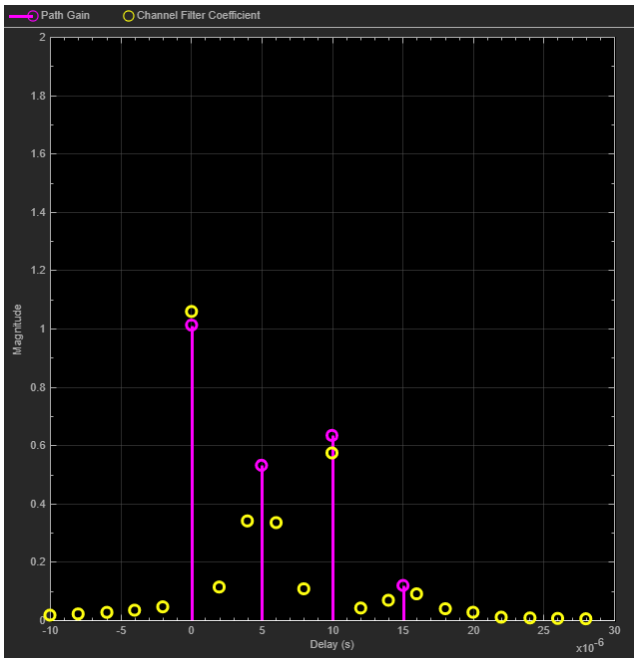
Fading and Doppler effects

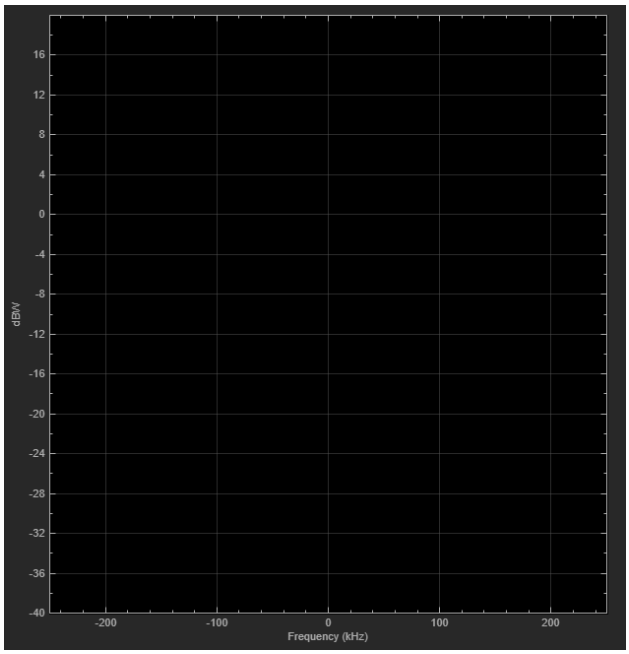
```
sampleRate500KHz = 500e3;    % Sample rate of 500 KHz
sampleRate20KHz  = 20e3;     % Sample rate of 20 KHz
maxDopplerShift  = 200; % Max Doppler shift of diffuse components (Hz)
delayVector = (0:5:15)*1e-6; % Discrete delays of four-path channel (s)
gainVector  = [0 -3 -6 -9];  % Average path gains (dB)
KFactor = 10;           % Linear ratio of specular to diffuse power
specDopplerShift = 100; % Doppler shift of specular component (Hz)
rayChan = comm.RayleighChannel( ...
   SampleRate=sampleRate500KHz, ...
   PathDelays=delayVector, ...
   AveragePathGains=gainVector, ...
   MaximumDopplerShift=maxDopplerShift, ...
   RandomStream="mt19937ar with seed", ...
   Seed=10, ...
   PathGainsOutputPort=true);
ricChan = comm.RicianChannel( ...
   SampleRate=sampleRate500KHz, ...
   PathDelays=delayVector, ...
   AveragePathGains=gainVector, ...
   KFactor=KFactor, ...
   DirectPathDopplerShift=specDopplerShift, ...
   MaximumDopplerShift=maxDopplerShift, ...
   RandomStream="mt19937ar with seed", ...
   Seed=100, ...
   PathGainsOutputPort=true);
M = 4;               % QPSK modulation
phaseoffset = pi/4;  % Phase offset for QPSK
bitsPerFrame = 1000;
msg = randi([0 1],bitsPerFrame,1);
modSignal = pskmod(msg,M,phaseoffset,InputType='bit');
rayChan(modSignal);
ricChan(modSignal);
release(rayChan);
release(ricChan);
figure(1);
rayChan.Visualization = "Impulse and frequency responses";
rayChan.SamplesToDisplay = "100%";
% Display impulse and frequency responses for 2 frames
numFrames = 2;
for i = 1:numFrames
   % Create random data
   msg = randi([0 1],bitsPerFrame,1);
   % Modulate data
   modSignal = pskmod(msg,M,phaseoffset,InputType='bit');
   % Filter data through channel and show channel responses
   rayChan(modSignal);
```

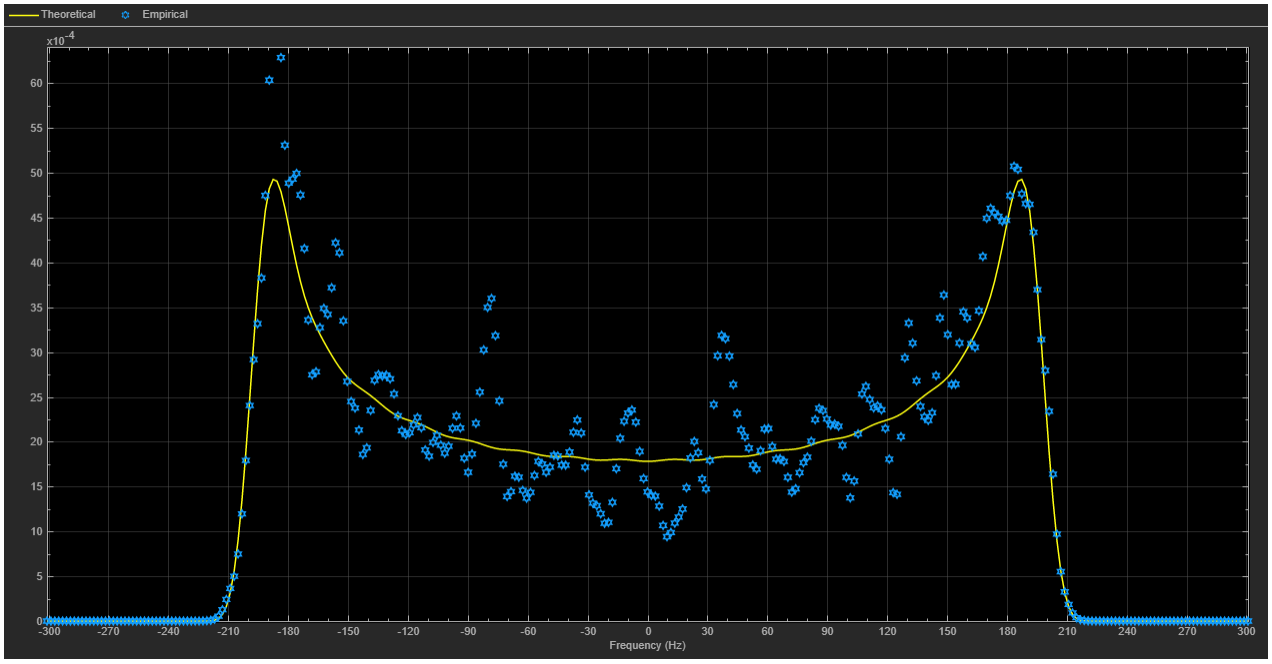
```
end
release(rayChan);
rayChan.Visualization = "Doppler spectrum";
% Display Doppler spectrum from 5000 frame transmission
numFrames = 5000;
for i = 1:numFrames
   msg = randi([0 1],bitsPerFrame,1);
   modSignal = pskmod(msg,M,phaseoffset,InputType='bit');
   rayChan(modSignal);
end
release(rayChan);
rayChan.Visualization = "Impulse and frequency responses";
rayChan.SampleRate = sampleRate20KHz;
rayChan.SamplesToDisplay = "25%";  % Display one of every four samples
% Display impulse and frequency responses for 2 frames
numFrames = 2;
for i = 1:numFrames
   msg = randi([0 1],bitsPerFrame,1);
   modSignal = pskmod(msg,M,phaseoffset,InputType='bit');
   rayChan(modSignal);
end
release(rayChan);
rayChan.PathDelays = 0;        % Single fading path with zero delay
rayChan.AveragePathGains = 0;  % Average path gain of 1 (0 dB)
for i = 1:numFrames
   msg = randi([0 1],bitsPerFrame,1);
   modSignal = pskmod(msg,M,phaseoffset,InputType='bit');
   rayChan(modSignal);
end
release(rayChan);
rayChan.Visualization = "Off"; % Turn off Rayliegh object visualization
ricChan.Visualization = "Off"; % Turn off Rician object visualization
% Same sample rate and delay profile for the Rayleigh and Rician objects
ricChan.SampleRate = rayChan.SampleRate;
ricChan.PathDelays = rayChan.PathDelays;
ricChan.AveragePathGains = rayChan.AveragePathGains;
% Configure a Time Scope System object to show path gain magnitude
gainScope = timescope( ...
   SampleRate=rayChan.SampleRate, ...
   TimeSpanSource="Property",...
   TimeSpan=bitsPerFrame/2/rayChan.SampleRate, ... % One frame span
   Name="Multipath Gain", ...
   ChannelName=["Rayleigh","Rician"], ...
   ShowGrid=true, ...
   YLimits=[-40 10], ...
   YLabel="Gain (dB)");
% Compare the path gain outputs from both objects for one frame
msg = randi([0 1],bitsPerFrame,1);
modSignal = pskmod(msg,M,phaseoffset,InputType='bit');
```

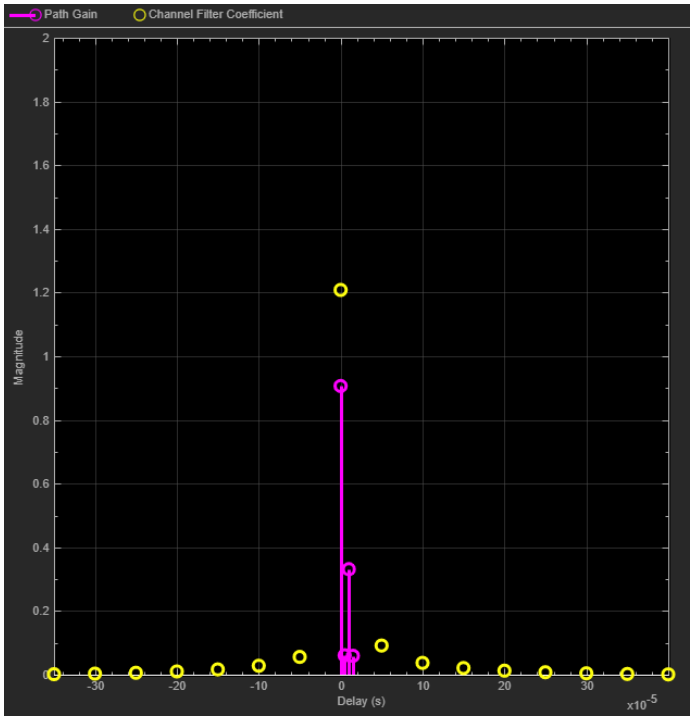
```
[~,rayPathGain] = rayChan(modSignal);
[~,ricPathGain] = ricChan(modSignal);
% Form the path gains as a two-channel input to the time scope
gainScope(10*log10(abs([rayPathGain,ricPathGain]).^2));
```

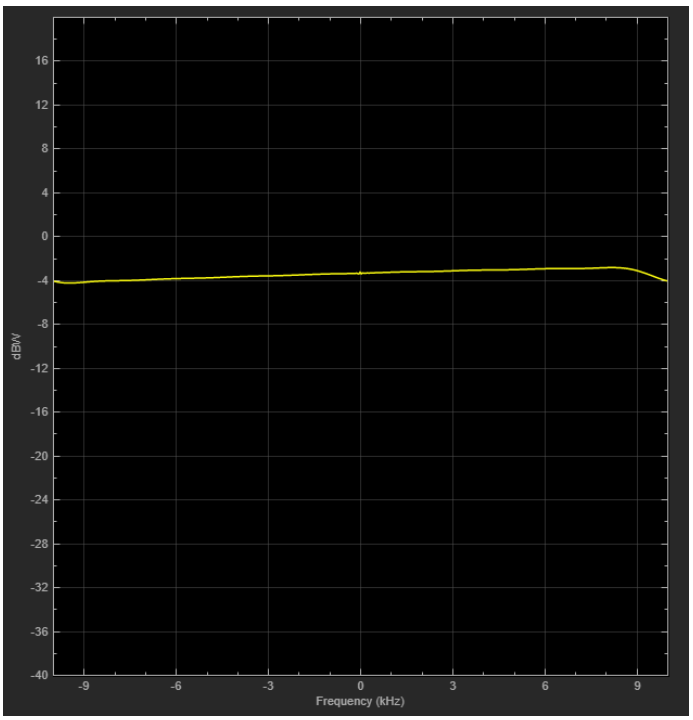
Output:

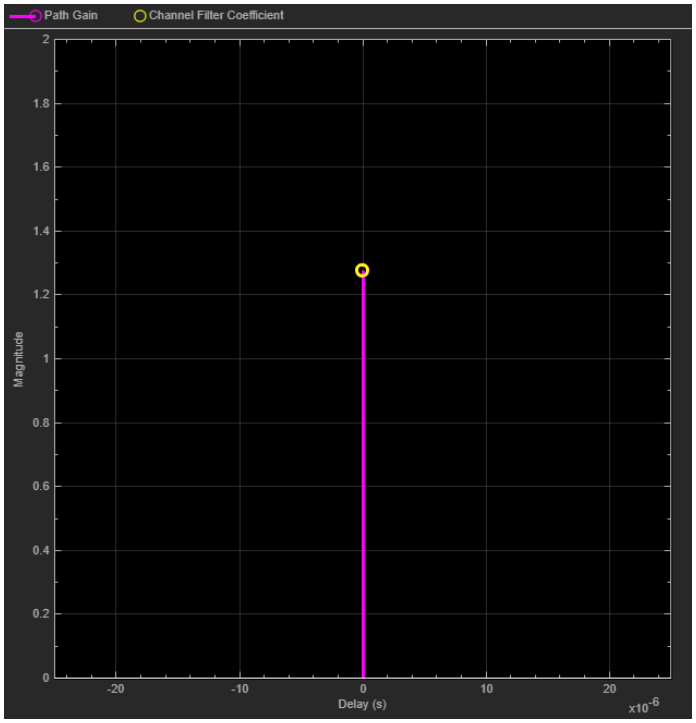


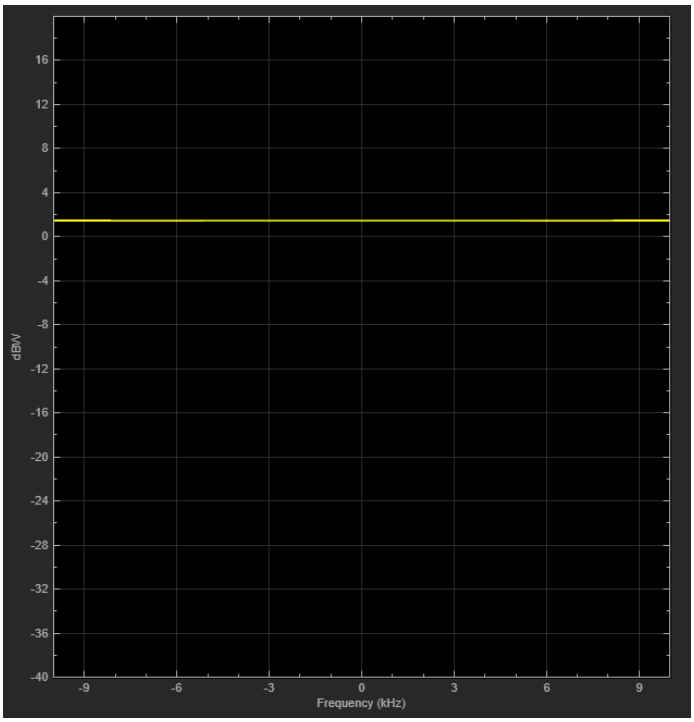


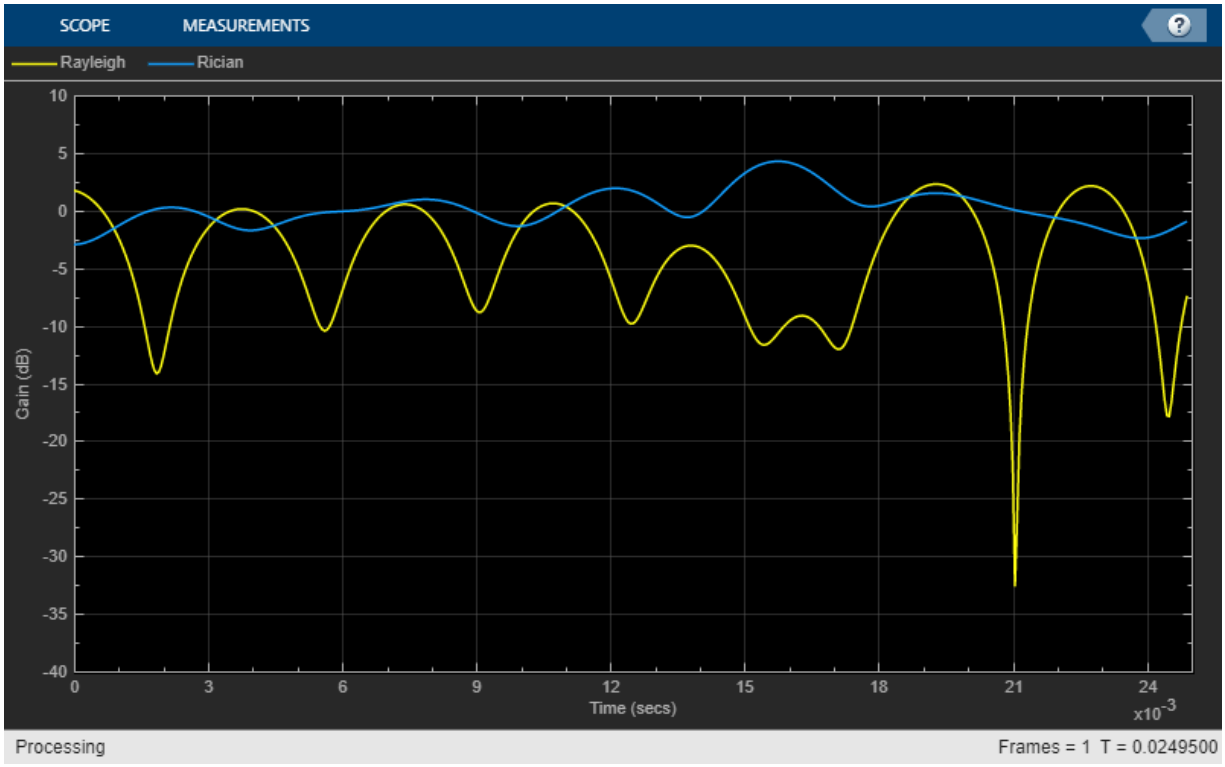



Path Gain
Channel Filter Coefficient
Magnitude
Delay (s)
x10^-5


dBW
Frequency (kHz)


Path Gain
Channel Filter Coefficient
Magnitude
Delay (s)
x10^-6


dBW
Frequency (kHz)



CHANNEL ESTIMATION

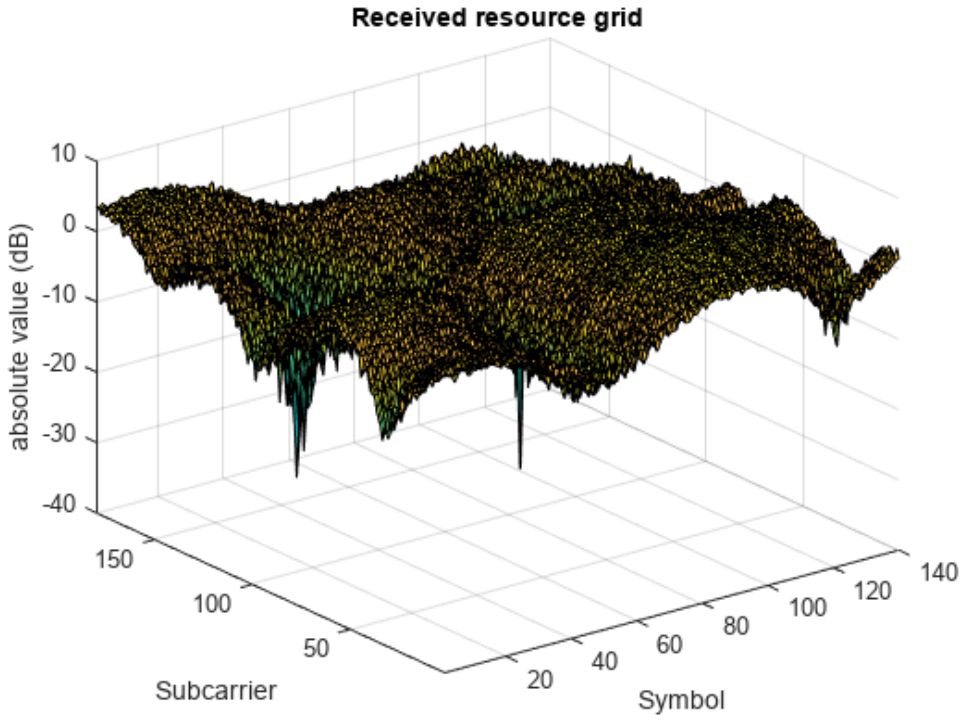
```
enb.NDLRB = 15;                 % Number of resource blocks
enb.CellRefP = 1;               % One transmit antenna port
enb.NCellID = 10;               % Cell ID
enb.CyclicPrefix = 'Normal';    % Normal cyclic prefix
enb.DuplexMode = 'FDD';         % FDD
SNRdB = 22;             % Desired SNR in dB
SNR = 10^(SNRdB/20);    % Linear SNR
rng('default');         % Configure random number generators
cfg.Seed = 1;                   % Channel seed
cfg.NRxAnts = 1;                % 1 receive antenna
cfg.DelayProfile = 'EVA';       % EVA delay spread
cfg.DopplerFreq = 120;          % 120Hz Doppler frequency
cfg.MIMOCorrelation = 'Low';    % Low (no) MIMO correlation
cfg.InitTime = 0;               % Initialize at time zero
cfg.NTerms = 16;                % Oscillators used in fading model
cfg.ModelType = 'GMEDS';        % Rayleigh fading model type
cfg.InitPhase = 'Random';       % Random initial phases
cfg.NormalizePathGains = 'On'; % Normalize delay profile power
cfg.NormalizeTxAnts = 'On';     % Normalize for transmit antennas
cec.PilotAverage = 'UserDefined'; % Pilot averaging method
cec.FreqWindow = 9;               % Frequency averaging window in REs
cec.TimeWindow = 9;               % Time averaging window in REs
```

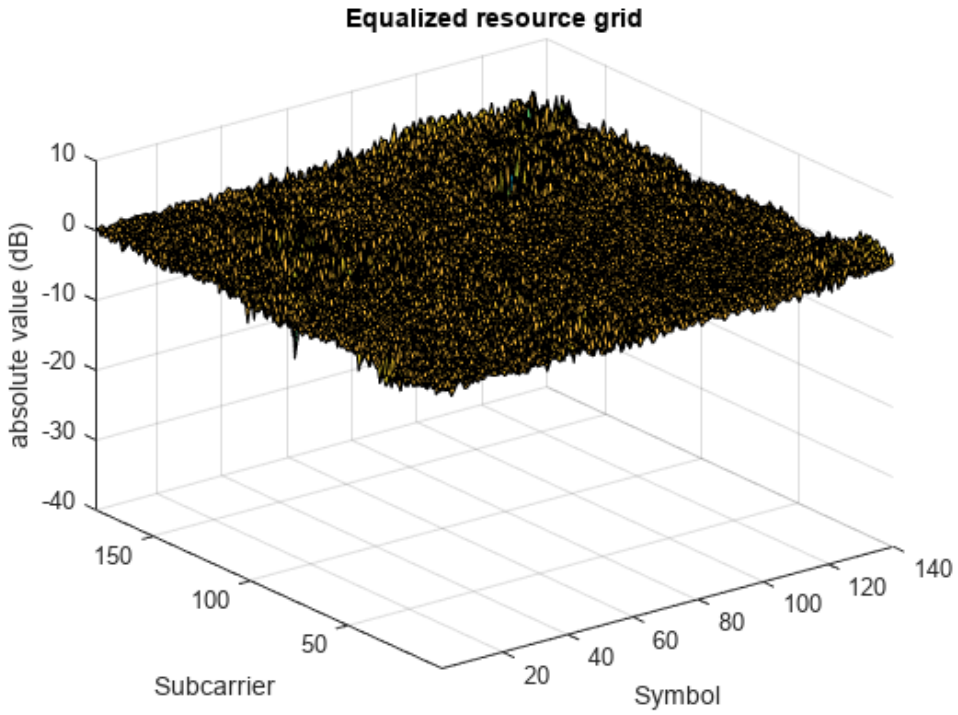
```
cec.InterpType = 'Cubic';         % Cubic interpolation
cec.InterpWinSize = 3;            % Interpolate up to 3 subframes
                                  % simultaneously
cec.InterpWindow = 'Centred';     % Interpolation windowing method
gridsize = lteDLResourceGridSize(enb);
K = gridsize(1);    % Number of subcarriers
L = gridsize(2);    % Number of OFDM symbols in one subframe
P = gridsize(3);    % Number of transmit antenna ports
txGrid = [];
% Number of bits needed is size of resource grid (K*L*P) * number of bits
% per symbol (2 for QPSK)
numberOfBits = K*L*P*2;
% Create random bit stream
inputBits = randi([0 1], numberOfBits, 1);
% Modulate input bits
inputSym = lteSymbolModulate(inputBits,'QPSK');
% For all subframes within the frame
for sf = 0:10

   % Set subframe number
   enb.NSubframe = mod(sf,10);

   % Generate empty subframe
   subframe = lteDLResourceGrid(enb);

   % Map input symbols to grid
   subframe(:) = inputSym;
   % Generate synchronizing signals
   pssSym = ltePSS(enb);
   sssSym = lteSSS(enb);
   pssInd = ltePSSIndices(enb);
   sssInd = lteSSSIndices(enb);
   % Map synchronizing signals to the grid
   subframe(pssInd) = pssSym;
   subframe(sssInd) = sssSym;
   % Generate cell specific reference signal symbols and indices
   cellRsSym = lteCellRS(enb);
   cellRsInd = lteCellRSIndices(enb);
   % Map cell specific reference signal to grid
   subframe(cellRsInd) = cellRsSym;

   % Append subframe to grid to be transmitted
   txGrid = [txGrid subframe]; %#ok

end
[txWaveform,info] = lteOFDMModulate(enb,txGrid);
txGrid = txGrid(:,1:140);
cfg.SamplingRate = info.SamplingRate;
% Pass data through the fading channel model
```

```
rxWaveform = lteFadingChannel(cfg,txWaveform);
% Calculate noise gain
N0 = 1/(sqrt(2.0*enb.CellRefP*double(info.Nfft))*SNR);
% Create additive white Gaussian noise
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
% Add noise to the received time domain waveform
rxWaveform = rxWaveform + noise;
offset = lteDLFrameOffset(enb,rxWaveform);
rxWaveform = rxWaveform(1+offset:end,:);
rxGrid = lteOFDMDemodulate(enb,rxWaveform);
enb.NSubframe = 0;
[estChannel, noiseEst] = lteDLChannelEstimate(enb,cec,rxGrid);
eqGrid = lteEqualizeMMSE(rxGrid, estChannel, noiseEst);
% Calculate error between transmitted and equalized grid
eqError = txGrid - eqGrid;
rxError = txGrid - rxGrid;
% Compute EVM across all input values
% EVM of pre-equalized receive signal
EVM = comm.EVM;
EVM.AveragingDimensions = [1 2];
preEqualisedEVM = EVM(txGrid,rxGrid);
fprintf('Percentage RMS EVM of Pre-Equalized signal: %0.3f%%\n', ...
       preEqualisedEVM);
% EVM of post-equalized receive signal
postEqualisedEVM = EVM(txGrid,eqGrid);
fprintf('Percentage RMS EVM of Post-Equalized signal: %0.3f%%\n', ...
       postEqualisedEVM);
% Plot the received and equalized resource grids
hDownlinkEstimationEqualizationResults(rxGrid, eqGrid);
```

OUTPUT:

```
Percentage RMS EVM of Pre-Equalized signal: 124.133%
Percentage RMS EVM of Post-Equalized signal: 15.598%
```

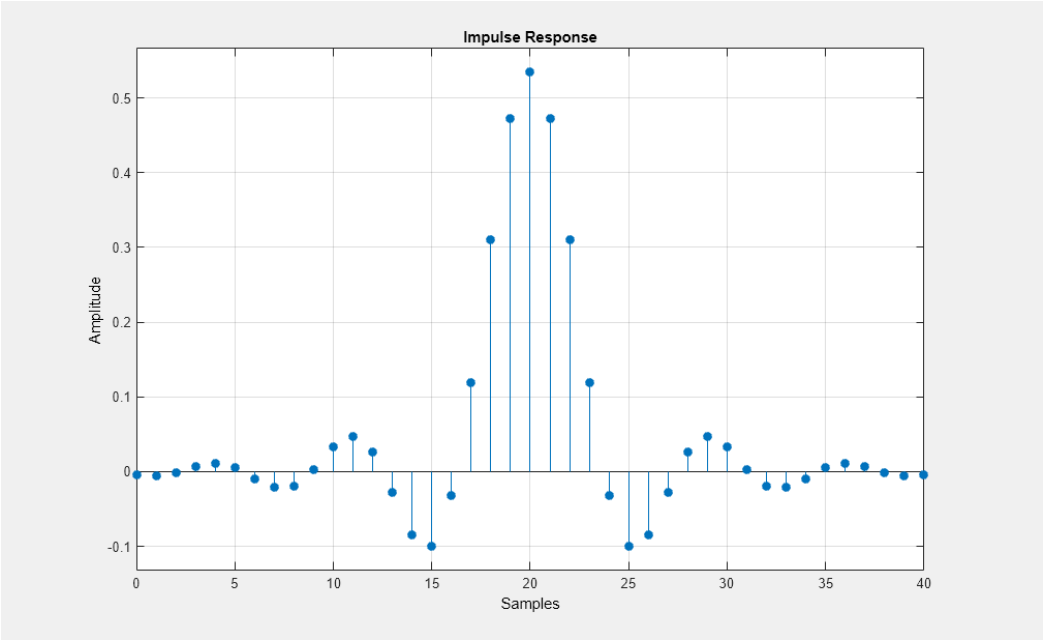

Received resource grid
absolute value (dB)
10
0
-10
-20
-30
-40
150
100
50
Subcarrier
20
40
60
80
100
120
140
Symbol

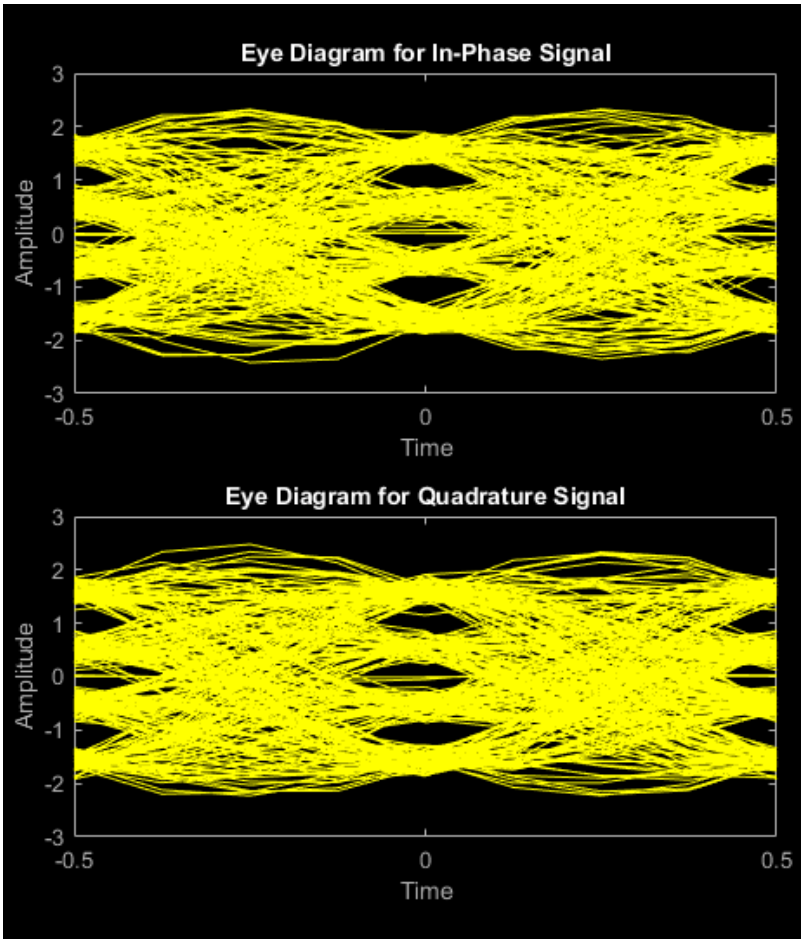

Equalized resource grid
absolute value (dB)
10
0
-10
-20
-30
-40
150
100
50
Subcarrier
20
40
60
80
100
120
140
Symbol

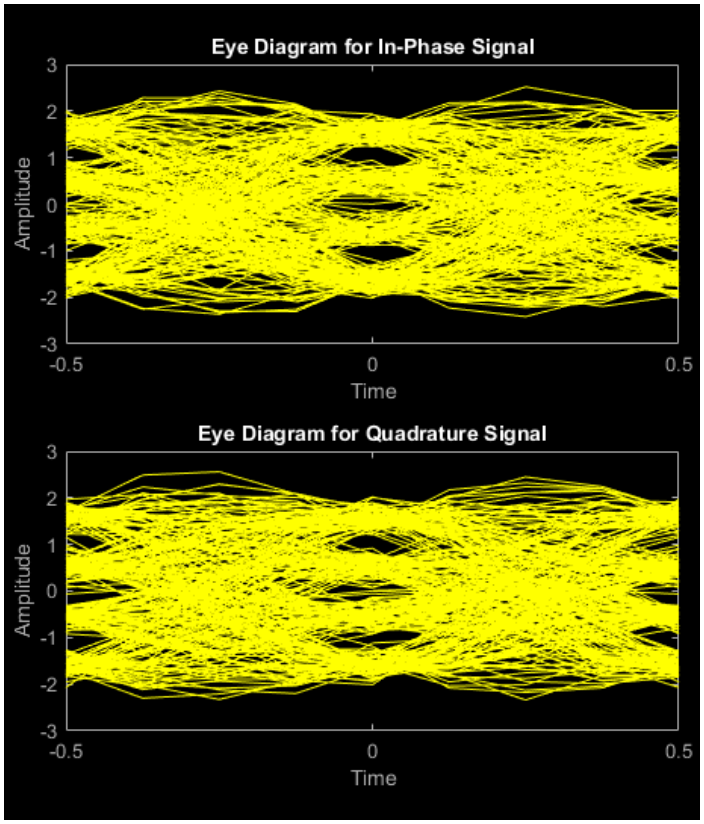
PULSE SHAPING

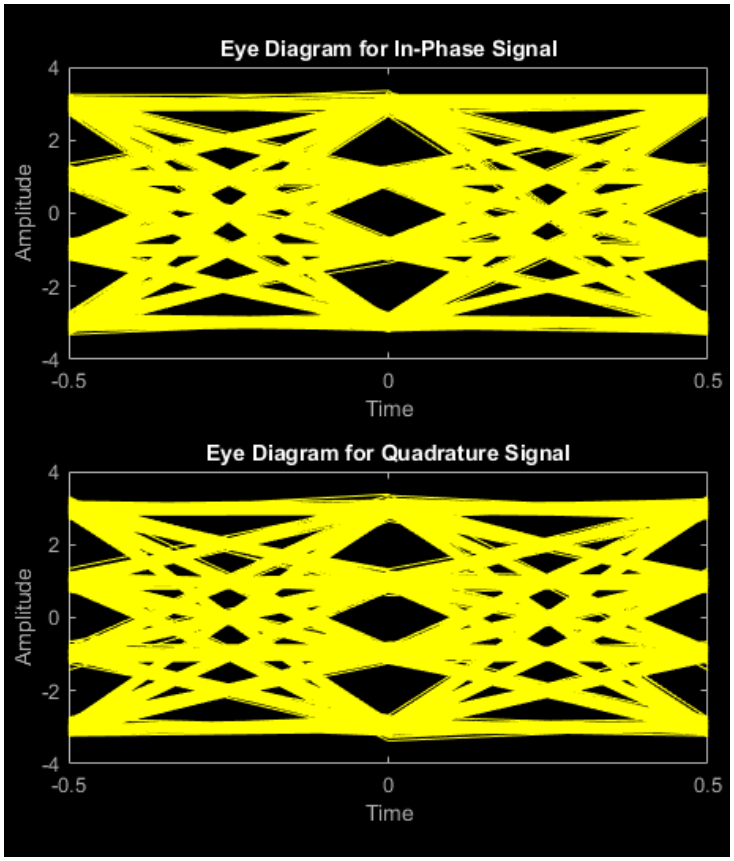
```
M = 16;            % Modulation order
k = log2(M);       % Bits per symbol
numBits = k*7.5e4; % Bits to process
sps = 4;           % Samples per symbol (oversampling factor)
filtlen = 10;      % Filter length in symbols
rolloff = 0.25;    % Filter rolloff factor
rrcFilter = rcosdesign(rolloff,filtlen,sps);
fvtool(rrcFilter,'Analysis','Impulse')
rng default;                    % Default random number generator
dataIn = randi([0 1],numBits,1); % Generate vector of binary data
dataSymbolsIn = bit2int(dataIn,k);
dataMod = qammod(dataSymbolsIn,M);
txFiltSignal = upfirdn(dataMod,rrcFilter,sps,1);
EbNo = 10;
snr = convertSNR(EbNo,'ebno', ...
   samplespersymbol=sps, ...
   bitspersymbol=k);
rxSignal = awgn(txFiltSignal,snr,'measured');
rxFiltSignal = ...
   upfirdn(rxSignal,rrcFilter,1,sps);       % Downsample and filter
rxFiltSignal = ...
   rxFiltSignal(filtlen + 1:end - filtlen); % Account for delay
dataSymbolsOut = qamdemod(rxFiltSignal,M);
dataOut = int2bit(dataSymbolsOut,k);
[numErrors,ber] = biterr(dataIn,dataOut);
fprintf(['\nFor an EbNo setting of %3.1f dB, ' ...
   'the bit error rate is %5.2e, based on %d errors.\n'], ...
   EbNo,ber,numErrors)
EbNo = 20;
snr = convertSNR(EbNo,'ebno', ...
   samplespersymbol=sps, ...
   bitspersymbol=k);
rxSignal = awgn(txFiltSignal,snr,'measured');
rxFiltSignal = upfirdn(rxSignal,rrcFilter,1,sps);       % Downsample and filter
rxFiltSignal = rxFiltSignal(filtlen + 1:end - filtlen); % Account for delay
eyediagram(txFiltSignal(1:2000),sps*2);
eyediagram(rxSignal(1:2000),sps*2);
eyediagram(rxFiltSignal(1:2000),2);
scatplot = scatterplot(sqrt(sps)*...
   rxSignal(1:sps*5e3),...
   sps,0);
hold on;
scatterplot(rxFiltSignal(1:5e3),1,0,'bx',scatplot);
title('Received Signal, Before and After Filtering');
legend('Before Filtering','After Filtering');
axis([-5 5 -5 5]); % Set axis range
hold off;
```

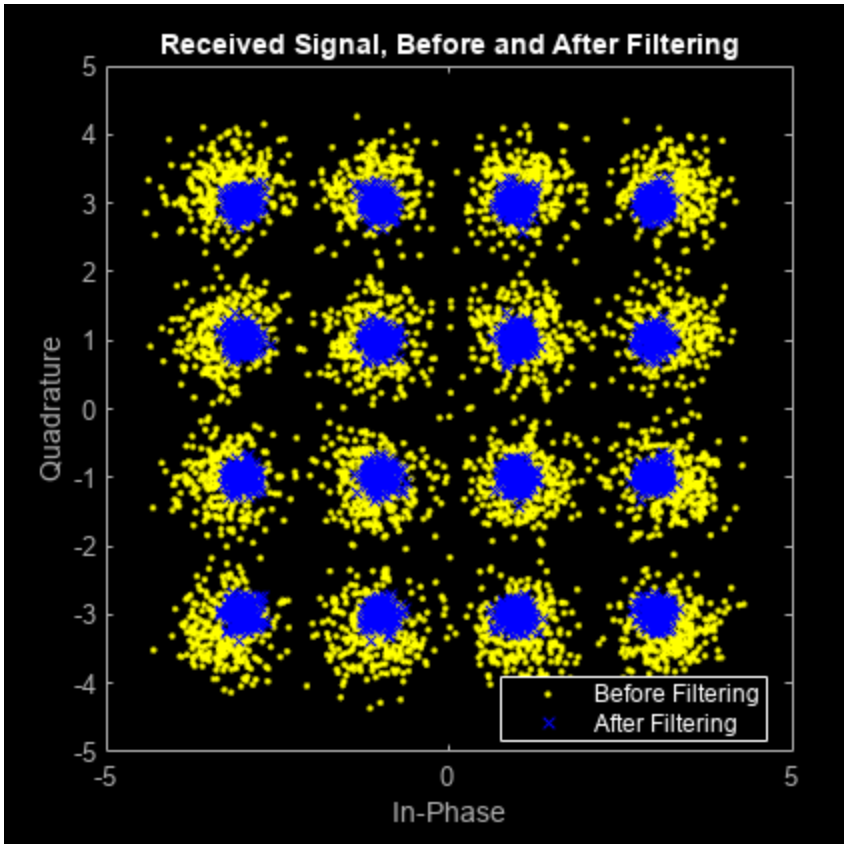
OUTPUT:


Impulse Response
Amplitude
Samples
0.5
0.4
0.3
0.2
0.1
0
-0.1
0
5
10
15
20
25
30
35
40


Eye Diagram for In-Phase Signal
Amplitude
Time
3
2
1
0
-1
-2
-3
-0.5
0
0.5
Eye Diagram for Quadrature Signal
Amplitude
Time
3
2
1
0
-1
-2
-3
-0.5
0
0.5


Eye Diagram for In-Phase Signal
Amplitude
Time
3
2
1
0
-1
-2
-3
-0.5
0
0.5
Eye Diagram for Quadrature Signal
Amplitude
Time
3
2
1
0
-1
-2
-3
-0.5
0
0.5


Eye Diagram for In-Phase Signal
Amplitude
Time
4
2
0
-2
-4
-0.5
0
0.5
Eye Diagram for Quadrature Signal
Amplitude
Time
4
2
0
-2
-4
-0.5
0
0.5



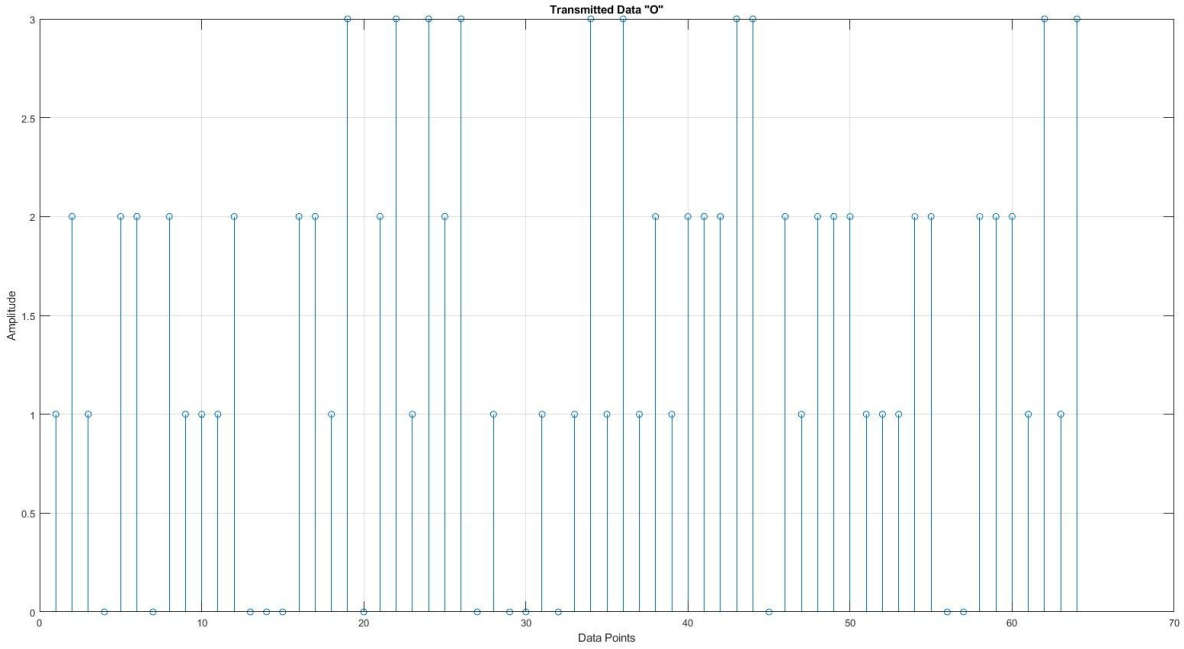
OFDM Transmitter and receiver

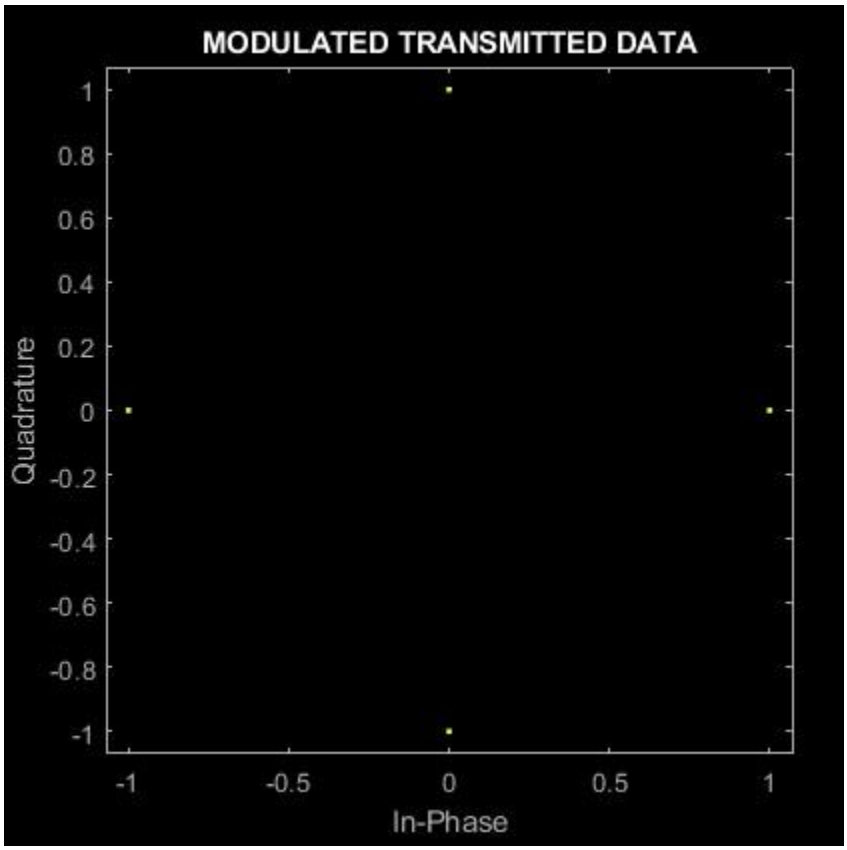
```
clear all
clc
close
%   ---------------
%   A: Setting Parameters
%   ---------------
M = 4;                          %   QPSK signal constellation
no_of_data_points = 64;        %   have 64 data points
block_size = 8;                 %   size of each ofdm block
cp_len = ceil(0.1*block_size);  %   length of cyclic prefix
no_of_ifft_points = block_size;          %   8 points for the FFT/IFFT
no_of_fft_points = block_size;
%   ---------------------------------------------
%   B:  %   +++++   TRANSMITTER    +++++
%   ---------------------------------------------
%   1.  Generate 1 x 64 vector of random data points
data_source = randsrc(1, no_of_data_points, 0:M-1);
figure(1)
stem(data_source); grid on; xlabel('Data Points'); ylabel('Amplitude')
title('Transmitted Data "O"')
```

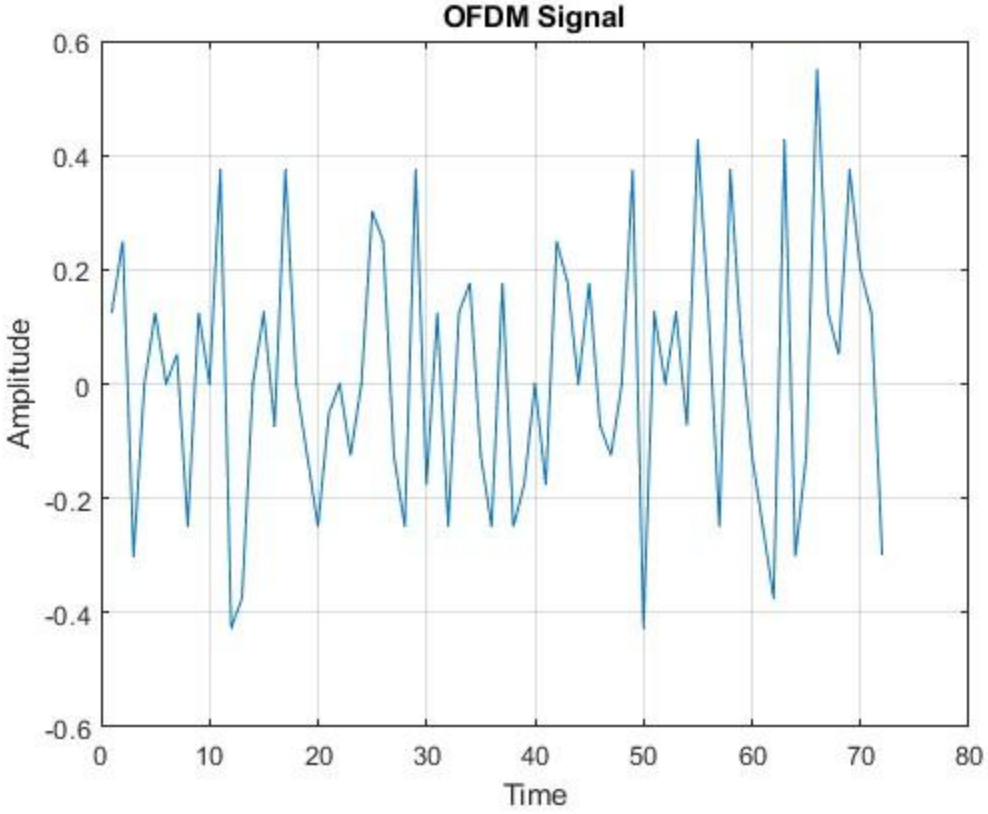
```
%   2.  Perform QPSK modulation
qpsk_modulated_data = pskmod(data_source, M);
scatterplot(qpsk_modulated_data);title('MODULATED TRANSMITTED DATA');
%   3.  Do IFFT on each block
%   Make the serial stream a matrix where each column represents a pre-OFDM
%   block (w/o cyclic prefixing)
%   First: Find out the number of colums that will exist after reshaping
num_cols=length(qpsk_modulated_data)/block_size;
data_matrix = reshape(qpsk_modulated_data, block_size, num_cols);
%   Second: Create empty matix to put the IFFT'd data
cp_start = block_size-cp_len;
cp_end = block_size;
%   Third: Operate columnwise & do CP
for i=1:num_cols
   ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),no_of_ifft_points);
   %   Compute and append Cyclic Prefix
   for j=1:cp_len
      actual_cp(j,i) = ifft_data_matrix(j+cp_start,i);
   end
   %   Append the CP to the existing block to create the actual OFDM block
   ifft_data(:,i) = vertcat(actual_cp(:,i),ifft_data_matrix(:,i));
end
%   4.  Convert to serial stream for transmission
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;
%   Actual OFDM signal to be transmitted
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);
figure(3)
plot(real(ofdm_signal)); xlabel('Time'); ylabel('Amplitude');
title('OFDM Signal');grid on;
%   ----------------------------------------
%   C:  %   +++++   HPA    +++++
%   ----------------------------------------
%To show the effect of the PA simply we will add random complex noise
%when the power exceeds the avg. value, otherwise it add nothing.
% 1. Generate random complex noise
noise = randn(1,len_ofdm_data) +  sqrt(-1)*randn(1,len_ofdm_data);
% 2. Transmitted OFDM signal after passing through HPA
avg=0.4;
for i=1:length(ofdm_signal)
     if ofdm_signal(i) > avg
         ofdm_signal(i) = ofdm_signal(i)+noise(i);
   end
   if ofdm_signal(i) < -avg
         ofdm_signal(i) = ofdm_signal(i)+noise(i);
   end
end
figure(4)
plot(real(ofdm_signal)); xlabel('Time'); ylabel('Amplitude');
```

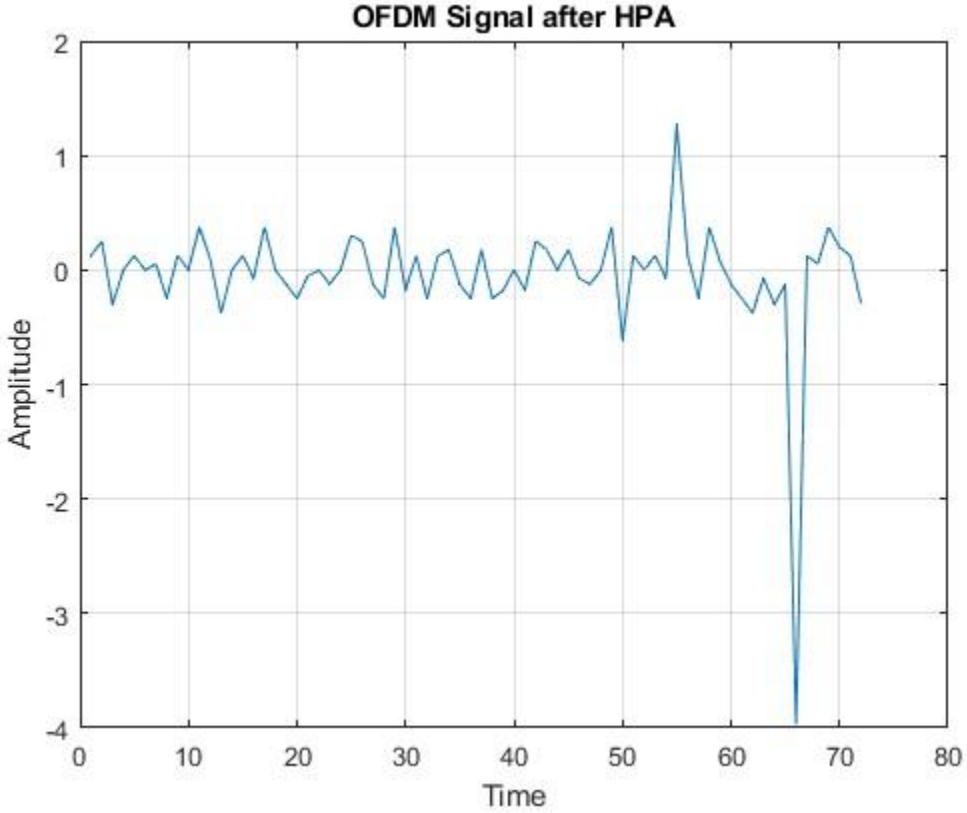
```
title('OFDM Signal after HPA');grid on;
%   --------------------------------
%   D:  %   +++++   CHANNEL    +++++
%   --------------------------------
%   Create a complex multipath channel
channel = randn(1,block_size) + sqrt(-1)*randn(1,block_size);
%   ----------------------------------------
%   E:  %   +++++   RECEIVER    +++++
%   ----------------------------------------
%   1.  Pass the ofdm signal through the channel
after_channel = filter(channel, 1, ofdm_signal);
%   2.   Add Noise
awgn_noise = awgn(zeros(1,length(after_channel)),0);
%   3.  Add noise to signal...
recvd_signal = awgn_noise+after_channel;
%   4.  Convert Data back to "parallel" form to perform FFT
recvd_signal_matrix = reshape(recvd_signal,rows_ifft_data, cols_ifft_data);
%   5.  Remove CP
recvd_signal_matrix(1:cp_len,:)=[];
%   6.  Perform FFT
for i=1:cols_ifft_data
   %   FFT
   fft_data_matrix(:,i) = fft(recvd_signal_matrix(:,i),no_of_fft_points);
end
%   7.  Convert to serial stream
recvd_serial_data = reshape(fft_data_matrix, 1,(block_size*num_cols));
scatterplot(recvd_serial_data);title('MODULATED RECEIVED DATA');
%   8.  Demodulate the data
qpsk_demodulated_data = pskdemod(recvd_serial_data,M);
scatterplot(recvd_serial_data);title('MODULATED RECEIVED DATA');
figure(5)
stem(qpsk_demodulated_data,'rx');
grid on;xlabel('Data Points');ylabel('Amplitude');title('Received Data "X"')
```

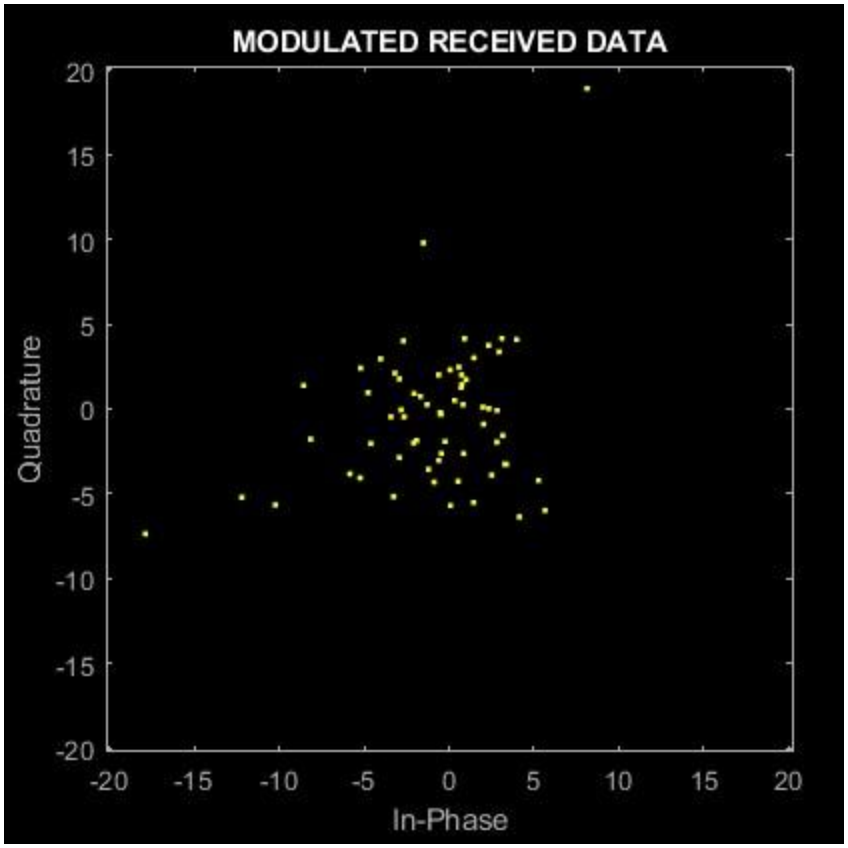
OUTPUT:

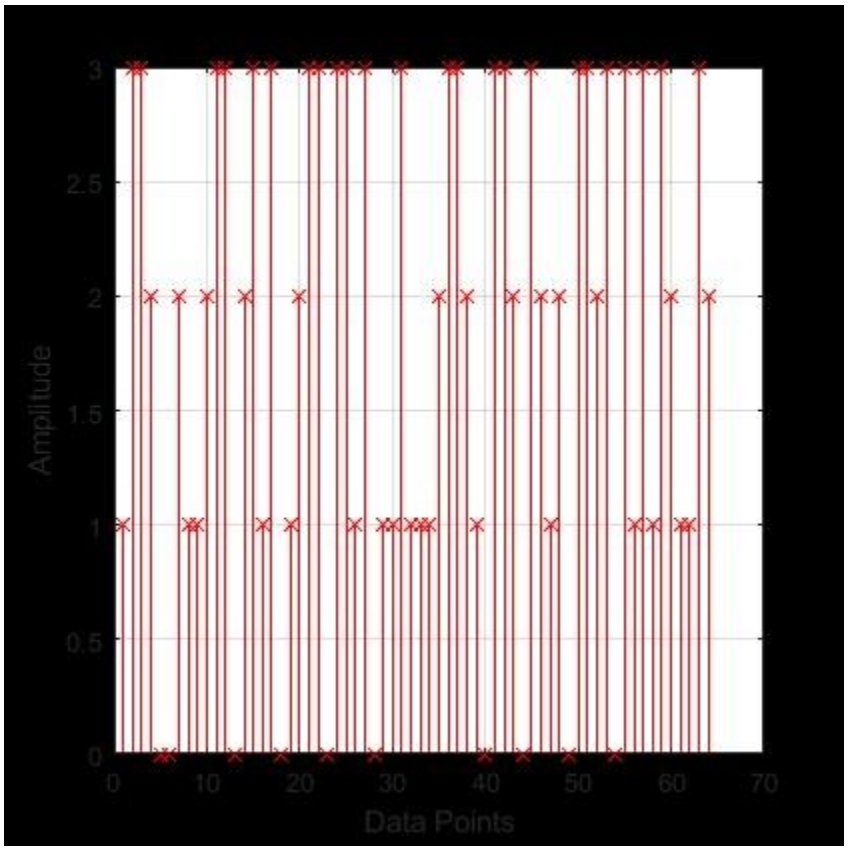

Transmitted Data "O"
Amplitude
Data Points
0
0.5
1
1.5
2
2.5
3
0
10
20
30
40
50
60
70


MODULATED TRANSMITTED DATA
Quadrature
In-Phase
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.5
0
0.5
1


OFDM Signal
Amplitude
Time
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
10
20
30
40
50
60
70
80


OFDM Signal after HPA
Amplitude
Time
2
1
0
-1
-2
-3
-4
0
10
20
30
40
50
60
70
80


MODULATED RECEIVED DATA
Quadrature
In-Phase
20
15
10
5
0
-5
-10
-15
-20
-20
-15
-10
-5
0
5
10
15
20


Amplitude
Data Points
3
2.5
2
1.5
1
0.5
0
0
10
20
30
40
50
60
70