Homework #6 Solutions

6.1 Phase Locked Loop (PLL)

Johnson, Sethares & Klein, exercise 10.21, on page 210. Oscillators that can adjust their phase in response to an input signal are more expensive than free-running oscillators. Figure 10.12 [on page 211] shows an alternative implementation of the Costas loop.

a. Show that this is actually carrying out the same calculations (albeit in a different order) as the implementation in Figure 10.9 [on page 208].
b. Write a simulation (or modify `costasloop.m`) to implement this alternative.

Prologue: In a receiver, the carrier phase can vary with time due to time-varying channel response and mismatch in the carrier frequency used in downconversion to the carrier frequency used in the transmitter for upconversion. Please see the epilogue for more info. Also please see the marker board notes on the setup for the steepest descent version of the Costas Loop.

Solution for part a: For the Costas Loop implementations, we're asked to show that the update equations for $\tau[k]$ are the same. In solving the problem, we'll need to use trigonometric identities

$$\sin(A+B) = \sin(A)\cos(B) + \cos(A)\sin(B)$$
$$\cos(A+B) = \cos(A)\cos(B) - \sin(A)\sin(B)$$

We'll need to assume that the phase $\theta[k]$ is slowly varying with time. This assumption implies that $\cos(\theta[k])$ is approximately constant for small values of $\theta[k]$, which allows the reordering of modulation by $\cos(\theta[k])$ and lowpass filtering in each branch. The reordering of modulation by $\sin(\theta[k])$ and lowpass filtering is performed similarly. It is a common assumption in PLLs that the phase being tracked is slowly varying with time (see the Epilogue for more info).

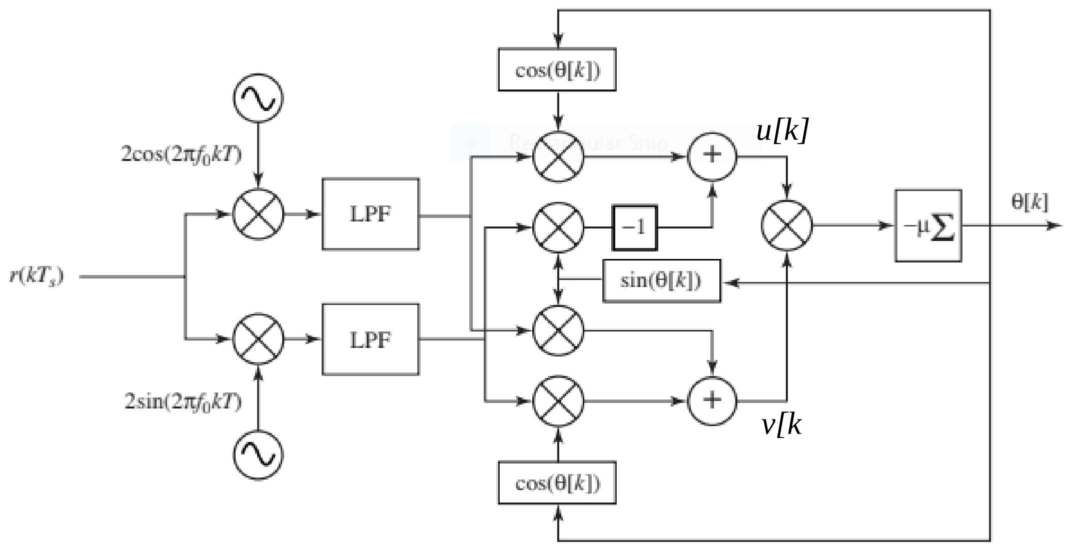


Figure 10.12 An alternative implementation of the Costas loop trades off less expensive oscillators for a more complex structure, as discussed in Exercise 10.21.

The sampling index is *k* and the sampling time is *T*s.

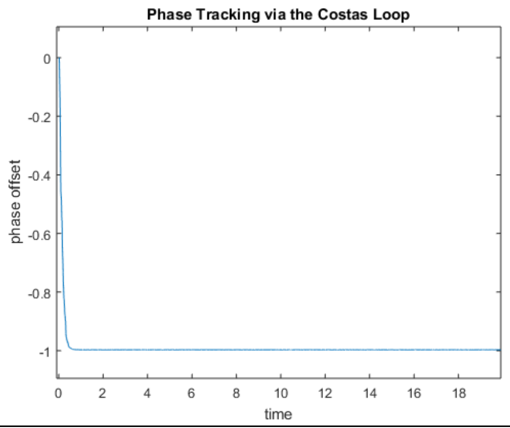
$\theta[k+1] = \theta[k] - \mu\, u[k]v[k]$ where $u[k]$ and $v[k]$ are the inputs fed to the final multiplier. Let $u[k]$ be the top input entering the multiplier and $v[k]$ be the bottom input entering the multiplier.

$$\begin{aligned}
u[k] &= \cos\left(\theta[k]\right) LPF\left\{2\ r[kT_s]\ \cos\left(2\pi f_0 kT_s\right)\right\} - \sin\left(\theta[k]\right)\ LPF\left\{2\ r[kT_s]\ \sin\left(2\pi f_0 kT_s\right)\right\} \\
&= LPF\left\{2\ r[kT_s]\cos\left(2\pi f_0 kT_s\right)\ \cos\left(\theta[k]\right)\right\} - LPF\left\{2\ r[kT_s]\ \sin\left(2\pi f_0 kT_s\right)\ \sin\left(\theta[k]\right)\right\} \\
&= LPF\left\{2\ r[kT_s]\ \left(\cos\left(2\pi f_0 kT_s\right)\cos\left(\theta[k]\right) - \sin\left(2\pi f_0 kT_s\right)\sin\left(\theta[k]\right)\right)\right\} \\
&= LPF\left\{2\ r[kT_s]\ \cos\left(2\pi f_0 kT_s + \theta[k]\right)\right\}
\end{aligned}$$

$$\begin{aligned}
v[k] &= \cos\left(\theta[k]\right) LPF\left\{2\ r[kT_s]\ \sin\left(2\pi f_0 kT_s\right)\right\} + \sin\left(\theta[k]\right) LPF\left\{2\ r[kT_s]\ \cos\left(2\pi f_0 kT_s\right)\right\} \\
&= LPF\left\{2\ r[kT_s]\sin\left(2\pi f_0 kT_s\right)\cos\left(\theta[k]\right)\right\} + LPF\left\{2\ r[kT_s]\cos\left(2\pi f_0 kT_s\right)\sin\left(\theta[k]\right)\right\} \\
&= LPF\left\{2\ r[kT_s]\ \left(\sin\left(2\pi f_0 kT_s\right)\cos\left(\theta[k]\right) + \cos\left(2\pi f_0 kT_s\right)\sin\left(\theta[k]\right)\right)\right\} \\
&= LPF\left\{2\ r[kT_s]\ \sin\left(2\pi f_0 kT_s + \theta[k]\right)\right\}
\end{aligned}$$

The above simplified expressions for $u[k]$ and $v[k]$ implement directly Figure 10.9. Both implementations perform the same mathematical operations for a slowly varying phase offset.

Solution for part b: The following Matlab code simulates the alternative Costas loop.

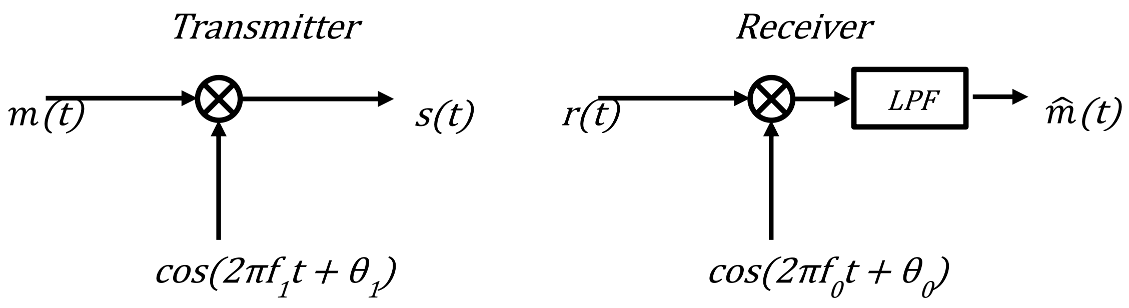


Initially, `phoff=-1.0` is set in `pulrecsig.m`. The alternative Costas Loop method tracks the phase successfully.

Figure 10.9 The Costas loop

```
% pllconverge.m simulate costas loop
% input rsc from pulrecsig.m

pulrecsig;
r=rsc;                                    % rsc is from pulrecsig.m
fl=500; ff=[0 .01 .02 1]; fa=[1 1 0 0];
h=remez(fl,ff,fa);                        % LPF design
mu=.003;                                   % algorithm stepsize
fc=1000;                                  % assumed freq. at receiver
theta=zeros(1,length(t)); theta(1)= 0;     % initialize estimate vector
zs=zeros(1,fl+1); zc=zeros(1,fl+1);       % initialize buffers for LPFs
for k=1:length(t)-1                       % z's contain past fl+1 inputs
  zs=[zs(2:fl+1), 2*r(k)*sin(2*pi*fc*t(k))];
  zc=[zc(2:fl+1), 2*r(k)*cos(2*pi*fc*t(k))];
  lpfs=fliplr(h)*zs'; lpfc=fliplr(h)*zc'; % new output of filters
  lpfs_c=lpfs*cos(theta(k));
  lpfs_s=-lpfs*sin(theta(k));
  lpfc_c=lpfc*cos(theta(k));
  lpfc_s=lpfc*sin(theta(k));
  update=(lpfc_c+lpfs_s)*(lpfc_s+lpfs_c);
  theta(k+1)=theta(k)-mu*update;          %algorithm update
end
plot(t,theta),
title('Phase Tracking via the Costas Loop')
xlabel('time'); ylabel('phase offset')
```

Epilogue: Suppose there is a mismatch between the carrier frequency, f_1 in the receiver and the carrier frequency, f_0 in the transmitter, we would show that the offset in carrier frequency, (f_1 - f_0) can be expressed as a form of phase offset. We can assume (f_1 - f_0) is very small but nonzero.



With an ideal channel assumption, we have,

$$r(t) = s(t) = m(t)\cos(2\pi f_1 t + \theta_1)$$

After downconversion in the receiver, we have,

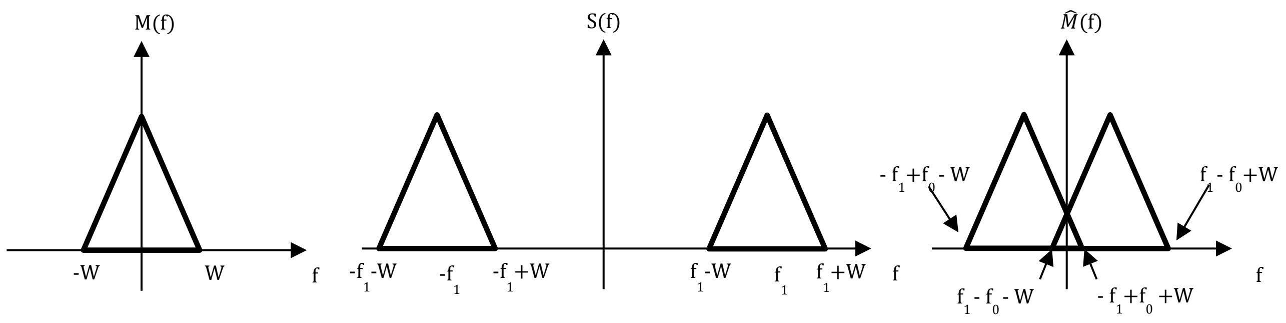
$$r(t)\cos(2\pi f_0 t + \theta_0) = m(t)\cos(2\pi f_1 t + \theta_1)\ \cos(2\pi f_0 t + \theta_0) =$$
$$\frac{1}{2}\, m(t)\,(\cos(2\pi f_1 t + \theta_1 + 2\pi f_0 t + \theta_0) + \cos(2\pi f_1 t + \theta_1 - 2\pi f_0 t - \theta_0))$$

using the identity cos(u) cos(v) = ½ cos(u - v) + ½ cos(u + v). Lowpass filtering will attenuate the first term and pass the second term in the parenthesis to give

$$\hat{m}(t) = \frac{1}{2}\, m(t)\cos(2\pi(f_1 - f_0)t + \theta_1 - \theta_0)$$

We define $\phi(t) = 2\pi(f_1 - f_0)t + \theta_1 - \theta_0$ as the time-varying phase offset. The time-varying phase offset $\phi(t)$ is a low-frequency signal with principal frequencies $-|f_1 - f_0|$, 0 and $|f_1 - f_0|$ where f_0 and f_1 are close in value. The maximum relative error between the two frequencies is often set by the communication standard. A maximum relative error of 10^{-3} would mean that if f_0 and f_1 were approximately 1 GHz, then $|f_1 - f_0| \leq 1$ MHz.

The spectrum of transmitted baseband signal, $M(f)$, passband signal, $S(f)$, and received baseband signal, $\hat{M}(f)$ are shown. Estimation for phase offset is still necessary to correct for the phase offset. A software approach can be used to correct for phase error; however, the cutoff frequency of the lowpass filter needs to have bandwidth of $W + |f_0 - f_1|$.



6.2 Timing Recovery

Johnson, Sethares & Klein, exercise 12.17, page 265. Consider a resampler with input $x(kT)$ and output $x(kT+\tau)$. Use the approximation in Exercise 12.16(b) to derive an approximate gradient-descent algorithm that minimizes the fourth-power performance function

$$J_{FP}(\tau) = \frac{1}{N} \sum_{k=k_0+1}^{k_0+N} x^4(kT+\tau)$$

for a suitably large N. [Note: Sampling time T_s has been replaced by the symbol time T.]

Prologue: In the receiver, the symbol timing offset, τ, is the offset to the start of the current symbol period. An offset of $\tau - T$ and $\tau + T$ would be at the start of the previous and next symbol period, respectively. During the kth symbol, the receiver estimates the symbol timing offset, and uses it to delay (if positive) or advance (if negative) the time that the (k+1)st symbol is sampled. It is okay if a symbol timing recovery algorithm converges to a timing offset of τ, $\tau - T$ or $\tau + T$.

Midterm #2 spring 2021 problem 2.1(d) applies steepest ascent for symbol timing recovery.

We should visualize objective function $J_{FP}(\tau)$ to see if we should maximize or minimize it. The objective function is an average of N fourth-power objective functions, and the concavity of each depends on the value of the rolloff parameter β for the square root raised cosine pulse shape, as plotted at the end of the solution to this problem. The concavity is down for β of 0.2 or 0.3 and up for β of 0.4, 0.5 or 1. We use $\beta = 0.2$ in this problem. We provide scatter plots for $J_{FP}(\tau)$ toward the end of the solution for this problem when the update for τ minimizes $J_{FP}(\tau)$.

A plot of the fourth-power objective from Fig. 2.11 on page 264 in JSK is shown over one symbol period. The objective function is **periodic** with period equal to the symbol time:

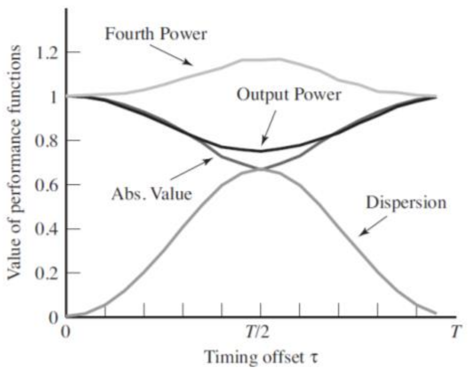


Figure 12.11 Four performance functions that can be used for timing recovery, plotted as a function of the timing offset τ. In this figure, the optimal answer is at $\tau = 0$. Some of the performance functions must be minimized and some must be maximized.

Solution for part a: The objective function sums fourth-order objective functions applied to the future *N* symbol periods and divides the result by *N*. In an implementation, we would buffer *N* symbol periods of samples. The implementation complexity increases with value of *N*.

$$J_{FP}(\tau) \approx avg\left\{\left(x(kT+\tau)\right)^4\right\}$$

The approximation in Exercise 12.16(b) is for the first derivative with respect to τ of the signal that results from sampling at the symbol time.

$$\frac{dx(kT+\tau)}{d\tau} = \lim_{\varepsilon\to 0}\frac{x(kT+\tau)-x(kT+\tau-\epsilon)}{\epsilon} \approx \frac{x(kT+\tau)-x(kT+\tau-\epsilon)}{\epsilon}$$

Differentiating $J_{FP}(\tau)$ with respect to τ and reordering differentiation and averaging operations gives,

$$\frac{dJ_{FP}(\tau)}{d\tau} \approx 4avg\left\{x^3(kT+\tau)\frac{dx(kT+\tau)}{d\tau}\right\}$$

The gradient descent update equation to minimize $J_{FP}(\tau)$ can written as follows:

$$\tau[k+1] = \tau[k] - \bar{\mu}\left.\frac{dJ_{FP}(\tau)}{d\tau}\right|_{\tau=\tau[k]}$$

$$\tau[k+1] \;=\; \tau[k] - 4\,\bar{\mu}\,avg\{x^3(kT+\tau[k])\}\left(\frac{x(kT+\tau[k])-x(kT+\tau[k]-\epsilon)}{\epsilon}\right)$$

$$\tau[k+1] = \tau[k] - \mu\sum_{k=k_0+1}^{k_0+N} x^3(kT+\tau[k])\left(x(kT+\tau[k])-x(kT+\tau[k]-\epsilon)\right)$$

where $\mu = \frac{4\bar{\mu}}{\epsilon N}$ has been combined into a single constant. From Exercise 12.16, we would like to have a small ε > 0 where ε << *T*. From the problem statement, we should use a suitably large *N*.

(b) Implement the algorithm in part (a) using `clockrecOP.m` as a basis and compare the behavior with the output power maximization approach in terms of convergence speed (i.e. number of iterations needed for the same value of the step size to find the symbol timing offset tau).

Solution for b: We modify `clockrecOP.m`, which uses symbol time of 1s and 2 samples/symbol. We use a timing offset of 0.5s, a rolloff (β) of 0.2, and 5000 random 2-PAM symbols. In the update equation, step size $\mu = \frac{4\bar{\mu}}{\epsilon N}$. We set ϵ to 0.01 and *N* to 40, and try $\bar{\mu}\ \epsilon\{.001, .01, .05, 5\}$.

```
clc; close all; clear all;
% clockrec4thpower.m:  clock recovery maximizing output power
% prepare transmitted signal
n=5000;                            % number of symbols
m=2;                               % oversampling factor (samples/symbol)
constel=2;                         % 2-pam constellation
beta=0.2;                          % rolloff parameter for srrc
l=50;                              % 1/2 length of pulse shape (in symbols)
chan=[1];                          % T/m "channel"
```

```
toffset= 0.5;                          % initial timing offset
% s=srrc(syms, beta, P, t_off);
% Generate a Square-Root Raised Cosine Pulse
pulshap=srrc(l,beta,m,toffset);        % srrc pulse shape
s=pam(n,constel,1);                    % random data sequence with var=1
sup=zeros(1,n*m);                      % upsample the data by placing...
sup(1:m:end)=s;                        % ... p zeros between each data point
hh=conv(pulshap,chan);                 % ... and pulse shape
r=conv(hh,sup);                        % ... to get received signal
matchfilt=srrc(l,beta,m,0);            % filter = pulse shape
x=conv(r,matchfilt);                   % convolve signal with matched filter

% run symbol clock recovery algorithm
epsilon = 0.01;
N = 40;
mubar = [0.001 0.01 0.05 5];
mu = 4*mubar/(epsilon*N);              % algorithm stepsize

for count=1:length(mu)
  tnow=l*m+1; tau=0; xs=zeros(1,n);             % initialize variables
  tausave=zeros(1,n); tausave(1)=tau; i=0;
  JFP=zeros(1,n);
  epsilon=0.01;
  tmax=length(x) - l*m - (N+1)*m;               % max time for simulation
  while tnow < tmax                             % run iterations
    i=i+1;
    xs(i)=interpsinc(x,tnow+tau,l);             % interp at tnow+tau
    % Update equation for tau
    dJFP_dtau = 0;                              % dJ_FP(tau) / d(tau)
    for k0 = 1 : N
      x_deltap = interpsinc(x,tnow+k0*m+tau,l);          % value to right
      x_deltam = interpsinc(x,tnow+k0*m+tau-epsilon,l);  % value to left
      dx = x_deltap - x_deltam;                      % approximate dx/dtau
      dJFP_dtau = dJFP_dtau + (x_deltap^3)*dx;       % derivative
    end
    tau = tau - mu(count) * dJFP_dtau;               % update tau
    tnow=tnow+m; tausave(i)=tau;                     % save for plotting
  end

  % plot results
  figure(count);
  subplot(2,1,1), plot(xs(1:i-9),'b.')     % plot constellation diagram
  grid on;
  title_str = ['constellation diagram, mu = ' num2str(mu(count))];
  title(title_str);
  ylabel('estimated symbol values')
  subplot(2,1,2), plot(tausave(1:i-9))      % plot trajectory of tau
  grid on;
  ylabel('offset estimates'), xlabel('iterations')
end
```

In Figures A, B, and C, the rate of convergence is faster for larger values of μ. If μ is too large, iterations may diverge as shown in Figure D. In all Figures, τ converges to -0.5s instead of 0.5s; however, both offsets led to the start of a symbol because the symbol time is 1s.

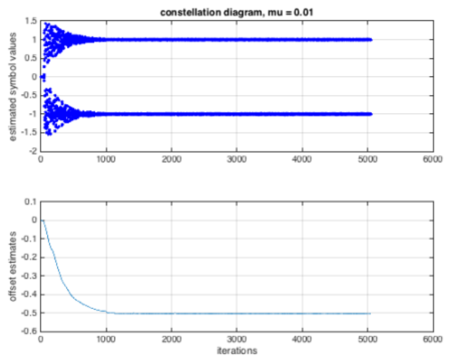


Figure A: μ = 0.01

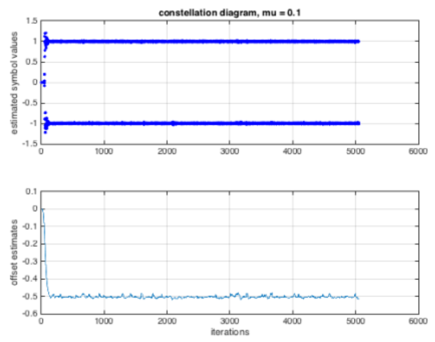


Figure B: μ = 0.1

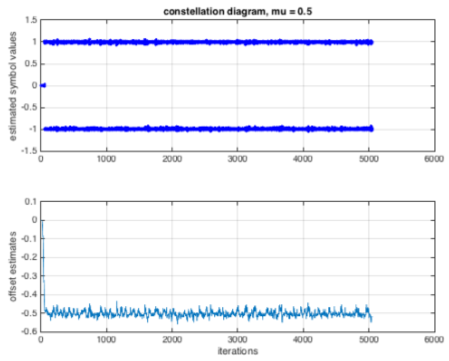


Figure C: μ = 0.5

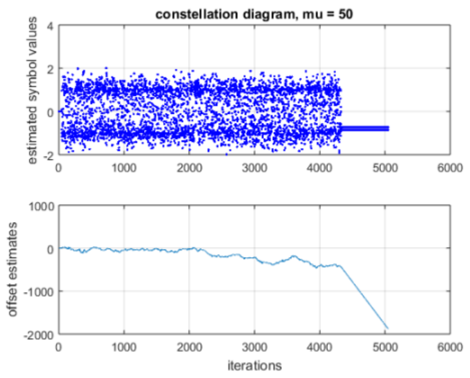
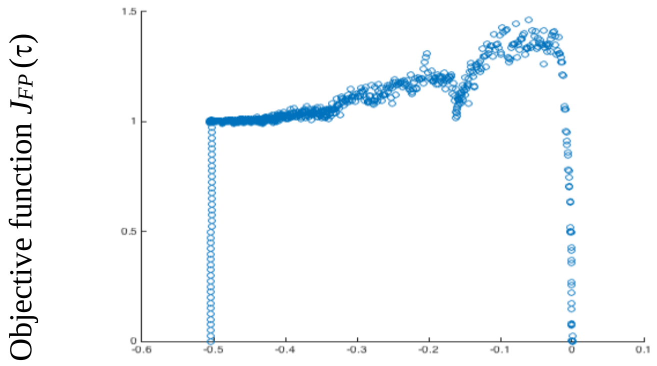


Figure D: μ = 50

We create a scatter plot of the values of τ computed during each simulation vs. the value of the objective function $J_{FP}(\tau)$ for each τ by changing the while loop in the above code as follows:

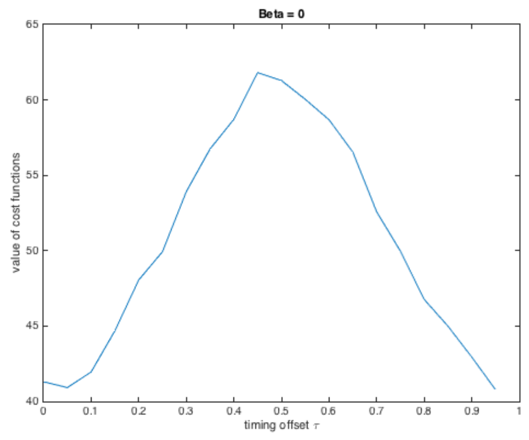
```
while tnow < tmax                            % run iterations
  i=i+1;
  xs(i)=interpsinc(x,tnow+tau,l);            % interp at tnow+tau
  % Update equation for tau
  dJFP_dtau = 0;                             % dJ_FP(tau) / d(tau)
  objfun = 0;
  for k0 = 1 : N
    x_deltap = interpsinc(x,tnow+k0*m+tau,l);          % value to right
    x_deltam = interpsinc(x,tnow+k0*m+tau-epsilon,l);  % value to left
    dx = x_deltap - x_deltam;                      % approximate dx/dtau
    dJFP_dtau = dJFP_dtau + (x_deltap^3)*dx;       % derivative
    objfun = objfun + (x_deltap^4);
  end
  JFP(i) = objfun / N;
  tau = tau - mu(count) * dJFP_dtau;               % update tau
  tnow=tnow+m; tausave(i)=tau;                     % save for plotting
end
figure(count + length(mu));
scatter(tausave, JFP);
```

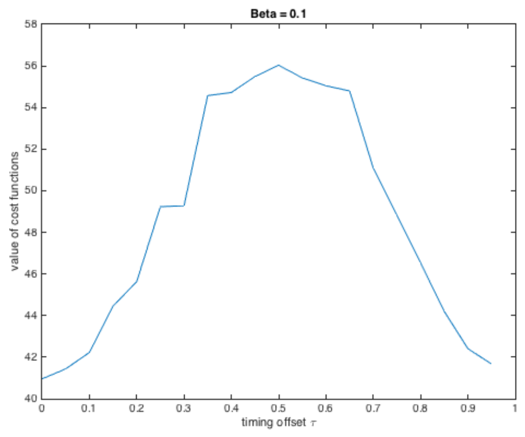
The scatter plots of objective function $J_{FP}(\tau)$ vs. symbol timing offset τ look very similar, so we only give the plot for μ = 0.01. The initial guess of τ is zero, and the final value is -0.5s.

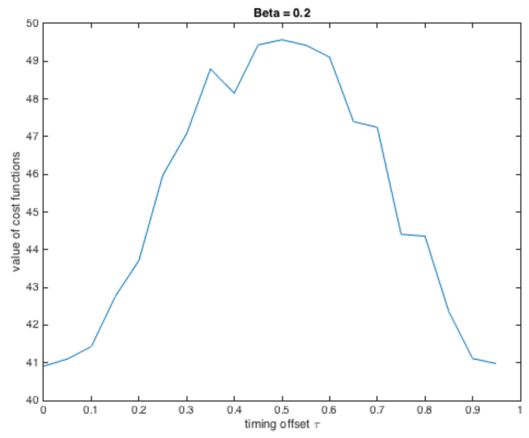


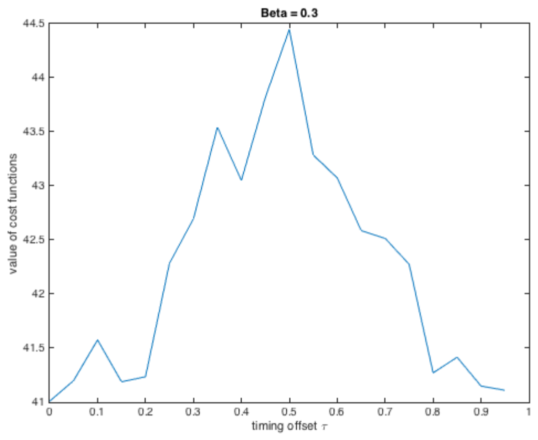
Below, the fourth-order power function is plotted as a function of the timing offset τ to visualize the action of the adaptive element. The error surface is plotted for different roll-off factors, β = 0, 0.1, 0.2, 0.3, 0.4, and 0.5. As β increases from 0.1 to 0.4, the surface gains more (local) minima. These plots represent the values of τ to which the adaptive element may converge, which may yield lower performance.

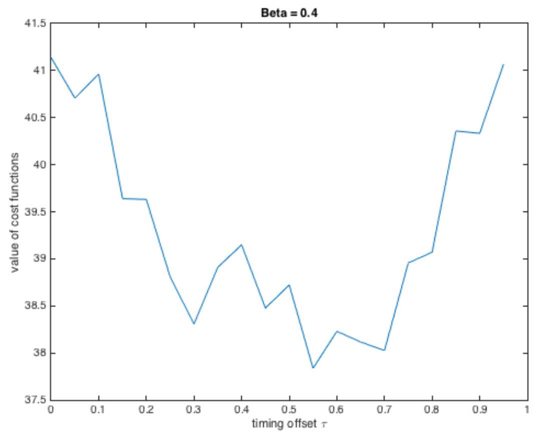
```
% clockrecDDcost.m: draw error surfaces for cluster variance performance
% in terms of chosen pulse shape
clc; close all; clear all;
l=10;                           % 1/2 duration of pulse shape
beta= [0 0.1 0.2 0.3 0.4 0.5 1]; % rolloff for pulse shape
m=20;                           % evaluate at m different points
for count=1:length(beta)
    ps=srrc(l,beta(count),m);            % make srrc pulse shape
    psrc=conv(ps,ps);            % convolve 2 srrc's to get rc
    psrc=psrc(l*m+1:3*l*m+1);    % truncate to same length as ps
    cost=zeros(1,m); n=20000;    % experimental performance
    x=zeros(1,n);
    for i=1:m                    % for each offset
      pt=psrc(i:m:end);          % rc is shifted i/m of a symbol
      for k=1:n                  % do it n times
        rd=pam(length(pt),4,5);  % random 4-PAM vector
        x(k)=sum(rd.*pt);        % received data point w/ ISI
      end
      cost(i)=sum(x.^4)/length(x);   % performance function
    end

    offset=(0:m-1)/m;
    figure(count);
    plot(offset,cost)
    title_str = ['Beta = ' num2str(beta(count))];
    title(title_str);
    xlabel('timing offset \tau')
    ylabel('value of cost functions')
end
```

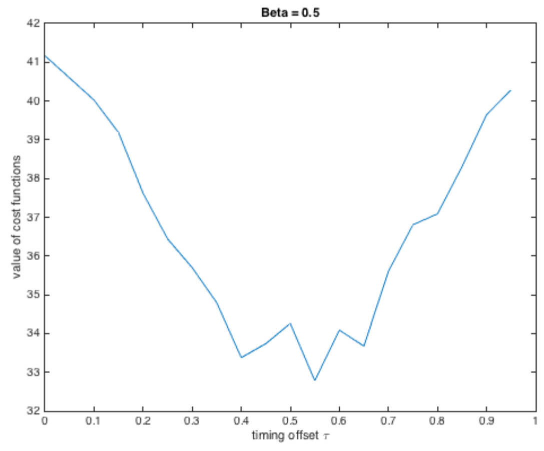












6.3 Simulation of 4-QAM and 16-QAM Transmission

Johnson, Sethares & Klein, exercise 16.2, page 362. 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) Even though a discrete-time QAM signal is bandpass, it is considered baseband because of the low carrier frequency relative to the carrier frequency used for continuous-time transmission, such as an RF frequency. Recall that a baseband signal has its energy concentrated near zero frequency (DC). The baseband PAM bandwidth is f_{sym} in lecture slide 7-10 for the rectangular pulse and $\frac{1}{2}(1+\beta)f_{\text{sym}}$ in lecture slide 7-13 for the raised cosine by replacing f_s with f_{sym}. For any carrier frequency greater than or equal to f_{sym} for the rectangular pulse shape and $\frac{1}{2}(1+\beta)f_{\text{sym}}$ for the raised cosine pulse shape, the QAM baseband bandwidth doubles to 2 f_{sym} for the rectangular pulse and $(1 + \beta)$ f_{sym} for the raised cosine pulse shape. Since 4-QAM and 16-QAM use the same pulse shape, transmission bandwidth would be the same.

(b) The average, peak and peak-to-average ratio for transmit power is on lecture slide 15-16. **A higher peak-to-average ratio makes the power amplifier more expensive to fabricate.** Parameter *d* represents half of the distance in the in-phase or quadrature component to the nearest neighbor in the constellation. Assuming energy in the pulse shape is one,

- 4-QAM has $2d^2$ average, $2d^2$ peak, and 1.0 peak-to-average ratio in transmit power
- 16-QAM has $10d^2$ average, $18d^2$ peak, and 1.8 peak-to-average ratio in transmit power

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

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 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 smaller QAM constellations. This is a tradeoff between obtaining higher data rates and meeting bit error rate requirements.

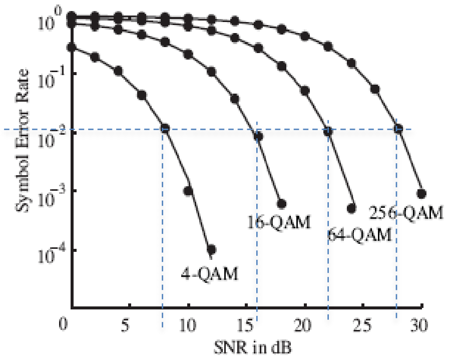


Figure 16.12 from pg. 379

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

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 approximately 10^{-2}, which occurs in Wi-Fi and cellular communication channels, the formula for SNR as a function of the number of bits per symbol, *B*, is approximately $\boldsymbol{C_0}$ **+ (3 dB/bit)** $\boldsymbol{B}$ where C_0 is a constant. (For 16-QAM, 64-QAM, and 256-QAM, $C_0 \approx 4$ dB, and for 4-QAM, $C_0 \approx 2$ dB.) For every 3 dB of improvement in SNR above C_0, we could add a bit to the QAM constellation and achieve the same symbol error rate. We can use this information to modify the channel capacity bound given by Claude Shannon for QAM

$$C = W \log_2(1 + \mathrm{SNR})$$

to give a bound on the achievable channel capacity of

$$C = W \log_2\left(1 + \frac{\mathrm{SNR}}{\Gamma}\right)$$

Here, **SNR is in linear units** and *W* is the transmission bandwidth in Hz, or equivalently the QAM baseband bandwidth. **Γ is the SNR gap converted to linear units**, which captures impairments other than the Gaussian model of thermal noise.

Second, pick a SNR value and take a vertical cut through the plot. At an SNR of 10 dB, 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.

Epilog. In probability of symbol error analysis, there are two extreme points to keep in mind:

Signal power = 0 in linear units. SNR = 0 in linear units, which is -∞ dB since the conversion is SNR_{dB} = 10 $\log_{10}$ SNR. The receiver can only randomly guess at the symbol of bits. For *J* bits, the number of levels is $M = 2^J$. The probability of error for a random guess is $(M\text{-}1)/M$.

Noise power = 0 in linear units. SNR = +∞, which is also +∞ dB. Probability of error is 0.

It is a common approach to plot the probability of symbol error on a log-log scale. The vertical axis would be the Probability of Symbol Error with marks at 10^0, 10^{-1}, 10^{-2}, etc. The horizontal axis would be SNR in dB. In this log-log scale, the curves for the Probability of Symbol Error vs. SNR for different constellation sizes (i.e. number of levels *M*) would not intersect.

PAM symbol error probability formula has the form $C_0\, Q\left(C_1 \sqrt{SNR}\right)$ where C_0 and C_1 are constants. $Q(x)$ is defined on lecture slide 14-22 and decays faster than a decaying exponential (lecture slide 14-23). QAM symbol error probability formula has the form $C_2\, Q\left(C_3 \sqrt{SNR}\right) - C_4\, Q^2\left(C_5 \sqrt{SNR}\right)$ where C_2, C_3, C_4 and C_5 are constants (lecture slide 15-15). In both formulas, SNR is in linear units.

When signal power and noise power are equal, the SNR in linear units is 1, or 0 dB.

In cellular LTE systems, the channel quality indicator (CQI) is a four-bit unsigned number that indicates the received SNR in dB, with 0 meaning the connection is out of range. The remaining numbers 1-15 indicate increasing SNR, where the mapping depends on LTE system settings. For example, the mapping below on the left in purple is CQI times 2 dB. The CQI is used to choose a QAM constellation size as shown below on the right.

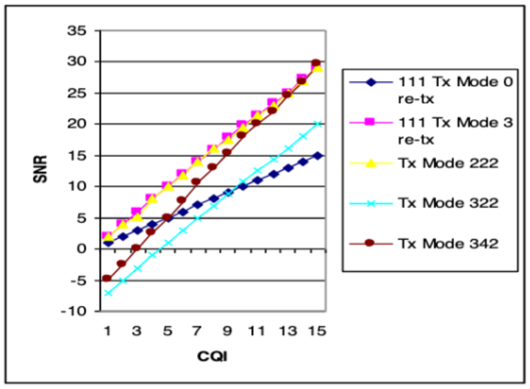


Fig. 6. SNR vs CQI for different Tx modes

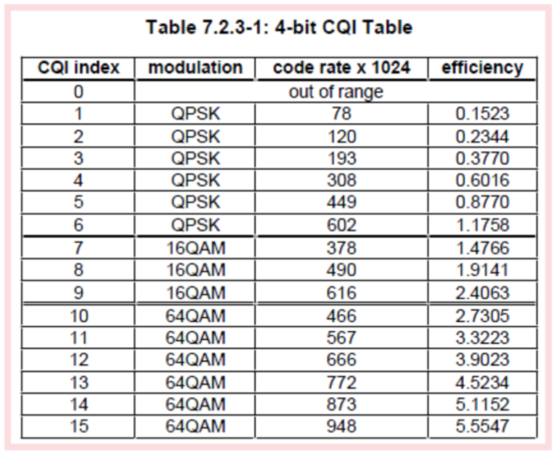


Table 7.2.3-1: 4-bit CQI Table

CQI index	modulation	code rate x 1024	efficiency
0	out of range		
1	QPSK	78	0.1523
2	QPSK	120	0.2344
3	QPSK	193	0.3770
4	QPSK	308	0.6016
5	QPSK	449	0.8770
6	QPSK	602	1.1758
7	16QAM	378	1.4766
8	16QAM	490	1.9141
9	16QAM	616	2.4063
10	64QAM	466	2.7305
11	64QAM	567	3.3223
12	64QAM	666	3.9023
13	64QAM	772	4.5234
14	64QAM	873	5.1152
15	64QAM	948	5.5547

http://www.ijiee.org/papers/201-X2020.pdf

https://ytd2525.wordpress.com/2014/02/02/cqi-channel-quality-indicator/