Principles of Communication - HW 3 - Martyn Staalsen

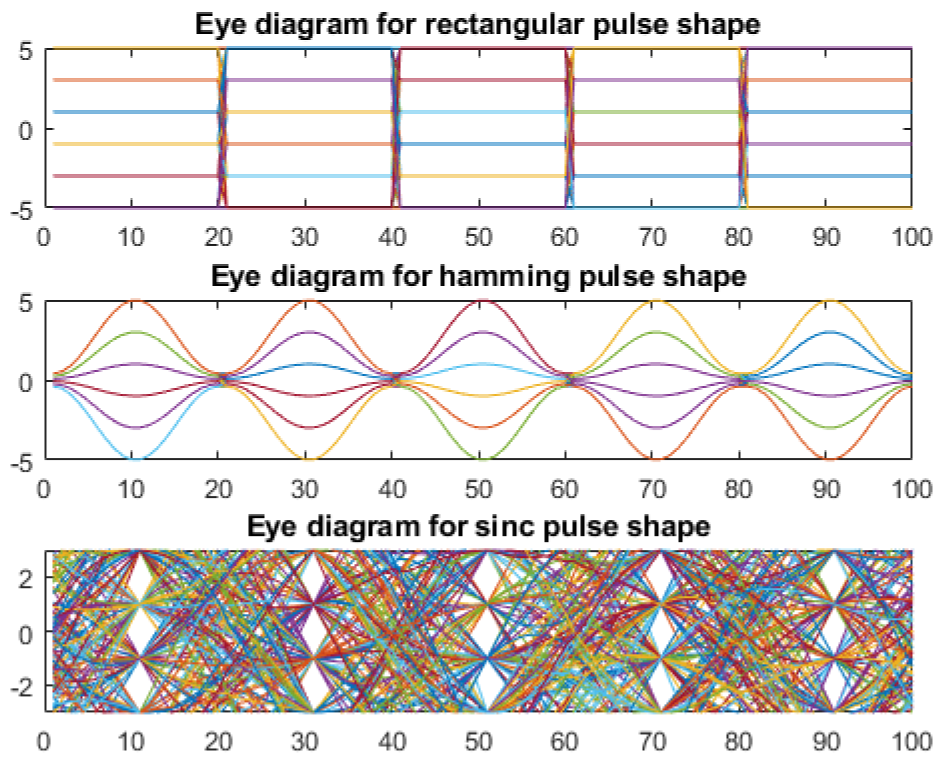
Table of Contents

11.3

```
figure()                             % used to plot figure eyediag3
N=1000; m=pam(N,6,35/3);             % random +/-1 signal of length N
M=20; mup=zeros(1,N*M); mup(1:M:N*M)=m;  % oversampling by factor of M
ps=ones(1,M);                            % square pulse width M
x=filter(ps,1,mup);              % convolve pulse shape with mup
neye=5;
c=floor(length(x)/(neye*M))
xp=x(N*M-neye*M*c+1:N*M);        % dont plot transients at start
q=reshape(xp,neye*M,c);          % plot in clusters of size
 5*Mt=(1:198)/50+1;
subplot(3,1,1), plot(q)
title('Eye diagram for rectangular pulse shape')

N=1000; m=pam(N,6,35/3);             % random +/-1 signal of length N
M=20; mup=zeros(1,N*M); mup(1:M:N*M)=m;  % oversampling by factor of M
ps=hamming(M);                           % square pulse width M
x=filter(ps,1,mup);              % convolve pulse shape with mup
%x=x+0.15*randn(size(x));
neye=5;
c=floor(length(x)/(neye*M))
xp=x(N*M-neye*M*c+1:N*M);        % dont plot transients at start
q=reshape(xp,neye*M,c);          % plot in clusters of size
 5*Mt=(1:198)/50+1;
subplot(3,1,2), plot(q)
title('Eye diagram for hamming pulse shape')

N=1000; m=pam(N,6,35/3);             % random +/-1 signal of length N
M=20; mup=zeros(1,N*M); mup(1:M:N*M)=m;  % oversampling by factor of M
L=10; ps=srrc(L,0,M,50);
ps=ps/max(ps);          % sinc pulse shape L symbols wide
x=filter(ps,1,mup);     % convolve pulse shape with mup
```

```
%x=x+0.15*randn(size(x));
neye=5;
c=floor(length(x)/(neye*M))
xp=x(N*M-neye*M*(c-3)+1:N*M);  % dont plot transients at start
q=reshape(xp,neye*M,c-3);      % plot in clusters of size
 5*Mt=(1:198)/50+1;
subplot(3,1,3), plot(q)
axis([0,100,-3,3])
title('Eye diagram for sinc pulse shape')
```

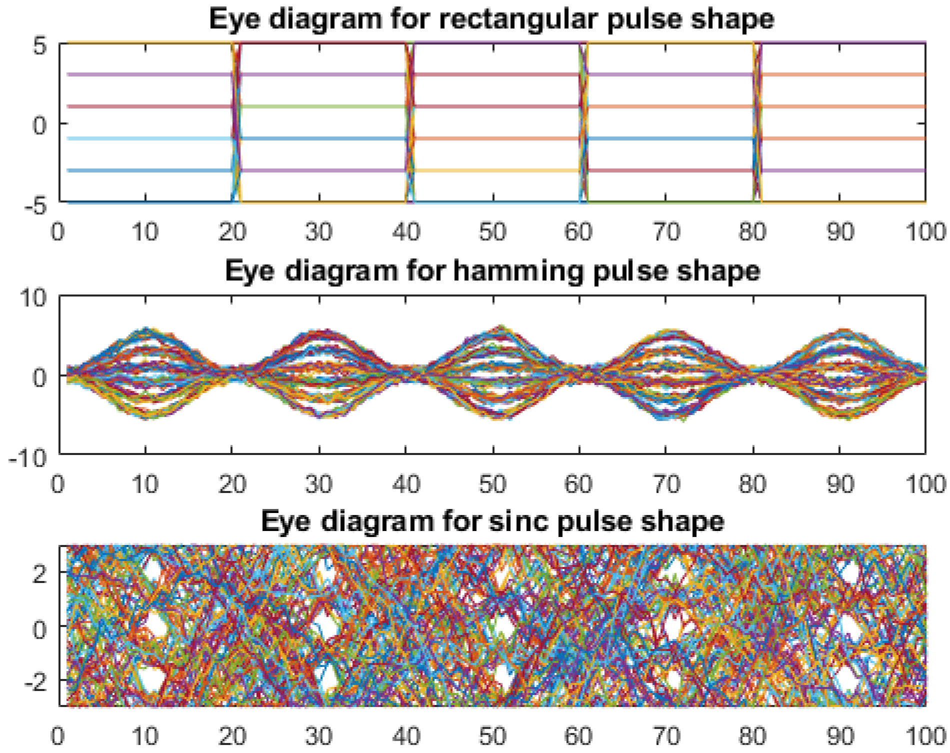
```
c =

   200


c =

   200


c =

   200
```



11.4

```
v = 0.33; %% any higher value of v seems to "close" the eye enough
 that errors should start occuring.


figure()                      % used to plot figure eyediag3
N=1000; m=pam(N,6,35/3);          % random +/-1 signal of length N
M=20; mup=zeros(1,N*M); mup(1:M:N*M)=m;  % oversampling by factor of M
ps=ones(1,M);                            % square pulse width M
x=filter(ps,1,mup);           % convolve pulse shape with mup
neye=5;
c=floor(length(x)/(neye*M))
xp=x(N*M-neye*M*c+1:N*M);     % dont plot transients at start
q=reshape(xp,neye*M,c);       % plot in clusters of size
 5*Mt=(1:198)/50+1;
subplot(3,1,1), plot(q)
title('Eye diagram for rectangular pulse shape')

N=1000; m=pam(N,6,35/3);          % random +/-1 signal of length N
M=20; mup=zeros(1,N*M); mup(1:M:N*M)=m;  % oversampling by factor of M
ps=hamming(M);                           % square pulse width M
x=filter(ps,1,mup);           % convolve pulse shape with mup
x=x+v*randn(size(x));
neye=5;
c=floor(length(x)/(neye*M))
xp=x(N*M-neye*M*c+1:N*M);     % dont plot transients at start
q=reshape(xp,neye*M,c);       % plot in clusters of size
 5*Mt=(1:198)/50+1;
subplot(3,1,2), plot(q)
title('Eye diagram for hamming pulse shape')

N=1000; m=pam(N,6,35/3);          % random +/-1 signal of length N
M=20; mup=zeros(1,N*M); mup(1:M:N*M)=m;  % oversampling by factor of M
L=10; ps=srrc(L,0,M,50);
ps=ps/max(ps);          % sinc pulse shape L symbols wide
x=filter(ps,1,mup);     % convolve pulse shape with mup
x=x+v*randn(size(x));
neye=5;
c=floor(length(x)/(neye*M))
xp=x(N*M-neye*M*(c-3)+1:N*M);  % dont plot transients at start
q=reshape(xp,neye*M,c-3);     % plot in clusters of size
 5*Mt=(1:198)/50+1;
subplot(3,1,3), plot(q)
axis([0,100,-3,3])
title('Eye diagram for sinc pulse shape')


c =

   200
```

```
c =

   200


c =

   200
```



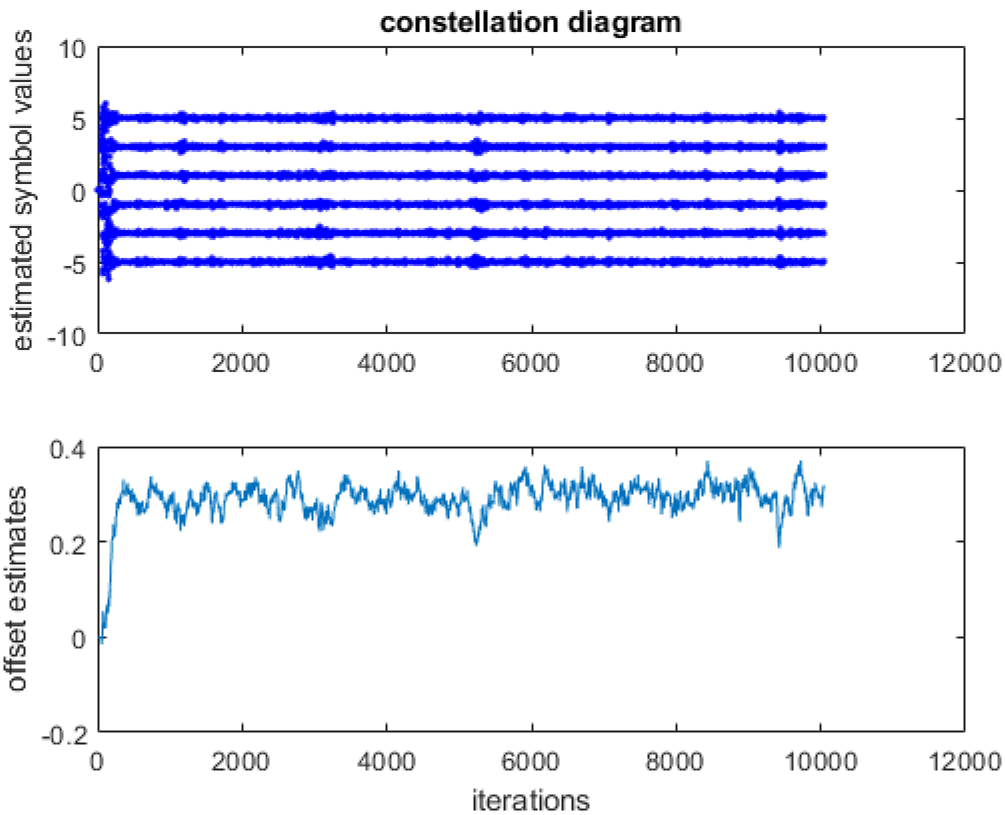
12.1

Use clockrecDD.m to “play with” the clock-recovery algorithm. a. How does mu affect the convergence rate? What range of stepsizes works? b. How does the signal constellation of the input affect the convergent value of tau? (Try 2-PAM and 6-PAM. Remember to quantize properly in the algorithm update.)

```
% clockrecDD.m: clock recovery minimizing 4-PAM cluster variance
% to minimize J(tau) = (Q(x(kT/M+tau))-x(kT/M+tau))^2

% prepare transmitted signal
n=10000;                         % number of data points
m=2;                             % oversampling factor
beta=0.3;                        % rolloff parameter for srrc
l=50;                            % 1/2 length of pulse shape (in
 symbols)
chan=[1];                        % T/m "channel"
```

```
toffset=-0.3;                     % initial timing offset
pulshap=srrc(l,beta,m,toffset);  % srrc pulse shape with timing offset
%s=pam(n,4,5);                     % random data sequence with var=5
%s=pam(n,2,1);
s=pam(n,6,35/3);
sup=zeros(1,n*m);                 % upsample the data by placing...
sup(1:m:n*m)=s;                   % ... m-1 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);       % matched filter = srrc pulse shape
x=conv(r,matchfilt);              % convolve signal with matched filter

% clock recovery algorithm
tnow=l*m+1; tau=0; xs=zeros(1,n);   % initialize variables
tausave=zeros(1,n); tausave(1)=tau; i=0;
mu=0.01;                            % algorithm stepsize
delta=0.1;                          % time for derivative
while tnow<length(x)-2*l*m          % run iteration
  i=i+1;
  xs(i)=interpsinc(x,tnow+tau,l);   % interp value at tnow+tau
  x_deltap=interpsinc(x,tnow+tau+delta,l); % value to right
  x_deltam=interpsinc(x,tnow+tau-delta,l); % value to left
  dx=x_deltap-x_deltam;             % numerical derivative
  qx=quantalph(xs(i),[-5,-3-1,1,3,5]);  % quantize to alphabet
  tau=tau+mu*dx*(qx-xs(i));         % alg update: DD
  tnow=tnow+m; tausave(i)=tau;      % save for plotting
end

% plot results
figure()
subplot(2,1,1), plot(xs(1:i-2),'b.')        % plot constellation
 diagram
title('constellation diagram');
ylabel('estimated symbol values')
subplot(2,1,2), plot(tausave(1:i-2))        % plot trajectory of tau
ylabel('offset estimates'), xlabel('iterations')

% a)
% reducing the value of mu will make the timing recovery algorithm
 converge
% more slowly, which results in more errors as while the timing is not
 yet
% correct. Increasing the value of mu makes the algorithm converge
 more
% quickly, but increasing it too much results in instability which can
% potentially cause errors in classification

% b)
% Changing the signal constellation to 2PAM makes the system converge
 more
% slowly, having similar behavior as reducing the value of mu. This
 can be
% compensated for by increasing mu. On the other
```

```
% hand, changing the constellation to 6PAM allows the system to
 converge
% reasonably quickly, but also results in significant instability,
 much
% as increasing the value of mu does. However, this instability seems
 to be
% inherant to the larger constellation size, not being able to be
% completely eliminated by lowering the value of mu.
```

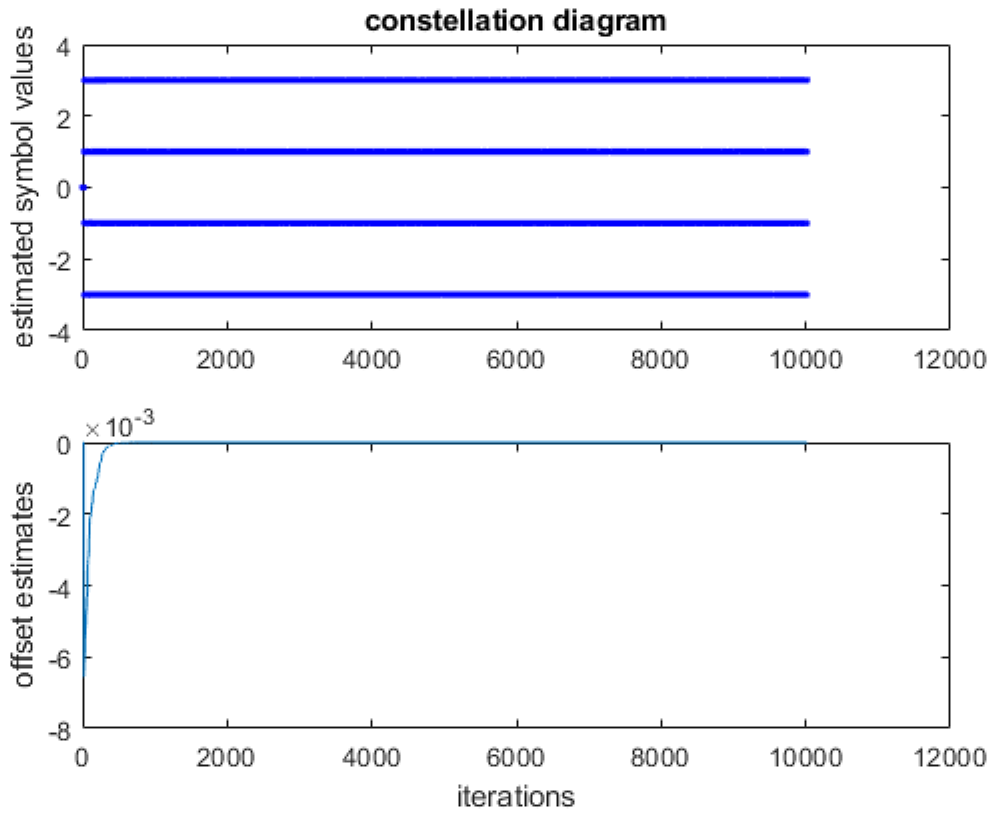


12.2

Implement a rectangular pulse shape. Does this work better or worse than the SRRC?

```
% clockrecDD.m: clock recovery minimizing 4-PAM cluster variance
% to minimize J(tau) = (Q(x(kT/M+tau))-x(kT/M+tau))^2

% prepare transmitted signal
n=10000;                          % number of data points
m=5;                              % oversampling factor
beta=0.3;                         % rolloff parameter for srrc
l=20;                             % 1/2 length of pulse shape (in
 symbols)
chan=[1];                         % T/m "channel"
toffset=-0.3;                     % initial timing offset
ssrc_pulshap=srrc(l,beta,m,toffset);  % srrc pulse shape with timing
 offset
```

```
w = 1;%width of rectangular pulse
rect_pulshap = rectPulse(l,m,-toffset);
pulshap=ssrc_pulshap;
pulshap=rect_pulshap;
s=pam(n,4,5);                   % random data sequence with var=5
sup=zeros(1,n*m);               % upsample the data by placing...
sup(1:m:n*m)=s;                 % ... m-1 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);     % matched filter = srrc pulse shape
matchfilt=rectPulse(l,m,0);
x=conv(r,matchfilt);            % convolve signal with matched filter

% clock recovery algorithm
tnow=l*m+1; tau=0; xs=zeros(1,n);   % initialize variables
tausave=zeros(1,n); tausave(1)=tau; i=0;
mu=0.1;                            % algorithm stepsize
delta=0.1;                          % time for derivative
while tnow<length(x)-2*l*m          % run iteration
  i=i+1;
  xs(i)=interpsinc(x,tnow+tau,l);   % interp value at tnow+tau
  x_deltap=interpsinc(x,tnow+tau+delta,l); % value to right
  x_deltam=interpsinc(x,tnow+tau-delta,l); % value to left
  dx=x_deltap-x_deltam;             % numerical derivative
  qx=quantalph(xs(i),[-3,-1,1,3]);  % quantize to alphabet
  tau=tau+mu*dx*(qx-xs(i));         % alg update: DD
  tnow=tnow+m; tausave(i)=tau;      % save for plotting
end
figure();
% plot results
subplot(2,1,1), plot(xs(1:i-2),'b.')        % plot constellation
 diagram
title('constellation diagram');
ylabel('estimated symbol values')
subplot(2,1,2), plot(tausave(1:i-2))        % plot trajectory of tau
ylabel('offset estimates'), xlabel('iterations')

% It appears that the rectangular pulse is problematic for the timing
% recovery algorithm. This could be because the rectangular pulse's
% frequency spectrum is too ambiguous for the algorithm to easily
 identify
% where the main pulse is centered, or it could be
% because I've implemented it poorly
```

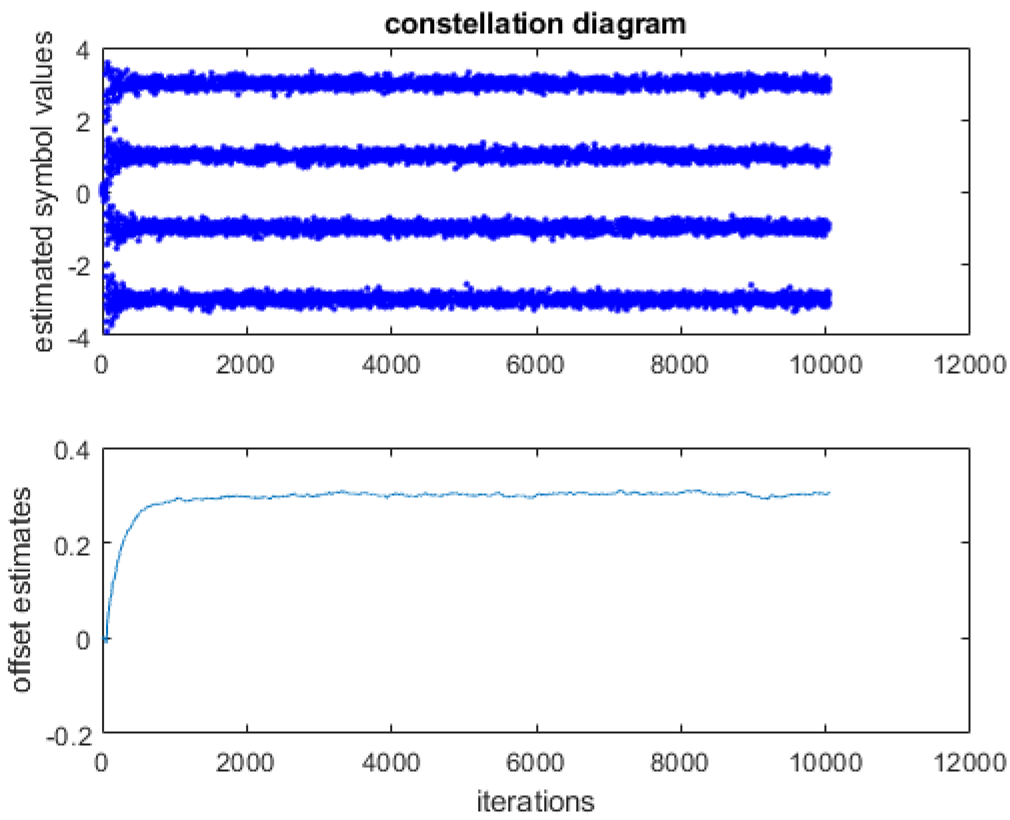


12.3

Add noise to the signal (add a zero-mean noise to the received signal using the Matlab randn function). How does this affect the convergence of the timing-offset parameter tau. Does it change the final converged value?

```
v=0.1;

% prepare transmitted signal
n=10000;                          % number of data points
m=2;                              % oversampling factor
beta=0.3;                         % rolloff parameter for srrc
l=50;                             % 1/2 length of pulse shape (in
 symbols)
chan=[1];                         % T/m "channel"
toffset=-0.3;                     % initial timing offset
pulshap=srrc(l,beta,m,toffset);   % srrc pulse shape with timing offset
s=pam(n,4,5);                     % random data sequence with var=5
sup=zeros(1,n*m);                 % upsample the data by placing...
sup(1:m:n*m)=s;                   % ... m-1 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);       % matched filter = srrc pulse shape
x=conv(r,matchfilt);              % convolve signal with matched filter
```

```
x=x+v*randn(size(x));

% clock recovery algorithm
tnow=l*m+1; tau=0; xs=zeros(1,n);   % initialize variables
tausave=zeros(1,n); tausave(1)=tau; i=0;
mu=0.01;                            % algorithm stepsize
delta=0.1;                          % time for derivative
while tnow<length(x)-2*l*m          % run iteration
  i=i+1;
  xs(i)=interpsinc(x,tnow+tau,l);   % interp value at tnow+tau
  x_deltap=interpsinc(x,tnow+tau+delta,l); % value to right
  x_deltam=interpsinc(x,tnow+tau-delta,l); % value to left
  dx=x_deltap-x_deltam;             % numerical derivative
  qx=quantalph(xs(i),[-3,-1,1,3]);  % quantize to alphabet
  tau=tau+mu*dx*(qx-xs(i));         % alg update: DD
  tnow=tnow+m; tausave(i)=tau;      % save for plotting
end

% plot results
figure();
subplot(2,1,1), plot(xs(1:i-2),'b.')        % plot constellation
 diagram
title('constellation diagram');
ylabel('estimated symbol values')
subplot(2,1,2), plot(tausave(1:i-2))        % plot trajectory of tau
ylabel('offset estimates'), xlabel('iterations')

% It appears that the presence of noise does not affect convergence
 speed
% for the timing recovery algorithm, but instead affects the stability
 with
% which the algorithm maitains its timing estimate. Specifically, a
 large
% amount of noise makes the offset estimate tau vary also noisy, which
% could contribute to further errors in categorizing recovered data
 points.
```



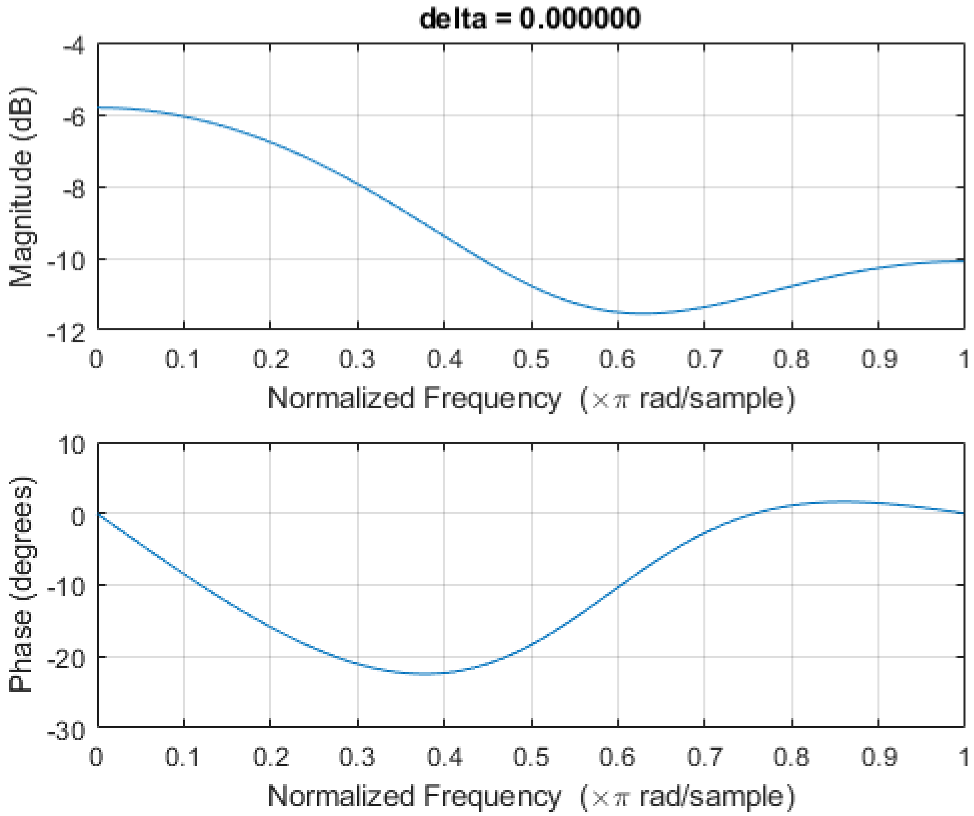
13.1

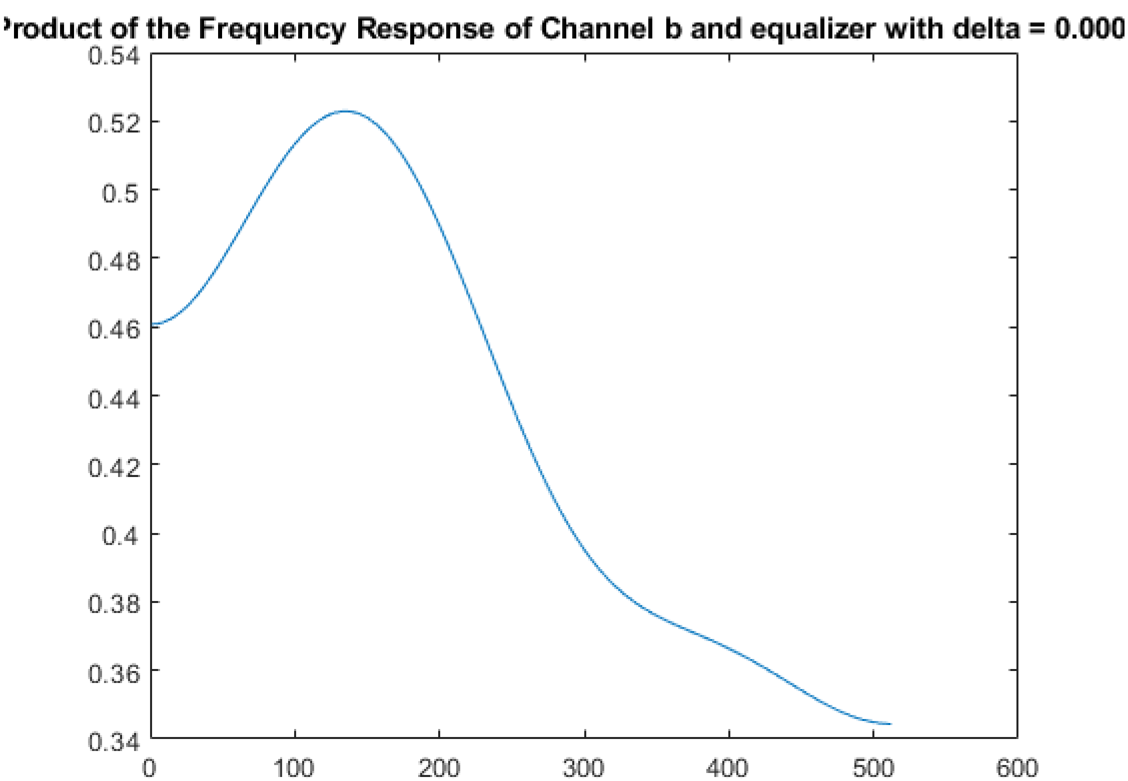
Plot the frequency response (using freqz) of the channel b in LSequalizer.m. Plot the frequency response of each of the four equalizers found by the program. For each channel/equalizer pair, form the product of the magnitude of the frequency responses. How close are these products to unity?

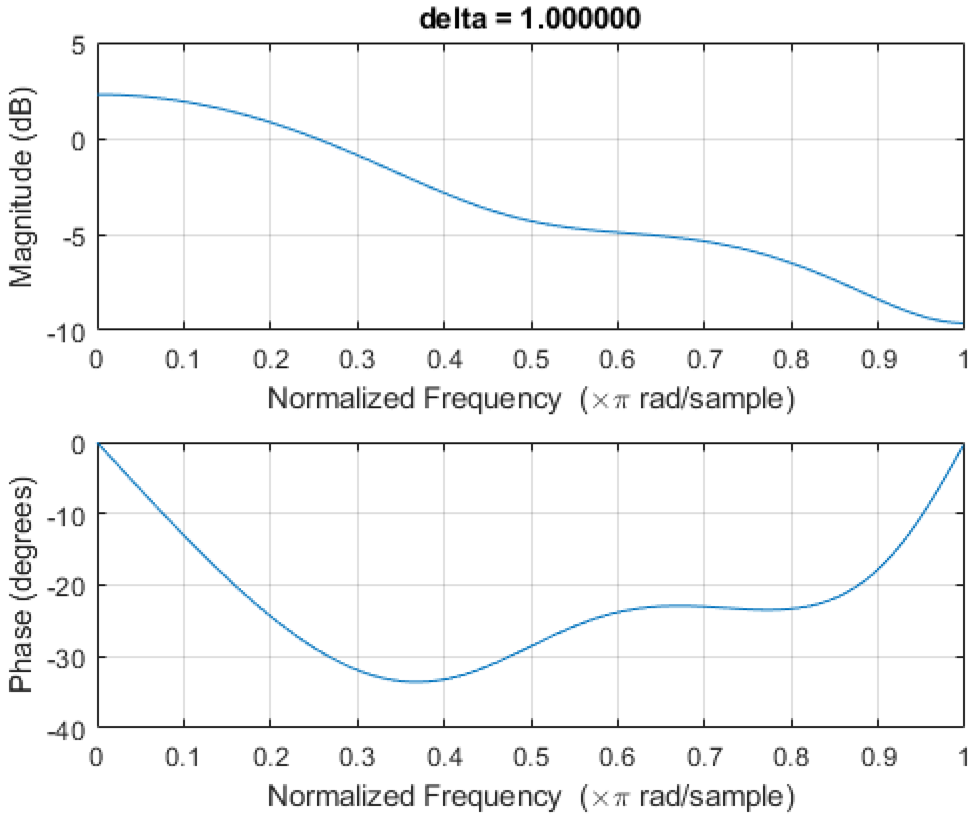
```
% LSequalizer.m find a LS equalizer f for the channel b

for q = 0:3;
    figure
    b=[0.5 1 -0.6];                    % define channel
    m=1000; s=sign(randn(1,m));        % binary source of length m
    r=filter(b,1,s);                   % output of channel
    n=3;                               % length of equalizer - 1
    delta=q;                           % use delay <=n*length(b)
    p=length(r)-delta;
    R=toeplitz(r(n+1:p),r(n+1:-1:1));  % build matrix R
    S=s(n+1-delta:p-delta)';           % and vector S
    f=inv(R'*R)*R'*S  ;                 % calculate equalizer f
    Jmin=S'*S-S'*R*inv(R'*R)*R'*S;      % Jmin for this f and delta
    y=filter(f,1,r);                   % equalizer is a filter
    dec=sign(y);                       % quantize and find errors
    err=0.5*sum(abs(dec(delta+1:m)-s(1:m-delta)))

    [hf,wf]=freqz(f);
    freqz(f);
```

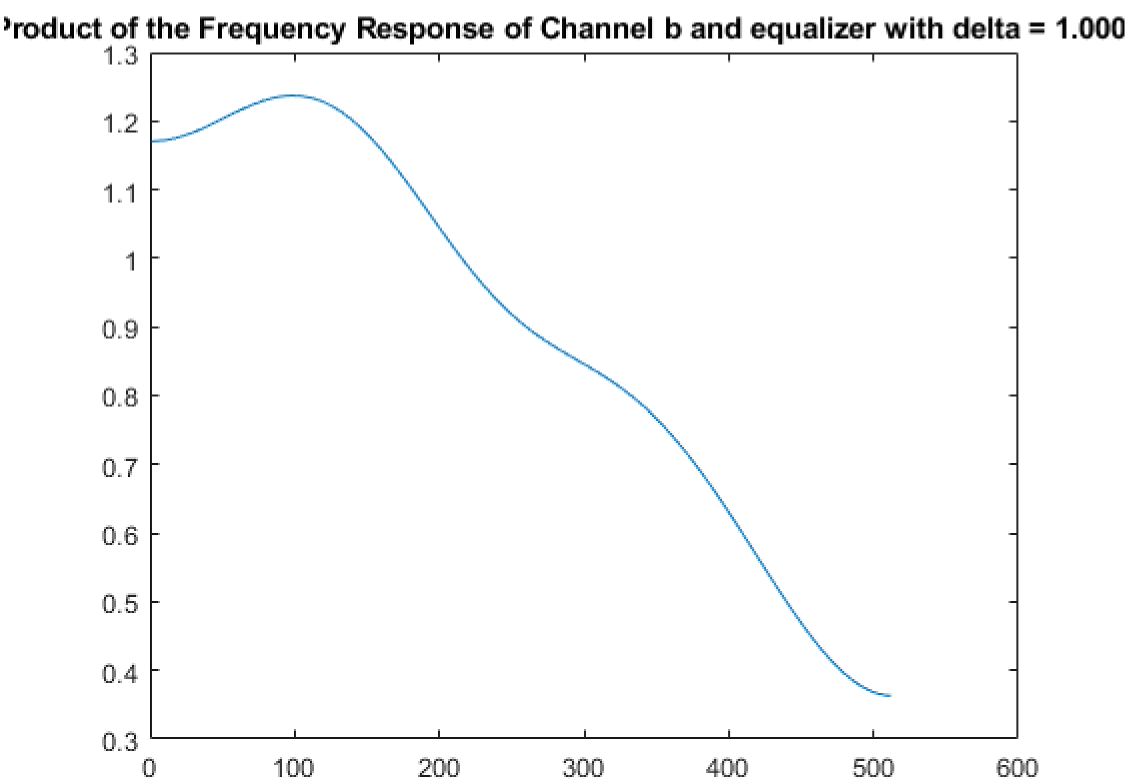
```
    title(sprintf("delta = %f",q));

    [hb,wb]=freqz(b);
    h_mult = zeros(1,length(hf));
    for p = 1:length(hf)
       h_mult(p) = abs(hb(p)*hf(p));
    end
    figure(); plot(1:length(h_mult),h_mult);
    title(sprintf("Product of the Frequency Response of Channel b and
 equalizer with delta = %f",q))

end
freqz(b)
title("frequency response of channel b")

% The product of frequency response magnitudes demonstrates that,
 while
% these equalizers cannot force the channel to exactly match unity,
 they
% can bring it significantly closer. Note that the equalizer with a
 larger
% delta is able to more effectively correct the channel's frequency
% response.
```

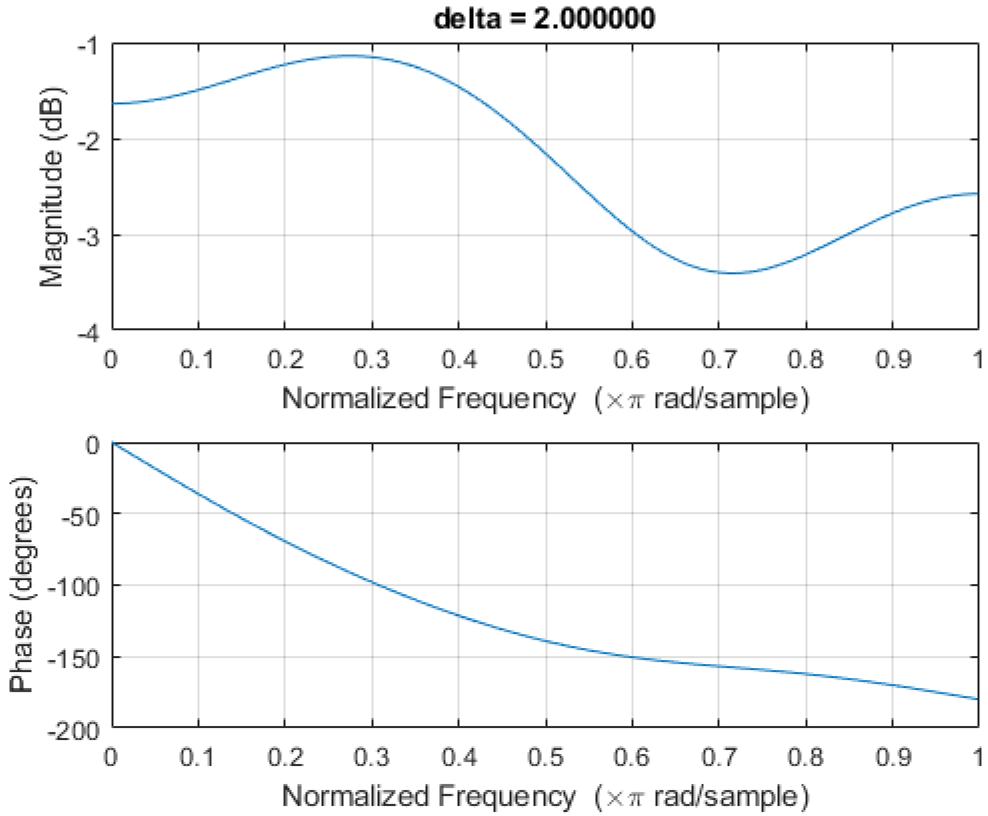
```
err =

   461


err =

     0


err =

     0


err =

     0
```

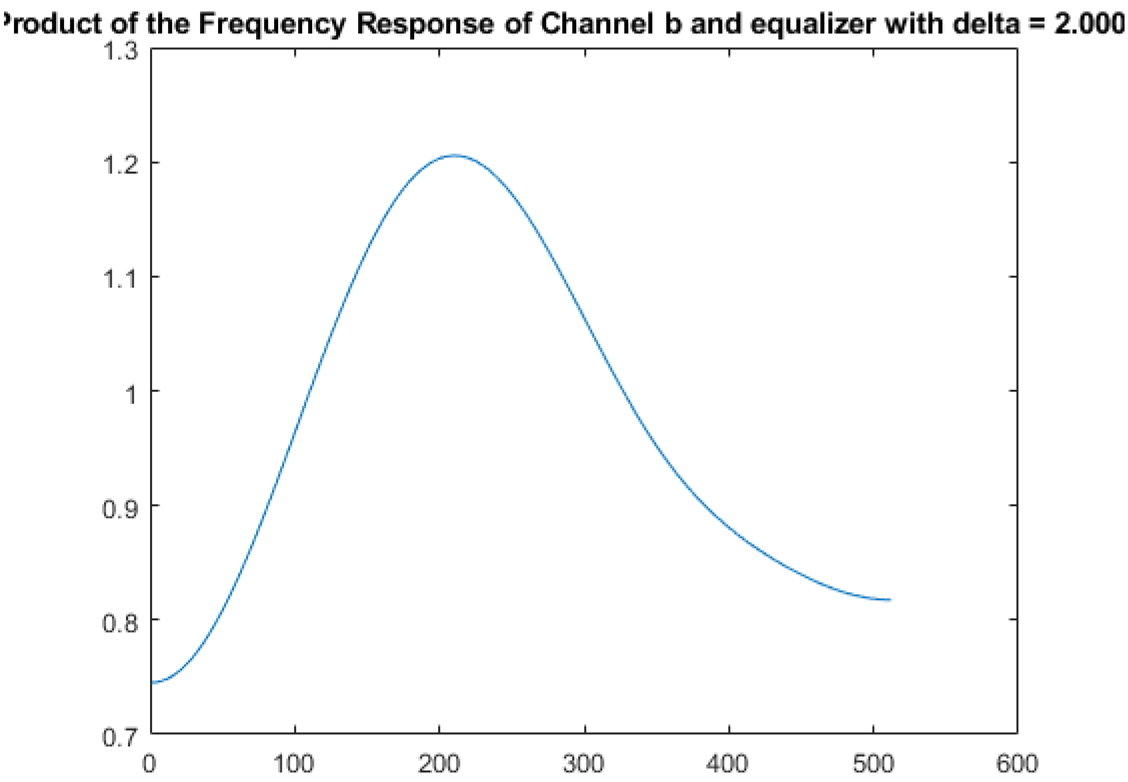

delta = 0.000000
Magnitude (dB)
-4
-6
-8
-10
-12
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Normalized Frequency (×π rad/sample)
Phase (degrees)
10
0
-10
-20
-30
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Normalized Frequency (×π rad/sample)

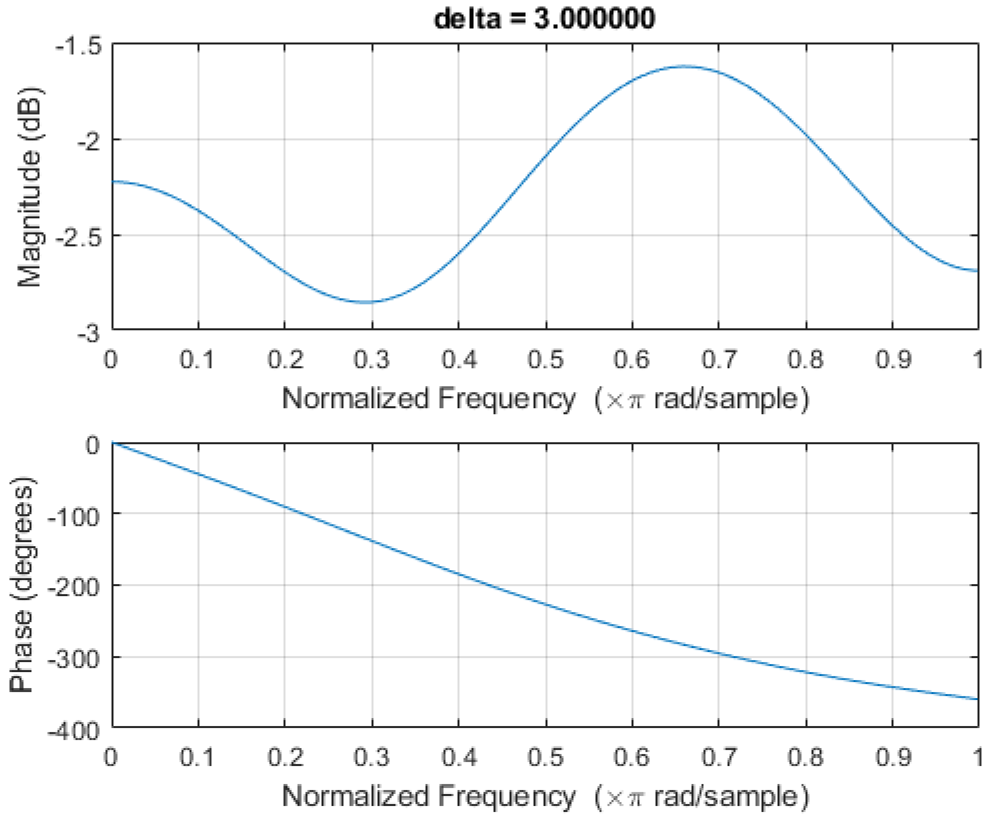

roduct of the Frequency Response of Channel b and equalizer with delta = 0.000
0.54
0.52
0.5
0.48
0.46
0.44
0.42
0.4
0.38
0.36
0.34
0 100 200 300 400 500 600

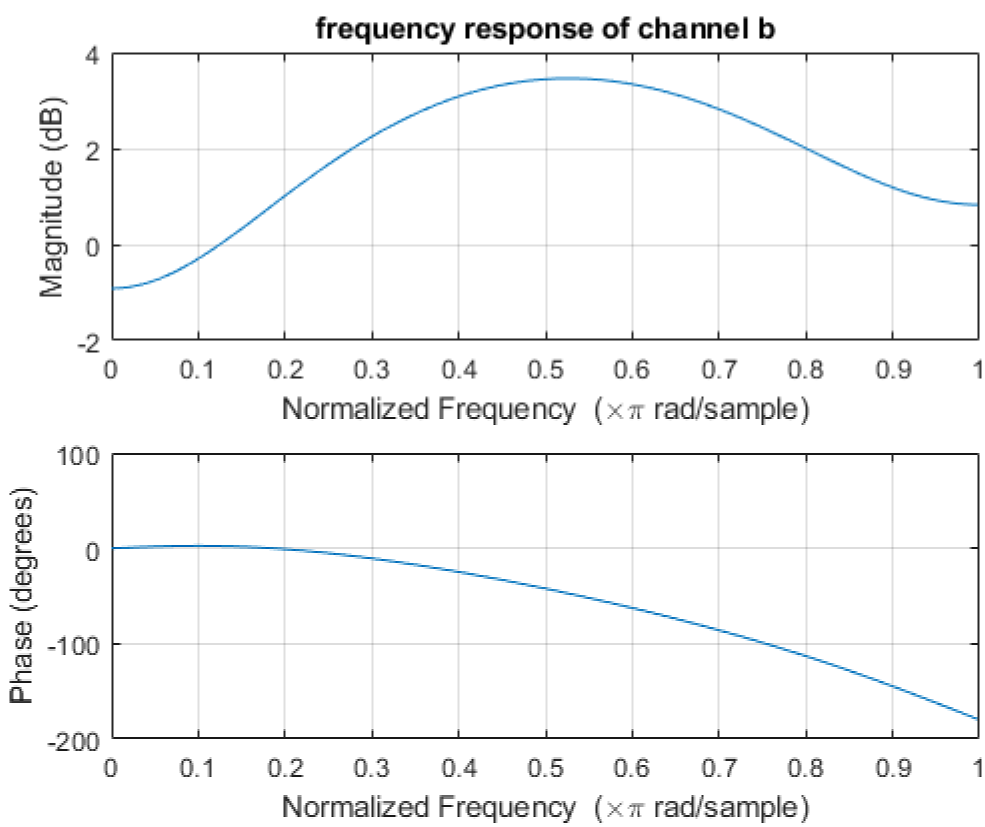

delta = 1.000000
Magnitude (dB)
Normalized Frequency (×π rad/sample)
Phase (degrees)
Normalized Frequency (×π rad/sample)


'roduct of the Frequency Response of Channel b and equalizer with delta = 1.000


delta = 2.000000
Magnitude (dB)
Normalized Frequency (×π rad/sample)
Phase (degrees)
Normalized Frequency (×π rad/sample)


Product of the Frequency Response of Channel b and equalizer with delta = 2.000


delta = 3.000000
Magnitude (dB)
-1.5
-2
-2.5
-3
Normalized Frequency (×π rad/sample)
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Phase (degrees)
0
-100
-200
-300
-400
Normalized Frequency (×π rad/sample)
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

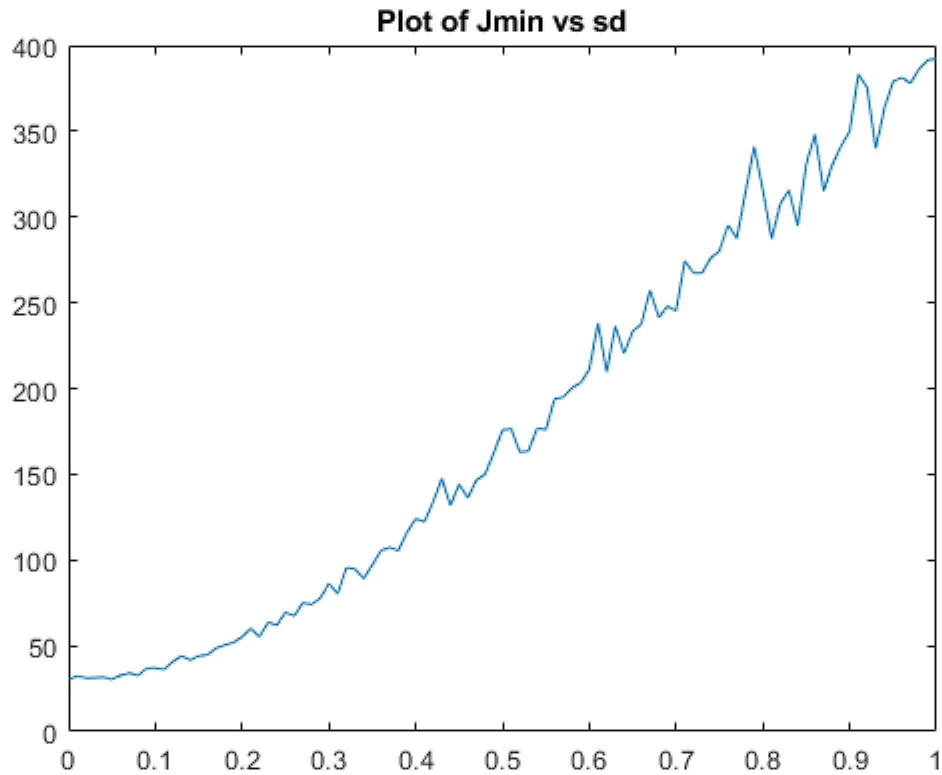

frequency response of channel b
Magnitude (dB)
4
2
0
-2
Normalized Frequency (×π rad/sample)
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Phase (degrees)
100
0
-100
-200
Normalized Frequency (×π rad/sample)
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

13.2

Add (uncorrelated, normally distributed) noise into the simulation using the command r=filter(b,1,s)+sd*randn(size(s)).

```
% a)
sd = 0.32 % this was the maximum value of sd I could add to not get
 any errors

b=[0.5 1 -0.6];                      % define channel
m=1000; s=sign(randn(1,m));          % binary source of length m
r=filter(b,1,s)+sd*randn(size(s));                    % output of
 channel
n=3;                                 % length of equalizer - 1
delta=2;                             % use delay <=n*length(b)
p=length(r)-delta;
R=toeplitz(r(n+1:p),r(n+1:-1:1));    % build matrix R
S=s(n+1-delta:p-delta)';             % and vector S
f=inv(R'*R)*R'*S ;                    % calculate equalizer f
Jmin=S'*S-S'*R*inv(R'*R)*R'*S;        % Jmin for this f and delta
y=filter(f,1,r);                     % equalizer is a filter
dec=sign(y);                         % quantize and find errors
err=0.5*sum(abs(dec(delta+1:m)-s(1:m-delta)))

% b)

t = 0:0.01:1;
jmins = zeros(1,length(t));
cnt = 1;
for sd = 0:0.01:1

    b=[0.5 1 -0.6];                      % define channel
    m=1000; s=sign(randn(1,m));          % binary source of length m
    r=filter(b,1,s)+sd*randn(size(s));                    % output of
 channel
    n=3;                                 % length of equalizer - 1
    delta=2;                             % use delay <=n*length(b)
    p=length(r)-delta;
    R=toeplitz(r(n+1:p),r(n+1:-1:1));    % build matrix R
    S=s(n+1-delta:p-delta)';             % and vector S
    f=inv(R'*R)*R'*S;                     % calculate equalizer f
    Jmin=S'*S-S'*R*inv(R'*R)*R'*S;        % Jmin for this f and delta
    jmins(cnt) = Jmin;
    cnt = cnt+1;
    y=filter(f,1,r);                     % equalizer is a filter
    dec=sign(y);                         % quantize and find errors
    err=0.5*sum(abs(dec(delta+1:m)-s(1:m-delta)));

end
figure();

plot(t,jmins);
title("Plot of Jmin vs sd");
```

```
% c)
sd = 0.2 % this was the maximum value of sd I could add to not get any
 errors

b=[0.5 1 -0.6];                    % define channel
m=1000; s=sign(randn(1,m));        % binary source of length m
r=filter(b,1,s)+sd*randn(size(s));                  % output of
 channel
n=3;                               % length of equalizer - 1
delta=1;                           % use delay <=n*length(b)
p=length(r)-delta;
R=toeplitz(r(n+1:p),r(n+1:-1:1));  % build matrix R
S=s(n+1-delta:p-delta)';           % and vector S
f=inv(R'*R)*R'*S ;                  % calculate equalizer f
Jmin=S'*S-S'*R*inv(R'*R)*R'*S;      % Jmin for this f and delta
y=filter(f,1,r);                   % equalizer is a filter
dec=sign(y);                       % quantize and find errors
err=0.5*sum(abs(dec(delta+1:m)-s(1:m-delta)))

% d)
% It seems that the first equalizer is better because it can handle a
% larger amplitude of noise (corresponding to a lower SNR) without
% getting errors.


sd =

    0.3200


err =

     0


sd =

    0.2000


err =

     1
```



Published with MATLAB® R2017b