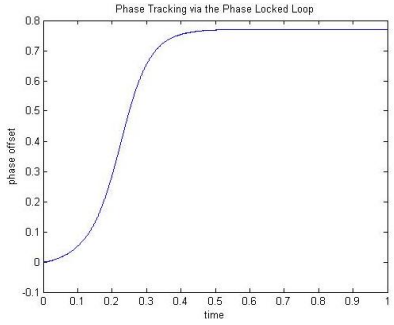

If $\theta[1]$ is set to -0.2 rad, the modified PLL experiences a 90° phase shift and converges to the correct value of -0.8 rad, as shown below. The initial PLL algorithm produces a similar result, but, again, convergence occurs more quickly.

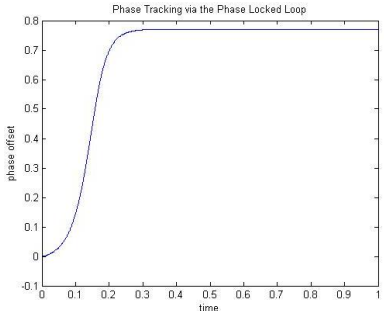


For all values of $\theta[1]$ between about -0.5 rad and 0.5 rad, the initial PLL algorithm converges consistently to -0.8 rad. However, the modified PLL algorithm produces different results for different initial values. For negative values of $\theta[1]$ in the same range, the algorithm converges correctly to a -0.8 rad phase offset. For non-negative values of $\theta[1]$, the algorithm produces a convergence value of about 0.77 (which is -0.8 + $\pi$/2). Therefore, the convergent values are not always the same as those from the original PLL algorithm.

(b) How does μ affect the convergence rate?

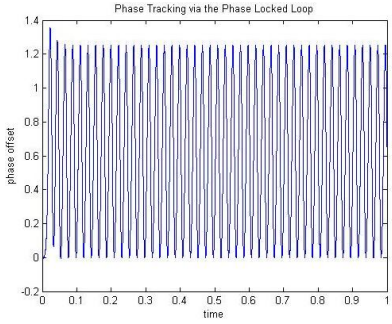
**Solution:** Changing $\mu$ affects the convergence rate. Incrementing $\mu$ increases the convergence rate. For example, the following plots show convergence when $\mu$ = 0.003 and when $\mu$ = 0.005.



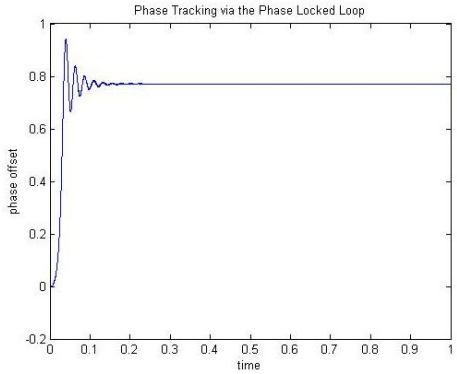

(c) How does μ affect the oscillations in θ?

**Solution:** For low values of $\mu$, such as 0.001< $\mu$ <0.005, oscillations are so small that $\theta$ appears to converge exactly to a specific value. As $\mu$ increases, oscillations become more noticeable. The following plot shows that $\mu$ = 0.2 creates large oscillations around the original convergence value of 0.77 rad. As $\mu$ increases, the oscillations increase in amplitude, until mu becomes so large that the phase signal is corrupted and the phase offset is no longer recoverable (for example, $\mu$ = 1).

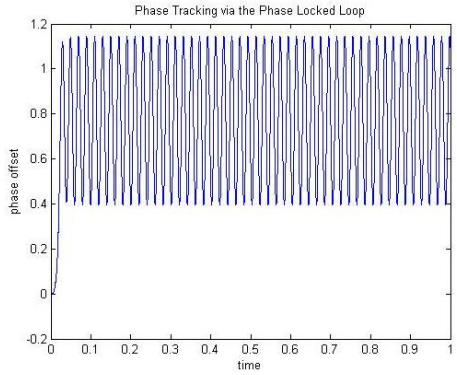



(d) What happens if μ is too large (say μ = 1)?

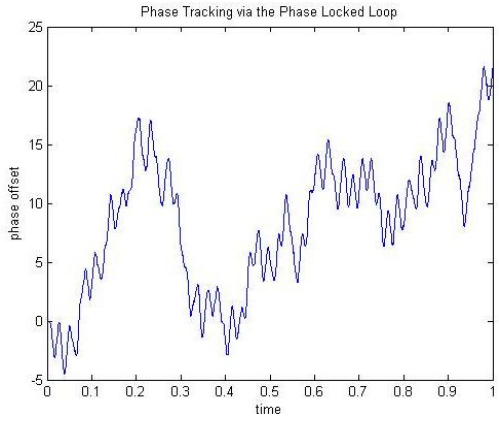
**Solution:** Incrementing $\mu$ only increases the convergence rate to an extent. Once it is increased beyond 0.01, the plots begin to experience Gibbs' Phenomena at the edges. The following plot shows convergence when $\mu$ = 0.05.




Once $\mu$ reaches 0.1, the convergence plots simply oscillate around the desired convergence value, instead of converging. Here's $\mu = 0.1$.

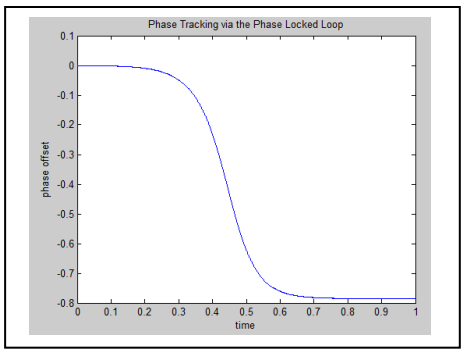


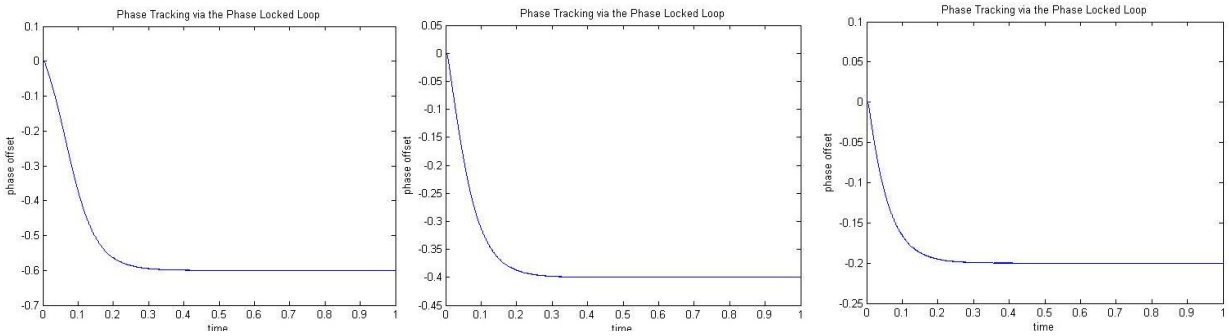

For $\mu >= 1$, the information is completely destroyed, and the convergence value for the phase offset is unrecoverable. The following plot results when $\mu = 1$.




(e) Does the convergence speed depend on the value of the phase offset?

**Solution:** Changing the phase offset does affect the convergence speed (since the phase converges to the phase offset value). The plot on the right is for a phase offset of $-\pi/4$, and it takes about 0.7s for convergence. The plots on the next page are for phase offsets of -0.6, -0.4, and -0.2. Each takes about 0.3s for convergence.

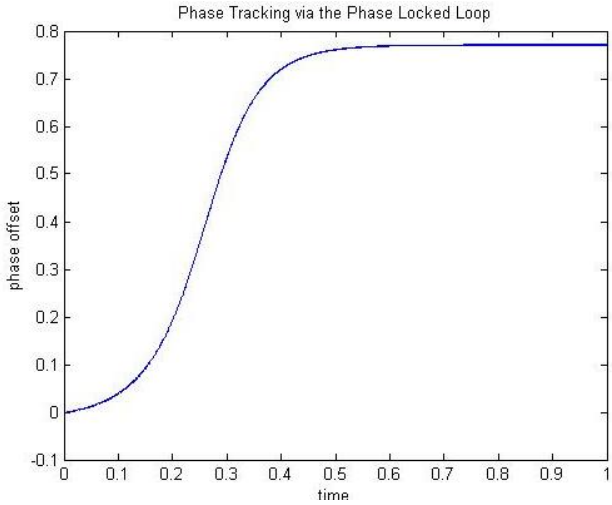





As can be seen from the plots, there seems to be little difference in convergence speeds between a larger and a smaller value of the phase offset.

(f) What happens when the LPF is removed (set equal to unity)?

**Solution:** Setting the LPF as unity effectively removes the LPF and replaces it with an all-pass filter. Doing so results in a less-clean convergence of values. Whereas the initial algorithm with the LPF converged clearly, removing the LPF results in a much thicker band of values where the phase should be converging, as can be seen in the plot (for phase offset = -0.8).




(g) Draw the corresponding error surface.

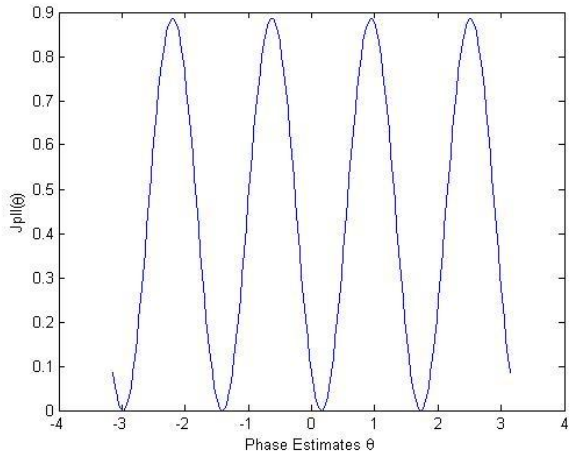
**Solution:** Error surface is $J(\theta) = \text{LPF}\{u^2(k\,T_s)\}$, where $u(k\,T_s) = r_p(k\,T_s)\cos(4\,\pi\,f_0\,k\,T_s + 2\,\theta)$ :

$$J(\theta) = \text{LPF}\ \{\ (\cos(4\,\pi\,f_0\,k\,T_s + 2\,\phi)\cos(4\,\pi\,f_0\,k\,T_s + 2\,\theta))^2\ \}$$

The lowpass filter is applied in the discrete-time variable $k$. For a fixed value of $\theta$, the lowpass filter has no effect. The following Matlab code plots the error surface for $k = 1$ and $\phi = -0.8$.

```
Ts=1/10000;                             % sampling time
fc=1000; phoff=-0.8;                    % carrier freq. and phase offset
f0=1000;                                % estimated carrier freq.

t = Ts;                                 % k = 1
rp=cos(4*pi*fc*t+2*phoff);
theta = -pi : pi/100 : pi;
Jpll = (rp * cos(4*pi*f0*t + 2*theta)).^2;
```

```
plot(theta,Jpll);                    % plot J(theta) vs theta
xlabel('Phase Estimates \theta')
ylabel('Jpll(\theta)')
```




The maximum value of the error surface (objective function) occurs when $\theta = \phi$ as well as when $\theta = \phi + n\ \pi / 2$ for any integer $n$.

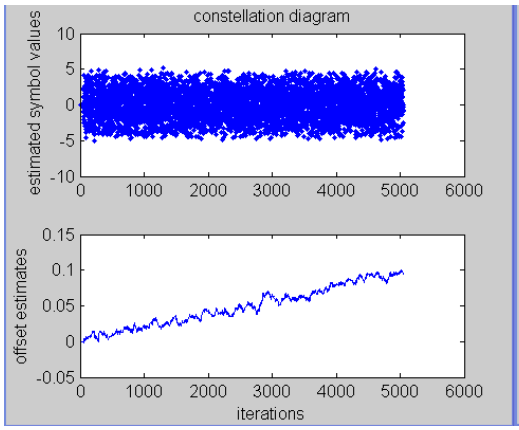
For the standard PLL approach, the "the sign of the derivative is preserved in the update (rather than its negative), indicating that the algorithm is searching for a maximum of the error surface rather than a minimum." [JSK, page 203]

### 6.2 Timing Recovery

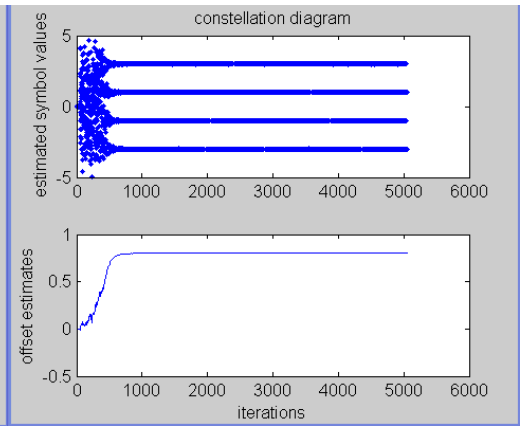
Johnson, Sethares and Klein, exercise 12.10, page 264.

Use the code in clockrecOP.m to "play with" the output power clock recovery algorithm. How does mu affect the convergence rate? What range of stepsizes works? How does the signal constellation of the input affect the convergent value of tau (try 4-PAM and 8-PAM)?

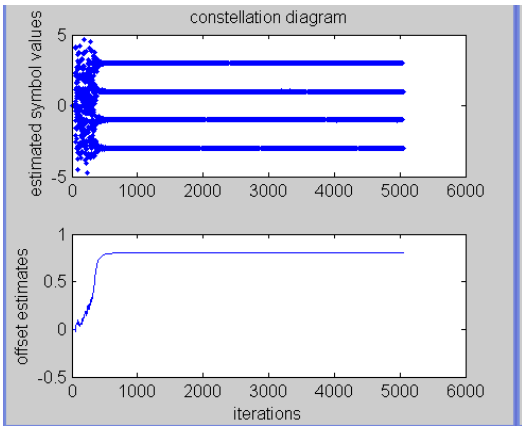
**Solution:** To test how the step size affects convergence, the timing offset was kept at -0.8, the rolloff (beta) was left at 0.3 and a 4-PAM signal of length 5000 symbols was generated. The following values of mu were tested: mu=[.003 .03 .04 .1] and the following plots were produced:



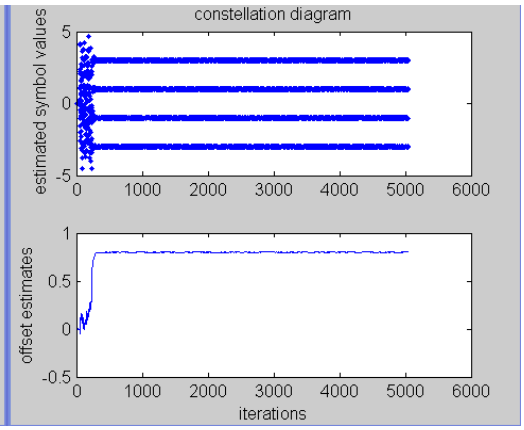

(a) Tau convergence for mu=.003




(b) Tau convergence for mu=.03



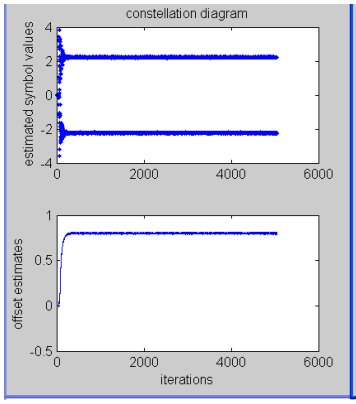

(c) Tau convergence for mu=.04



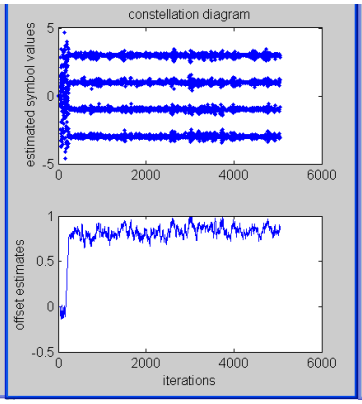

(d) Tau convergence for mu=.1

The value of mu is proportional to how quickly tau converges to the correct offset. If mu is too small (as in fig (a)) tau will not be able to reach the correct value within the given time period (in this case, 5000 symbols). As shown in fig (b), a small mu leads to a slower (but smoother) convergence. As mu is increased (fig (c)), tau will converge to the correct value more quickly. If mu grows too large, then the iterations may diverge.

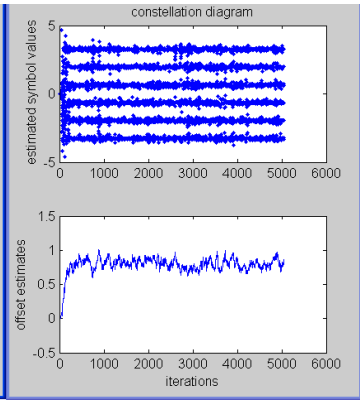
To analyze the effect of signal constellation of the input on the convergent value of tau, mu was set to 0.03 and the pam input signal was varied to test 2-PAM, 4-PAM and 6-PAM inputs. The PAM levels were adjusted so that the distance between levels was higher. I set 2-PAM to [-3,3], the 4-PAM to [-9,-3,3,9] and 6-PAM to [-15,-9,-3,3,9,15]. All of these appear to converge to the correct offset value. As the number of levels increases, the "path to offset settling" seems noisier.




(e) 2-PAM signal, mu=.03




(f) 4-PAM signal, mu=.03




(g) 6-PAM signal, mu=.03

**6.3 Simulation of 4-QAM and 16-QAM Transmission (JSK 16.2)**
Johnson, Sethares and Klein, exercise 16.2, page 264.

Use qamcompare.m as a basis to implement a 16-QAM modulation as shown in Figure 16.2. Compare this system to 4-QAM on the basis of

a. amount of spectrum used, and
b. required power in the transmitted signal.

What other trade-offs are made when moving from a 4-QAM to a 16-QAM modulation scheme?

**Solution:** In this problem, we compare 4-QAM modulation with 16-QAM modulation in terms of spectral use and required power in the transmitted signal.

Here, we implement 4-QAM as 2-PAM in the in-phase component and 2-PAM in the quadrature component. Likewise, we implement 16-QAM as 4-PAM in the in-phase component and 4-PAM in the quadrature component. We modify the Matlab script `qamcompare.m` for 4-QAM in Johnson, Sethares and Klein to simulate 16-QAM. The only line of code that changes is in bold:

```
N=1000; M=20; Ts=.0001;          % #symbols, oversampling factor
time=Ts*(N*M-1); t=0:Ts:time;    % sampling interval and time
s1=pam(N,4,1); s2=pam(N,4,1);    % length N real 2-level signals
ps=hamming(M);                   % pulse shape of width M
fc=1000; th=-1.0; j=sqrt(-1);    % carrier freq. and phase
s1up=zeros(1,N*M); s2up=zeros(1,N*M);
s1up(1:M:end)=s1;                % oversample by M
s2up(1:M:end)=s2;                % oversample by M
sp1=filter(ps,1,s1up);           % convolve pulse shape with s1
sp2=filter(ps,1,s2up);           % convolve pulse shape with s2
% make real and complex-valued versions and compare
vreal=sp1.*cos(2*pi*fc*t+th)-sp2.*sin(2*pi*fc*t+th);
vcomp=real((sp1+j*sp2).*exp(j*(2*pi*fc*t+th)));
max(abs(vcomp-vreal))            % verify that they're the same
```

(a) Digital QAM transmission is considered to be a baseband signal even though the transmitted spectrum is actually bandpass. A baseband signal has its energy concentrated near zero frequency (DC). The transmitted spectrum is centered at the carrier frequency, and the transmission bandwidth is equal to the symbol rate. (See lecture slide 16-5.) For our case, 4-QAM and 16-QAM transmitted signals use the same pulse shape, and hence, have the same transmission bandwidth.

(b) We will work out the average and peak transmit power in lecture. We use the parameter $d$ to represent half of the distance in the in-phase or quadrature component to the nearest neighbor in the constellation. Assuming that the energy in the pulse shape is one,

- 4-QAM has average transmit power of $2d^2$ and peak transmit power of $2d^2$
- 16-QAM has average transmit power of $10d^2$ and peak transmit power of $18d^2$

These power ratings increase with increasing number of levels in the QAM constellation if the value of $d$ is kept the same.

If we keep the average transmit power the same for 4-QAM and 16-QAM, then

$$10d_{16QAM}^2 = 2d_{4QAM}^2$$

or equivalently

$$d_{16QAM} = \frac{1}{\sqrt{5}} d_{4QAM} \approx 0.44 d_{4QAM}$$

The probability of symbol error increases when $d$ decreases.

(c) While larger constellations are able to offer much faster bit rates and higher levels of spectral efficiency, the larger constellations are considerably less resilient to noise and interference. While it is possible to transmit more bits per symbol, if the energy of the constellation is to remain the same, the points on the constellation must be closer together and the transmission

becomes more susceptible to noise. This results in a higher bit error rate than for the lower order QAM variants. In this way, there is a balance between obtaining the higher data rates and maintaining an acceptable bit error rate.

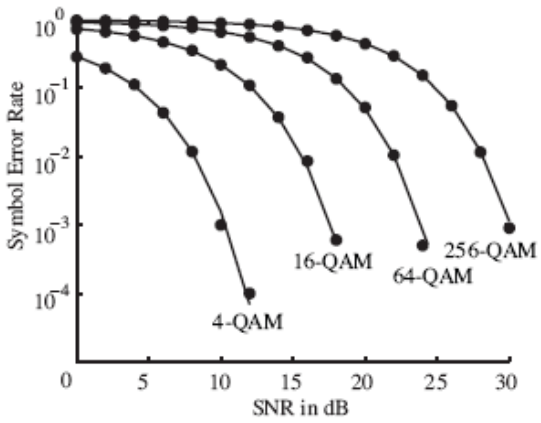



**Figure 16.12 from pg. 381**

Plotting symbol error rate vs. signal-to-noise ratio (SNR) is a very common first step in analyzing communication system performance. This curve plots the lower bound from a formula. The signal-to-noise ratio (SNR) is measured in the receiver at the output of the sampling block at the output of the matched filter.

One can interpret the plot in a couple of ways. First, pick a symbol error rate and take a horizontal cut across the plot. At a symbol error rate of $10^{-2}$, the formula for SNR as a function of the number of bits per symbol, B, is approximately 4 dB + (3 dB/bit) B. So, for every 3 dB of improvement in SNR, we could add a bit to the QAM constellation. Second, pick a SNR value and take a vertical cut through the plot. At an SNR of 10 dB, for example, the symbol error rate for 4-QAM is two orders of magnitude lower than that of 16-QAM.

Another way to use a plot of symbol error rate vs. SNR is to simulate a communication system for different SNR settings and scatter plot the results. This could allow comparison of two equalization methods, two timing recovery methods, etc. Superimposing the lower bound from a formula on the plot shows how close (or far away) the methods are from the ideal answer.

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.