

**EE 6390-Introduction to Wireless Communications Systems**

**Spring-2015**

**Project Report**

**Physical Layer Simulation of a Simplified LTE-OFDM System**

Tasmiah S. Khan & Gayathri N. Kannepalli

**Project Description**

This project entails simulation of simplified LTE-OFDM system in Matlab. The modulation schemes QPSK and 16-QAM are used to modulate the data. The modulated signals are transmitted over AWGN channel as well as multipath channel. The Bit Error Rate (BER) are computed under each scenario. Finally, the empirical spectral efficiency is studied for adaptive modulation scheme.

The project work was equally divided between both of us. One of us worked on Adaptive modulation and BER calculation with AWGN channel and the other worked on the BER calculation with multipath channel as well as integrating all the code to make the simulation work. The contribution on the project report was equal.

**System Parameters**

The following parameters were used to simulate the system

Channel Bandwidth = 10MHz

Modulation Types: QPSK, 16-QAM

FFT Size = 64

CP Length = 16

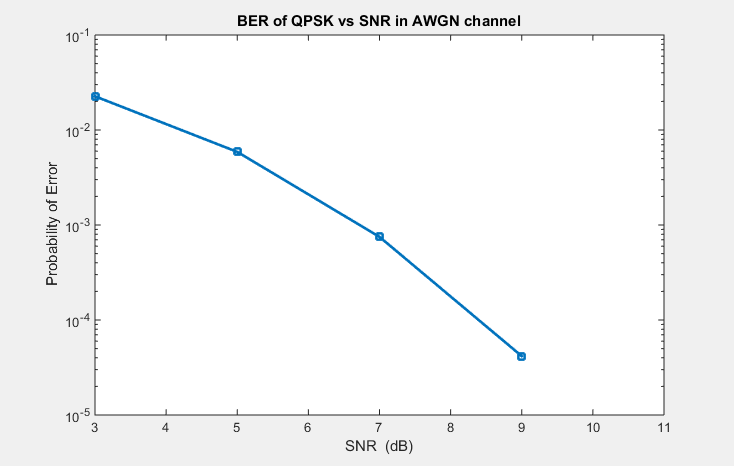
No. of used subcarriers (Nused) = 52

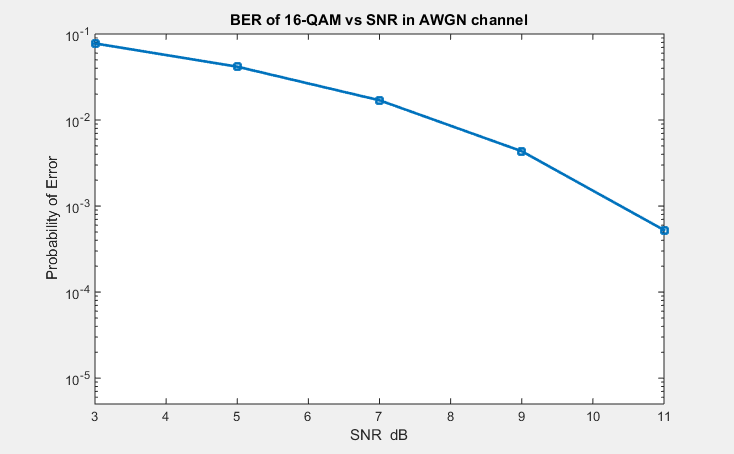
No. of pilot subcarriers (Nref) = 4

No. of data subcarriers per symbol (Ndata)= 48

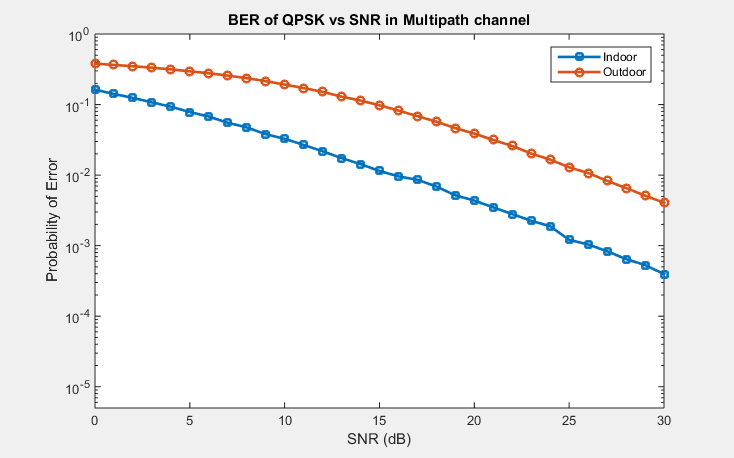
No. of null subcarriers (Nleft,Ndc,Nright) = 12

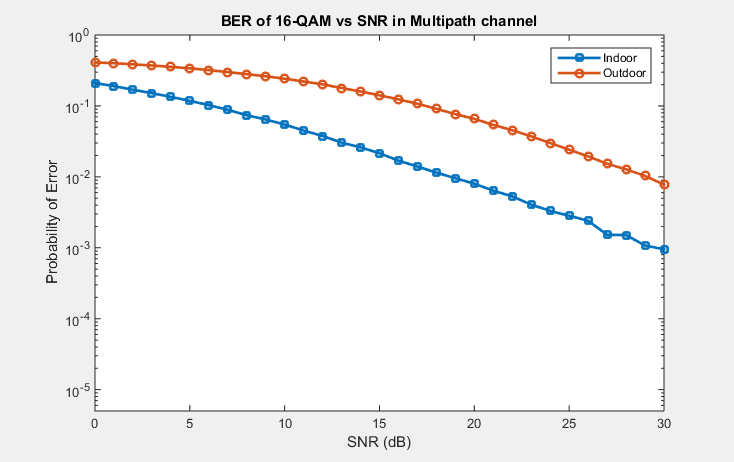
The following plots of BER are generated for QPSK and 16-QAM under AWGN channel



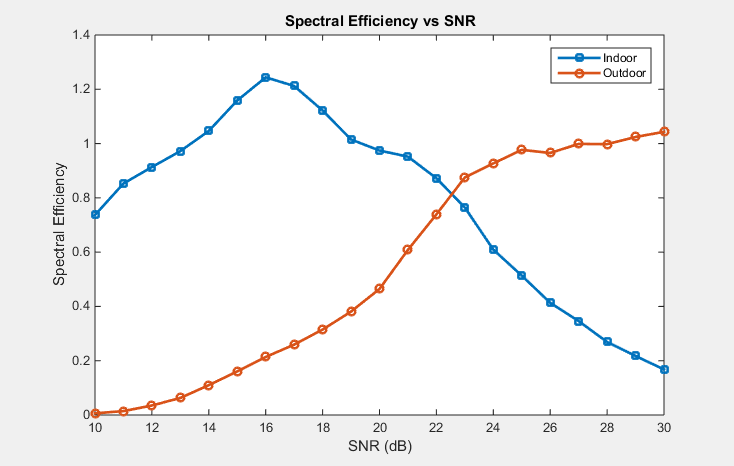


For Multipath Channel simulation, Channel B was used. BER was calculated for indoor and outdoor models for both QPSK and 16-QAM.





The plot of empirical spectral efficiency for the indoor and outdoor channel models using adaptive modulation is shown below



**Matlab Code**

function LTE\_OFDM\_SIMULATION()

clc

close all;

clear all;

sequence = rand(1,4)>0.5; %pseudo binary sequence

pilot = 2.\*sequence - 1; %Pilot signal

range = 10:30; %Adaptive Modulation range

S1 = 100; %100 symbols

S2 = 10; %The number of Monte-Carlo trials for each symbol

AWGN\_SNR\_dB = [3 5 7 9 11];

AWGN\_SNR = 10.^((AWGN\_SNR\_dB)./10); %SNR in linear scale

for i = 1:S1

inputQPSK = rand(1,2\*48)>0.5; %input to QPSK modulator

inputQAM = rand(1,4\*48)>0.5; %input to 16 QAM modulator

for j = 1:length(AWGN\_SNR)

for k = 1:S2

%QPSK Modulation

QPSK\_Modulated = QPSK(1,inputQPSK); %QPSK Modulation

QPSK\_OFDM = Transmitter\_OFDM(QPSK\_Modulated,pilot); %OFDM Transmission

QPSK\_OFDM\_Channel = AWGN(QPSK\_OFDM,AWGN\_SNR(j)); %AWGN Channel

[QPSKDemodulated\_OFDM var] = Receiver\_OFDM(1,QPSK\_OFDM\_Channel,[],[],[]); %OFDM Receiver

QPSK\_Demodulated = QPSK(2,QPSKDemodulated\_OFDM); %QPSK Demodulation

error1(k) = sum(inputQPSK ~= QPSK\_Demodulated)/(48\*2); %bit error probability

%16 QAM Modulation

QAM\_Modulated = QAM(1,inputQAM); %QAM Modulation

QAM\_OFDM = Transmitter\_OFDM(QAM\_Modulated,pilot); %OFDM Transmission

QAM\_OFDM\_Channel = AWGN(QAM\_OFDM,AWGN\_SNR(j)); %AWGN Channel

[QAMDemodulated\_OFDM var] = Receiver\_OFDM(1,QAM\_OFDM\_Channel,[],[],[]); %OFDM Receiver

QAM\_Demodulated = QAM(2,QAMDemodulated\_OFDM); %QAM Demodulation

error2(k) = sum(inputQAM ~= QAM\_Demodulated)/(48\*4); %bit error probability

end

sym\_error1(i,j) = sum(error1)/S2;

sym\_error2(i,j) = sum(error2)/S2;

end

end

for j = 1:length(AWGN\_SNR)

bit\_error1(j) = sum(sym\_error1(:,j))/S1;

ber\_error2(j) = sum(sym\_error2(:,j))/S1;

end

figure

semilogy(AWGN\_SNR\_dB,bit\_error1);

title('BER of QPSK vs SNR in AWGN channel');

xlabel('SNR (dB)');

ylabel('Probability of Error');

figure

semilogy(AWGN\_SNR\_dB,ber\_error2);

title('BER of 16-QAM vs SNR in AWGN channel');

xlabel('SNR dB');

ylabel('Probability of Error');

AWGN\_SNR\_dB = 0:30;

AWGN\_SNR = 10.^((AWGN\_SNR\_dB)./10); %SNR in linear scale

for i = 1:S1

inputQPSK = rand(1,2\*48)>0.5; %input to QPSK modulator

inputQAM = rand(1,4\*48)>0.5; %input to 16 QAM modulator

[tapsIndoor tapsOutdoor] = Multipath\_Taps();

for j = 1:length(AWGN\_SNR)

for k = 1:S2

%QPSK Modulation

QPSK\_Modulated = QPSK(1,inputQPSK); %QPSK Modulation

QPSK\_OFDM = Transmitter\_OFDM(QPSK\_Modulated,pilot); %OFDM Transmission

[QPSK\_Indoor\_OFDM\_Channel QPSK\_Outdoor\_OFDM\_Channel] = Channel\_multipath(QPSK\_OFDM, AWGN\_SNR(j), tapsIndoor, tapsOutdoor); %Multipath Channel

[QPSKDemodulated\_Indoor\_OFDM QPSKDemodulated\_Outdoor\_OFDM] = Receiver\_OFDM(2,QPSK\_Indoor\_OFDM\_Channel,QPSK\_Outdoor\_OFDM\_Channel,tapsIndoor,tapsOutdoor); %OFDM Receiver

QPSKDemodulated\_Indoor = QPSK(2,QPSKDemodulated\_Indoor\_OFDM); %QPSK Demodulation for Indoor

QPSKDemodulated\_Outdoor = QPSK(2,QPSKDemodulated\_Outdoor\_OFDM); %QPSK Demodulation for Outdoor

error\_Indoor1(k) = sum(inputQPSK ~= QPSKDemodulated\_Indoor)/(48\*2); %BER for Indoor

error\_Outdoor1(k) = sum(inputQPSK ~= QPSKDemodulated\_Outdoor)/(48\*2); %BER for Outdoor

%16 QAM Modulation

QAM\_Modulated = QAM(1,inputQAM); %QAM Modulation

QAM\_OFDM = Transmitter\_OFDM(QAM\_Modulated,pilot); %OFDM Transmission

[QAM\_Indoor\_OFDM\_Channel QAM\_Outdoor\_OFDM\_Channel] = Channel\_multipath(QAM\_OFDM, AWGN\_SNR(j), tapsIndoor, tapsOutdoor); %Multipath Channel

[QAMDemodulated\_Indoor\_OFDM QAM\_Demod\_OFDMP] = Receiver\_OFDM(2,QAM\_Indoor\_OFDM\_Channel,QAM\_Outdoor\_OFDM\_Channel,tapsIndoor,tapsOutdoor); %OFDM Receiver

QAMDemodulated\_Indoor = QAM(2,QAMDemodulated\_Indoor\_OFDM); %QAM Demodulation for Indoor

QAMDemodulated\_Outdoor = QAM(2,QAM\_Demod\_OFDMP); %QAM Demodulation for Outdoor

error\_Indoor2(k) = sum(inputQAM ~= QAMDemodulated\_Indoor)/(48\*4); %BER for Indoor

error\_Outdoor2(k) = sum(inputQAM ~= QAMDemodulated\_Outdoor)/(48\*4); %BER for Outdoor

end

per\_Indoor1(i,j) = sum(error\_Indoor1)/S2;

per\_Outdoor1(i,j) = sum(error\_Outdoor1)/S2;

per\_Indoor2(i,j) = sum(error\_Indoor2)/S2;

per\_Outdoor2(i,j) = sum(error\_Outdoor2)/S2;

end

end

for j = 1:length(AWGN\_SNR) %Total Probability

ber\_Indoor1(j) = sum(per\_Indoor1(:,j))/S1;

ber\_Outdoor1(j) = sum(per\_Outdoor1(:,j))/S1;

ber\_Indoor2(j) = sum(per\_Indoor2(:,j))/S1;

ber\_Outdoor2(j) = sum(per\_Outdoor2(:,j))/S1;

end

%QPSK

figure

semilogy(AWGN\_SNR\_dB,ber\_Indoor1);

hold on;

semilogy(AWGN\_SNR\_dB,ber\_Outdoor1);

legend('Indoor','Outdoor',1);

title('BER of QPSK vs SNR in Multipath channel');

xlabel('SNR (dB)');

ylabel('Probability of Error');

%16 QAM

figure

semilogy(AWGN\_SNR\_dB,ber\_Indoor2);

hold on;

semilogy(AWGN\_SNR\_dB,ber\_Outdoor2);

legend('Indoor','Outdoor',1);

title('BER of 16-QAM vs SNR in Multipath channel');

xlabel('SNR (dB)');

ylabel('Probability of Error');

%Adaptive Modulation

Indoor\_SpectralEff = zeros(1,length(range));

Outdoor\_SpectralEff = zeros(1,length(range));

for k = 1:S1

[tapsIndoor tapsOutdoor] = Multipath\_Taps();

[spectralEff\_Indoor spectralEff\_Outdoor] = adaptive\_Modulation(tapsIndoor,tapsOutdoor);

Indoor\_SpectralEff = Indoor\_SpectralEff + spectralEff\_Indoor;

Outdoor\_SpectralEff = Outdoor\_SpectralEff + spectralEff\_Outdoor;

end

Indoor\_SpectralEff = Indoor\_SpectralEff./S1;

Outdoor\_SpectralEff = Outdoor\_SpectralEff./S1;

figure

plot(range,Indoor\_SpectralEff);

hold on;

plot(range,Outdoor\_SpectralEff);

legend('Indoor','Outdoor',2);

title('Spectral Efficiency vs SNR');

xlabel('SNR (dB)');

ylabel('Spectral Efficiency');

end

% Functions:

% QPSK Modulation:

function [output] = QPSK(value, input)

QPSK\_signals = [-1-1i -1+1i 1+1i 1-1i]; %corresponding to [-3pi/4 3pi/4 pi/4, -pi/4]

if (value == 1)

QPSK\_bits = 2.\*input - 1; %mapping the bits to +1 or -1

seperation = reshape(QPSK\_bits,2,length(input)/2); %QPSK in-phase/quadrature-phase seperation

QPSK\_inphase = seperation(1,:); %in-phase component (input) for QPSK

QPSK\_quadrature = seperation(2,:); %quadrature-phase component (input) for QPSK

output = QPSK\_inphase + (1i.\*QPSK\_quadrature);

else

output = zeros(1,length(input)\*2);

QPSK\_distance = [abs(input - QPSK\_signals(1));

abs(input - QPSK\_signals(2));

abs(input - QPSK\_signals(3));

abs(input - QPSK\_signals(4))];

for m = 1:length(input)

QPSK\_symbol = QPSK\_signals(find(QPSK\_distance(:,m)==min(QPSK\_distance(:,m)),1,'first'));

output(2\*m - 1) = (real(QPSK\_symbol)+1)/2;

output(2\*m) = (imag(QPSK\_symbol)+1)/2;

end

end

end

%16-QAM Modulation:

function [output] = QAM(value, input)

QAM\_symbols = (2/sqrt(10)).\*[-3 -1 3 1]; %corresponding to [00 01 10 11] and making Eb = 1

if (value == 1)

QAM\_seperation = reshape(input,4,length(input)/4); %QAM in-phase/quadrature-phase bits seperation

d1 = QAM\_seperation(1,:); %d1 of QAM

d2 = QAM\_seperation(2,:); %d2 of QAM

d3 = QAM\_seperation(3,:); %d3 of QAM

d4 = QAM\_seperation(4,:); %d4 of QAM

QAM\_inphase = zeros(1,length(input)/4);

QAM\_seperation = zeros(1,length(input)/4);

for i = 1:length(input)/4

QAM\_inphase(i) = QAM\_symbols(((2\*d1(i))+d2(i))+ 1);

QAM\_seperation(i) = QAM\_symbols(((2\*d3(i))+d4(i))+ 1);

end

output = QAM\_inphase + (1i.\*QAM\_seperation);

else

output = zeros(1,length(input)\*4);

QAM\_distance = zeros(length(QAM\_symbols)\*4,length(input));

for i = 1:(length(QAM\_symbols)\*4)

bin = dec2bin(i-1,4);

QAM\_distance(i,:) = abs(input - ((QAM\_symbols(bin2dec(bin(1:2))+ 1))+(1i\*(QAM\_symbols(bin2dec(bin(3:4))+ 1)))));

end

for i = 1:length(input)

bin = dec2bin((find(QAM\_distance(:,i)==min(QAM\_distance(:,i)),1,'first')-1),4);

output(4\*i - 3) = str2double(bin(1));

output(4\*i - 2) = str2double(bin(2));

output(4\*i - 1) = str2double(bin(3));

output(4\*i) = str2double(bin(4));

end

end

end

% AWGN Channel:

function [signal\_received] = AWGN(input, awgnSNR)

inputLength = length(input); %Input length

signal\_inphase = real(input) + (1/sqrt(2\*awgnSNR))\*randn(1,inputLength); % In-phase received signal

signal\_quadrature = imag(input) + (1/sqrt(2\*awgnSNR))\*randn(1,inputLength);% Quadrature received signal

signal\_received = signal\_inphase + (1i.\*signal\_quadrature);

end

%Multipath Channel Taps:

function [tapsIndoor tapsOutdoor] = Multipath\_Taps()

sampleDuration = 1/(10\*10^6); %Channel bandwidth = 10MHz

delayIndoor = (10^-9).\*[0 100 200 300 500 700]; %Delay profile for Indoor channel model

relativePower\_IndoordB = [0 -3.6 -7.2 -10.8 -18 -25.2]; %Power profile for indoor channel model

delayOutdoor = (10^-9).\*[0 5 30 45 75 90 105 140 210 230 250 270 275 475 595 690]; %Delay profile for Outdoor Multipath channel model

relativePower\_OutdoordB = [-1.5 -10.2 -16.6 -19.2 -20.9 -20.6 -16.6 -16.6 -23.9 -12 -23.9 -21 -17.7 -24.6 -22 -29.2]; %Power profile for Outdoor Multipath channel model

tapsIndoor = zeros(1,80);

tapsOutdoor = zeros(1,80);

relativePower\_Indoor = 10.^(relativePower\_IndoordB./10);

relativePower\_Outdoor = 10.^(relativePower\_OutdoordB./10);

for i = 1:length(delayIndoor)

tapsIndoor(round(delayIndoor(i)/sampleDuration) + 1) = sqrt(relativePower\_Indoor(i)\*((randn(1,1)^2) + (randn(1,1)^2))/2);

end

for m = 1:length(delayOutdoor)

tapsOutdoor(round(delayOutdoor(m)/sampleDuration) + 1) = sqrt(relativePower\_Outdoor(m)\*((randn(1,1)^2) + (randn(1,1)^2))/2);

end

end

% Multipath Channels:

function [indoorChan outdoorChan] = Channel\_multipath(input, awgnSNR, tapsIndoor, tapsOutdoor)

inputLength = length(input); %Input length

noise\_inphase = (1/sqrt(2\*awgnSNR))\*randn(1,inputLength); %In-phase noise

noise\_quadrature = (1/sqrt(2\*awgnSNR))\*randn(1,inputLength); %Quadrature-phase noise

indoorChan = ifft((fft(input,inputLength).\*fft(tapsIndoor,inputLength)),inputLength) + (noise\_inphase + (1i\*noise\_quadrature));

outdoorChan = ifft((fft(input,inputLength).\*fft(tapsOutdoor,inputLength)),inputLength) + (noise\_inphase + (1i\*noise\_quadrature));

end

% OFDM Transmit:

function [output] = Transmitter\_OFDM(input, pilot)

fft\_size = 64; %Length of fft/ifft

Length\_cp = 16; %Length of cyclic prefix

Begin\_cp = fft\_size - Length\_cp + 1;

End\_cp = fft\_size;

ofdmSequence = [zeros(1,6) input(1:5) pilot(1) input(6:18) pilot(2) input(19:24) zeros(1,1) input(25:30) pilot(3) input(31:43) pilot(4) input(44:48) zeros(1,5)];

inputIFFT = sqrt(fft\_size)\*ifft(ofdmSequence,64); %IFFT

output = [inputIFFT(Begin\_cp:End\_cp) inputIFFT]; %Adding Cyclic Prefix

end

% OFDM Receive:

function [IndoorDetection OutdoorDetection] = Receiver\_OFDM(value,indoor, outdoor, IndoorTaps, OutdoorTaps)

fft\_size = 64; %Length of fft/ifft

outputLength = length(indoor); %Length of received signal (same for both Indoor and Outdoor)

if (value == 1) %value = 1 for AWGN channel

Indoorfft = (1/sqrt(fft\_size))\*fft(indoor(17:80),fft\_size);

IndoorDetection = [Indoorfft(7:11) Indoorfft(13:25) Indoorfft(27:32) Indoorfft(34:39) Indoorfft(41:53) Indoorfft(55:59)];

OutdoorDetection = [];

else

indoor = ifft((fft(indoor,outputLength)./fft(IndoorTaps,outputLength)),outputLength);

outdoor = ifft((fft(outdoor,outputLength)./fft(OutdoorTaps,outputLength)),outputLength);

Indoorfft = (1/sqrt(fft\_size))\*fft(indoor(17:80),fft\_size);

Outdoorfft = (1/sqrt(fft\_size))\*fft(outdoor(17:80),fft\_size);

IndoorDetection = [Indoorfft(7:11) Indoorfft(13:25) Indoorfft(27:32) Indoorfft(34:39) Indoorfft(41:53) Indoorfft(55:59)];

OutdoorDetection = [Outdoorfft(7:11) Outdoorfft(13:25) Outdoorfft(27:32) Outdoorfft(34:39) Outdoorfft(41:53) Outdoorfft(55:59)];

end

end

% Adaptive modulation:

function [spectralEffIndoor spectralEffOutdoor] = adaptive\_Modulation(tapsIndoor,tapsOutdoor)

lengh = 64;

SNRrange = 10:30; %range of average SNR is from 10 to 30dB

noisePower = 1./(10.^((SNRrange)./10));

fftIndoor = fft(tapsIndoor,lengh);

fftOutdoor = fft(tapsOutdoor,lengh);

spectralEffOutdoor = zeros(1,length(SNRrange));

spectralEffIndoor = zeros(1,length(SNRrange));

for i = 1:length(SNRrange)

SNRIndoor = 10\*log10((abs(fftIndoor).^2)./noisePower(i));

SNROutdoor = 10\*log10((abs(fftOutdoor).^2)./noisePower(i));

nbits = 0;

for j = 1:length(SNRIndoor)

if ((SNRIndoor(j) >= 10.61) && (SNRIndoor(j) < 13.94)) %QPSK

nbits = nbits + 2;

else if ((SNRIndoor(j) >= 17.25) && (SNRIndoor(j) < 20.4)) %16 QAM

nbits = nbits + 4;

end

end

end

spectralEffIndoor(i) = nbits/length(SNRIndoor);

nbits = 0;

for j = 1:length(SNROutdoor)

if ((SNROutdoor(j) >= 10.61) && (SNROutdoor(j) < 13.94)) %QPSK

nbits = nbits + 2;

else if ((SNROutdoor(j) >= 17.25) && (SNROutdoor(j) < 20.4)) %16 QAM

nbits = nbits + 4;

end

end

end

spectralEffOutdoor(i) = nbits/length(SNROutdoor);

end

end