1. **Bit Error Rate n Presence of AWGN**

clc

clear all

close all

bit\_count = 100000; %no. of random bits to be generated for a single shot of BER calculation

SNR = 0: 1: 10; %Range of SNR over which to simulate

for k = 1: 1: length(SNR)

tote = 0; %total error bits

totb = 0; %total bits

while tote < 100 %until you get 100 errors

rbits = round(rand(1,bit\_count)); %generate random bits

tx = -2\*(rbits-0.5); % BPSK Modulation: Directly to Bipolar NRZ

N0 = 1/10^(SNR(k)/10); %noise level

rx = tx + sqrt(N0/2)\*(randn(1,length(tx))+i\*randn(1,length(tx)));

rx2 = rx < 0; % BPSK demodulator logic at the Receiver

diff = rbits - rx2; % Calculate Bit Errors

tote = tote + sum(abs(diff)); %total errors

totb = totb + length(rbits); %total bits generated

end

BER(k) = tote / totb; % Calculate Bit Error Rate

end

semilogy(SNR,BER,'\*r');

hold on;

xlabel('Eb/No (dB)');

ylabel('BER');

title('Eb/No(SNR) Vs BER plot for BPSK Modualtion in AWGN Channel');

thber = 0.5\*erfc(sqrt(10.^(SNR/10))); % Theoretical BER

semilogy(SNR,thber);

grid on;

legend('Simulated Curve', 'Theoretical Curve');

1. **DSSS CDMA**

clc;

close all;

clear all;

%input('Enter the inpt Bits:');

b=[1 0 1 0 1 0 1 0 1 0]

ln=length(b);

% Converting bit 0 to -1

for i=1:ln

if b(i)==0

b(i)=-1

end

end

%Generating the bit sequence with each bit 8 sample ling

k=1;

for i= 1:ln

for j=1:8

bb(k)=b(i);

j=j+1;

k=k+1;

end

i=i+1;

end

len=length(bb);

subplot(2,1,1);

stairs(bb,'linewidth',2);

axis([0 len,-2 3]);

title('ORIGINAL BIT SEQUENCE b(t)');

%Generating the pseudorandom bit pattern for spreading

pr\_sig=round(rand(1,len));

for i=1:len

if pr\_sig(i)==0

pr\_sig(i)=-1;

end

end

subplot(2,1,2);

stairs(pr\_sig,'linewidth',2);

axis([0 len -2 3]);

title('PSEUDORANDOM BIT SEQUENCE pr\_sig(t)');

%Multiplying bit sequence with psseudorandom sequence

for i=1:len

bbs(i)=bb(i).\*pr\_sig(i);

end

%Modulating the hopped signal

dsss=[];

t=0:1/10:2\*pi;

c1=cos(t);

c2=cos(t+pi);

len=length(bb);

for k=1:len

if bbs(1,k)== -1

dsss=[dsss c1];

else

dsss=[dsss c2];

end

end

figure;

subplot(2,1,1);

stairs(bbs,'linewidth',2);

axis ([0 len -2 3]);

title('MULTIPLIER OUTPUT SEQUENCE b(t)\*pr\_sig(t)');

subplot(2,1,2);

plot(dsss);

title('DS-SS Signal---');

1. **GMSK**

clc;

clear all;

close all;

DRate=1; %data rate or 1 bit in one second

M=18; %no of smples per bit

N=36; %no of bits for simulation [-18:18]

BT=0.5;%Bandwidth\*period (cannot change)

T=1/DRate; %data period ,i.e 1 bit in one second

Ts=T/M;

k=[-18:18]; %Chens value more than needed;%only introduces a little more delay

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%COSTRUCTION OF GAUSSIAN FILTER FOLLOWED BY SHAPING OF DATA BITS USING

%GAUSSIAN FILTER

alpha=sqrt(log(2))/(2\*pi\*BT);

h=exp(-(k\*Ts).^2/(2\*alpha^2\*T^2))/(sqrt(2\*pi)\*alpha\*T); %Gaussian filter response in time domain

figure(1);

plot(h,'\*r')

title('Response of Gaussian filter');

xlabel('Sample at Ts');

ylabel('Normalized Magnitude');

ipbit=[1 1 0 0 0 1 1 1];

m=filter(h,1,ipbit);

m\_sig=[];

fc=1

fs=10^3;

w=2\*pi\*fc;

T=1

%%Weighting function generator

t=-1:1/fs:7; %advanced inphase %length is 8fs+1

t1=0:1/fs:8; %quadrature %length is 8fs+1

cos\_w=cos(2\*pi\*t/2);%Inphase

sin\_w=sin(2\*pi\*t1/2); %quadrature

ai=ipbit(1:2:end);

aq=ipbit(2:2:end);

m1=filter(h,1,ai);

m2=filter(h,1,aq);

ai\_c=[];

aq\_s=[];

ai\_c=kron(m1(1:2:end),ones(1,4000));

aq\_s=kron(m2(2:2:end),ones(1,4000));

ai\_c=[ai\_c zeros(1,1)]; %to make length=8fs+1

aq\_s=[zeros(1,1) aq\_s];%to make length=8fs+1

aii=ai\_c.\*cos\_w;

aqq=aq\_s.\*sin\_w;

i=aii.\*cos(w\*t);

q=aqq.\*sin(w\*t);

m\_sig=i+q;

figure(2)

subplot(2,1,1)

plot(i);

subplot(2,1,2)

plot(q);

figure(3)

subplot(2,2,1);

plot(m\_sig);

**3.1 MSK Power Spectral Density**

%-------------------------------------------

% matlab/spectra.m

% This program computes power spectra of

% MSK Digital Modulation Methods

%-------------------------------------------

pi = 3.141592;

sqrpi = pi^2;

MSK = [];

xaxis = [];

for i=1:1000

f = i/100;

% f is frequency normalized to to 1/(bit duration)

xaxis = [xaxis, f ];

ymsk = 16/sqrpi \* (cos(6.2832 \* f))^2/ (1- 16 \* f^2)^2;

MSK = [MSK, 10 \* log10(ymsk)];

end

plot(xaxis,MSK, 'r-');

axis([0 10 -60 10]);

ylabel('Spectral Power Level in dB');

xlabel('Frequency Offset / Bit Rate') ;

**3.2 GMSK PSD**

%%%%%%%%%%%% GMSK PSD %%%%%%%%%%%%%%%%%%%%%%%%%%%

BW = 285e3;

BW2 = 186e3;

baseline\_dB = -76;

% baseline\_dB = -999 % disable the constant term

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% empirical GMSK(GMSK narrow-bandwidth pulse) model equation

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

f = (-500e3:1e3:500e3)+1e-3;

gaussPart = exp(-(2\*f/BW) .^2);

sincPart = sin(pi\*f/BW2) ./ (pi\*f/BW2);

flatPart = 10^(baseline\_dB/20);

H\_dB = 20\*log10(abs(gaussPart .\* sincPart) + flatPart);

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% plot the spectrum

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

figure(); grid on; hold on;

h = plot(f/1e6, H\_dB, 'b'); set(h, 'linewidth', 2);

xlabel('f/MHz');

ylabel('PSD/dB');

title('GMSK spectrum');

**4. HATA/Okumura Model**

% Okumura/Hata Model 3

clc;

close all;

clear all;

d = 1:0.01:20;

hm = 5;

hb1 = 30;

hb2 = 100;

hb3 = 200;

fc = 1000;

% a. For Large Cities

% fc >= 400MHz

ahm = 3.2\*(log10(11.75\*hm)).^2 - 4.97;

% A. Typical Urban

L50urban1 = 69.55 + 26.16\*log10(fc) + (44.9 - 6.55\*log10(hb1))\*log10(d) - 13.82\*log10(hb1) - ahm;

L50urban2 = 69.55 + 26.16\*log10(fc) + (44.9 - 6.55\*log10(hb2))\*log10(d) - 13.82\*log10(hb2) - ahm;

L50urban3 = 69.55 + 26.16\*log10(fc) + (44.9 - 6.55\*log10(hb3))\*log10(d) - 13.82\*log10(hb3) - ahm;

% B. Typical Suburban

L50suburban1 = L50urban1 - 2\*(log10(fc/28)).^2 - 5.4;

L50suburban2 = L50urban2 - 2\*(log10(fc/28)).^2 - 5.4;

L50suburban3 = L50urban3 - 2\*(log10(fc/28)).^2 - 5.4;

% C. Typical Rural

L50rural1 = L50urban1 - 4.78\*(log10(fc)).^2 + 18.33\*log10(fc) - 40.94;

L50rural2 = L50urban2 - 4.78\*(log10(fc)).^2 + 18.33\*log10(fc) - 40.94;

L50rural3 = L50urban3 - 4.78\*(log10(fc)).^2 + 18.33\*log10(fc) - 40.94;

figure(1);

plot(d, L50urban1, 'r', d, L50urban2, '--r', d, L50urban3,':r');

hold on;

legend('large urban hb=30', 'large urban hb=100', 'large urban hb=200', 'suburban hb=30', 'suburban hb=100', 'suburban hb=200', 'rural hb=30', 'rural hb=100','rural hb=200');

figure(2);

plot(d, L50suburban1, 'b', d, L50suburban2, '--b', d, L50suburban3, ':b');

hold on;

legend('large urban hb=30', 'large urban hb=100', 'large urban hb=200', 'suburban hb=30', 'suburban hb=100', 'suburban hb=200', 'rural hb=30', 'rural hb=100','rural hb=200');

figure(3);

plot(d, L50rural1, 'g', d, L50rural2, '--g', d, L50rural3, ':g');

hold on;

legend('large urban hb=30', 'large urban hb=100', 'large urban hb=200', 'suburban hb=30', 'suburban hb=100', 'suburban hb=200', 'rural hb=30', 'rural hb=100','rural hb=200');

grid on;

xlabel('d [km]');

ylabel('L [dB]');

title('Hata Model for different base station ant. ht. in different environments');

**5.LOST CALL SYSTEM**

%Data and voice traffic analysis in Lost call System

clc;

clear all;

close all;

N=input('Enter the number of trunks');

A=input('Enter the value of ''A');

kk=1;

%GOS will be used for performance analysis

for n=1:N %For N trunks we will have N grade of service

num=power(A,n)/factorial(n);

den=0;

for k=0:n

den=den+power(A,k)/factorial(k);

end

final(kk)=num/den;

kk=kk+1;

end

disp(final);

n=1:N

stem(n,final());

xlabel('Number of trunks');

ylabel('gos');

hold on; %for plotting all values together

plot(n,final);

hold on;

**6.OFDM MODULATION**

%OFDM===================================

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 data points phase representations

data\_source = randsrc(1, no\_of\_data\_points, 0:M-1);

figure(1)

stem(data\_source); grid on; xlabel('data points'); ylabel('transmitted data phase representation')

title('Transmitted Data "O"')

% 2. Perform QPSK modulation

qpsk\_modulated\_data = pskmod(data\_source, M);

scatterplot(qpsk\_modulated\_data);title('qpsk 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;

% ------------------------------------------

% E: % +++++ RECEIVER +++++

% ------------------------------------------

% 1. Pass the ofdm signal through the channel

recvd\_signal = ofdm\_signal;

% 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));

% 8. Demodulate the data

qpsk\_demodulated\_data = pskdemod(recvd\_serial\_data,M);

scatterplot(qpsk\_modulated\_data);title('qpsk modulated received data')

figure(5)

stem(qpsk\_demodulated\_data,'rx');

grid on;xlabel('data points');ylabel('received data phase representation');title('Received Data "X"')

**7.Speech Coding And Decoding**

clc;

close all;

clear all;

n=input('Enter n value for n-bit PCM system : ');

n1=input('Enter number of samples in a period : ');

L=2^n;

% % Signal Generation

% x=0:1/100:4\*pi;

% y=8\*sin(x); % Amplitude Of signal is 8v

% subplot(2,2,1);

% plot(x,y);grid on;

% Sampling Operation

x=0:2\*pi/n1:4\*pi; % n1 nuber of samples have tobe selected

s=8\*sin(x);

subplot(3,1,1);

plot(s);

title('Analog Signal');

ylabel('Amplitude--->');

xlabel('Time--->');

subplot(3,1,2);

stem(s);grid on; title('Sampled Sinal'); ylabel('Amplitude--->'); xlabel('Time--->');

% Quantization Process

vmax=8;

vmin=-vmax;

del=(vmax-vmin)/L;

part=vmin:del:vmax; % level are between vmin and vmax with difference of del

code=vmin-(del/2):del:vmax+(del/2); % Contaion Quantized valuses

[ind,q]=quantiz(s,part,code); % Quantization process

% ind contain index number and q contain quantized values

l1=length(ind);

l2=length(q);

for i=1:l1

if(ind(i)~=0) % To make index as binary decimal so started from 0 to N

ind(i)=ind(i)-1;

end

i=i+1;

end

for i=1:l2

if(q(i)==vmin-(del/2)) % To make quantize value inbetween the levels

q(i)=vmin+(del/2);

end

end

subplot(3,1,3);

stem(q);grid on; % Display the Quantize values

title('Quantized Signal');

ylabel('Amplitude--->');

xlabel('Time--->');

% Encoding Process

figure

code=de2bi(ind,'left-msb'); % Cnvert the decimal to binary

k=1;

for i=1:l1

for j=1:n

coded(k)=code(i,j); % convert code matrix to a coded row vector

j=j+1;

k=k+1;

end

i=i+1;

end

subplot(2,1,1); grid on;

stairs(coded); % Display the encoded signal

axis([0 100 -2 3]); title('Encoded Signal');

ylabel('Amplitude--->');

xlabel('Time--->');

% Demodulation Of PCM signal

qunt=reshape(coded,n,length(coded)/n);

index=bi2de(qunt','left-msb'); % Getback the index in decimal form

q=del\*index+vmin+(del/2); % getback Quantized values

subplot(2,1,2); grid on;

plot(q); % Plot Demodulated signal

title('Demodulated Signal');

ylabel('Amplitude--->');

**8.TDMA**

clc;

close all;

clear all;

% Signal generation

x=0:.5:4\*pi; % siganal taken upto 4pi

sig1=8\*sin(x); % generate 1st sinusoidal signal

l=length(sig1);

sig2=8\*triang(l); % Generate 2nd traingular Sigal

% Display of Both Signal

subplot(2,2,1);

plot(sig1);

title('Sinusoidal Signal');

ylabel('Amplitude--->');

xlabel('Time--->');

subplot(2,2,2);

plot(sig2);

title('Triangular Signal');

ylabel('Amplitude--->');

xlabel('Time--->');

% Display of Both Sampled Signal

subplot(2,2,3);

stem(sig1);

title('Sampled Sinusoidal Signal');

ylabel('Amplitude--->');

xlabel('Time--->');

subplot(2,2,4);

stem(sig2);

title('Sampled Triangular Signal');

ylabel('Amplitude--->');

xlabel('Time--->');

l1=length(sig1);

l2=length(sig2);

for i=1:l1

sig(1,i)=sig1(i); % Making Both row vector to a matrix

sig(2,i)=sig2(i);

end

% TDM of both quantize signal

tdmsig=reshape(sig,1,2\*l1);

% Display of TDM Signal

figure

stem(tdmsig);

title('TDM Signal');

ylabel('Amplitude--->');

xlabel('Time--->');

% Demultiplexing of TDM Signal

demux=reshape(tdmsig,2,l1);

for i=1:l1

sig3(i)=demux(1,i); % Converting The matrix into row vectors

sig4(i)=demux(2,i);

end

% display of demultiplexed signal

figure

subplot(2,1,1)

plot(sig3);

title('Recovered Sinusoidal Signal');

ylabel('Amplitude--->');

xlabel('Time--->');

subplot(2,1,2)

plot(sig4);

title('Recovered Triangular Signal');

ylabel('Amplitude--->');

xlabel('Time--->');