EXPERIMENT 1
Autocorrelation, PSD, Lowpass, Bandpass

Autocorrelation, PSD

```
echo on
N=1000;
M=75;
Rx_av=zeros(1,M+ 1 );
Sx_av=zeros(1,M+ 1);
for j=1:10, % Take the ensemble average over ten realizations
end;
X=rand(1,N)-1/2; % N i.i.d. uniformly distributed random variables
% between -112 and 112.
Rx=Rx_est(X,M); % autocorrelation of the realization
Sx=fftshift(abs(fft(Rx))); % power spectrum of the realization
Rx_av=Rx_av+Rx;
Sx_av=Sx_av+Sx;
echo off ;
% sum of the autocorrelations
% sum of the spectrums
echo on ;
Rx_av=Rx_av/10;
Sx_av=Sx_av/10;
% Plot Ensemble Average Spectrum
figure;
subplot(2, 1, 1);
plot(Sx_av);
title('Ensemble Average Spectrum');
xlabel('Frequency');
ylabel('Magnitude');

% Plot Ensemble Average Autocorrelation
subplot(2, 1, 2);
plot(Rx_av);
title('Ensemble Average Autocorrelation');
xlabel('Lag');
ylabel('Autocorrelation');

% Plot FFT of Ensemble Average Spectrum
figure;
subplot(2, 1, 1);
Sxy = fftshift(abs(fft(Sx_av)));
plot(Sxy);
title('FFT of Ensemble Average Spectrum');
xlabel('Frequency');
ylabel('Magnitude');

% Plot FFT of Autocorrelation
subplot(2, 1, 2);
xy = fftshift(abs(fft(Rx_av)));
plot(xy);
title('FFT of Ensemble Average Autocorrelation');
xlabel('Frequency');
ylabel('Magnitude');
% ensemble average autocorrelation
```

```
% ensemble average spectrum

function[Rx] = Rx_est(X,M)

N = 1000;
Rx = zeros(1,M+1);
for m=1:M+1
    for n=1:N-m+1
        Rx(m) = Rx(m) + X(n)*X(n+m-1);
    end
    Rx(m) = Rx(m)/(N-m+1);
end
end
```

Low Pass Samples,

```
N = 1000;   % The maximum value of n
M = 50;
Rxav = zeros(1, M + 1);
Ryav = zeros(1, M + 1);
Sxav = zeros(1, M + 1);
Syav = zeros(1, M + 1);

% Perform ensemble averaging over ten realizations
for i = 1:10
    X = rand(1, N) - (1/2);
    Y(1) = 0;

    % Generate Y(n) using a first-order autoregressive process
    for n = 2:N
        Y(n) = 0.9 * Y(n - 1) + X(n);
    end

    % Compute autocorrelations of X(n) and Y(n)
    Rx = Rx_est(X, M);
    Ry = Rx_est(Y, M);

    % Compute power spectra of X(n) and Y(n)
    Sx = fftshift(abs(fft(Rx)));
    Sy = fftshift(abs(fft(Ry)));

    % Accumulate results for ensemble averaging
    Rxav = Rxav + Rx;
    Ryav = Ryav + Ry;
    Sxav = Sxav + Sx;
    Syav = Syav + Sy;
end

% Average over the ten realizations
Rxav = Rxav / 10;
Ryav = Ryav / 10;
Sxav = Sxav / 10;
Syav = Syav / 10;

% Display results
disp('Ensemble Average Autocorrelation of X:');
```

```
disp(Rxav);
disp('Ensemble Average Autocorrelation of Y:');
disp(Ryav);
disp('Ensemble Average Power Spectrum of X:');
disp(Sxav);
disp('Ensemble Average Power Spectrum of Y:');
disp(Syav);

% Plot Ensemble Average Autocorrelations and Power Spectra
figure;

% Plot Ensemble Average Autocorrelations
subplot(2, 1, 1);
plot(0:M, Rxav, 'DisplayName', 'X(n)');
title('Ensemble Average Autocorrelations');
xlabel('Lag');
ylabel('Autocorrelation');
legend;
subplot(2, 1, 2);
plot(0:M, Ryav, 'DisplayName', 'Y(n)');

title('Ensemble Average Autocorrelations');
xlabel('Lag');
ylabel('Autocorrelation');
legend;

figure;
% Plot Ensemble Average Power Spectra
subplot(2, 1, 1);
frequencies = linspace(-0.5, 0.5, M + 1);
plot(frequencies, Sxav, 'DisplayName', 'X(n)');
title('Ensemble Average Power Spectra');
xlabel('Normalized Frequency');
ylabel('Power Spectral Density');
legend;
subplot(2, 1, 2);
plot(frequencies, Syav, 'DisplayName', 'Y(n)');

title('Ensemble Average Power Spectra');
xlabel('Normalized Frequency');
ylabel('Power Spectral Density');
legend;

% Autocorrelation estimation function
function [Rx] = Rx_est(X, M)
    N = length(X);
    Rx = zeros(1, M + 1);

    for m = 1:M + 1
        for n = 1:N - m + 1
            Rx(m) = Rx(m) + X(n) * X(n + m - 1);
        end
        Rx(m) = Rx(m) / (N - m + 1);
    end
end
```

Bandpass Samples,

```
N = 1000;  % number of samples

for i = 1:2:N
    % Your loop body is currently empty. If you have code to include here, add it.
end

m = 0;
sgma = 1;

% Initialize arrays to store Gaussian random variables
X1 = zeros(1, N);
X2 = zeros(1, N);

% Generate Gaussian random variables
for i = 1:2:N
    [X1(i), X1(i + 1)] = gngauss(m, sgma);
    [X2(i), X2(i + 1)] = gngauss(m, sgma);
end

% Initialize filter coefficients
A = [1, -0.9];
B = 1;

% Filter the Gaussian random variables
Xc = filter(B, A, X1);
Xs = filter(B, A, X2);

% Bandpass modulation
fc = 1000 / pi;
band_pass_process = zeros(1, N);

for i = 1:N
    band_pass_process(i) = Xc(i) * cos(2 * pi * fc * i) - Xs(i) * sin(2 * pi * fc
* i);
end

% Determine the autocorrelation and the spectrum of the bandpass process
M = 50;
bpp_autoco = Rx_est(band_pass_process, M);
bpp_spectrum = fftshift(abs(fft(bpp_autoco)));

% Plotting commands
figure;

subplot(2, 1, 1);
plot(0:M, bpp_autoco);
title('Autocorrelation of Bandpass Process');
xlabel('Lag');
ylabel('Autocorrelation');

subplot(2, 1, 2);
frequencies = linspace(-0.5, 0.5, M + 1);
plot(frequencies, bpp_spectrum);
title('Power Spectral Density of Bandpass Process');
xlabel('Normalized Frequency');
ylabel('Power Spectral Density');
```

```
% Function to generate Gaussian random variables
function [gsrvl, gsrv2] = gngauss(m, sgma)
    u = rand;
    z = sgma * (sqrt(2 * log(1 / (1 - u))));
    u = rand;
    gsrvl = m + z * cos(2 * pi * u);
    gsrv2 = m + z * sin(2 * pi * u);
end

% Autocorrelation estimation function
function [Rx] = Rx_est(X, M)
    N = length(X);
    Rx = zeros(1, M + 1);

    for m = 1:M + 1
        for n = 1:N - m + 1
            Rx(m) = Rx(m) + X(n) * X(n + m - 1);
        end
        Rx(m) = Rx(m) / (N - m + 1);
    end

end
```

EXPERIMENT 2
Central Limit theorem

For roll of N die

```
central_limit_theorem(2, 10000);
function central_limit_theorem(N, num_trials)
    % Simulate rolling N fair dice num_trials times
    rolls = zeros(num_trials, N);

    for i = 1:num_trials
        % Roll N fair dice
        rolls(i, :) = randi([1, 6], 1, N);
    end

    % Calculate the sum of each set of N dice rolls
    sums = sum(rolls, 2);

    % Plot the histogram
    figure;
    histogram(sums, 'Normalization', 'probability', 'BinWidth', 1, 'EdgeColor',
'w');

    % Set plot labels and title
    title(sprintf('Central Limit Theorem for Rolling %d Fair Dice %d times', N,
num_trials));
    xlabel('Sum of Dice Rolls');
    ylabel('Probability');
end
```

For toss of N coins

```
central_limit_theorem_coins(100, 10000);
function central_limit_theorem_coins(N, num_tosses)
    % Simulate tossing N fair coins num_tosses times
    tosses = randi([0, 1], num_tosses, N); % 0 represents tails, 1 represents
heads

    % Calculate the sum of each set of N coin tosses
    sums = sum(tosses, 2);

    % Plot the histogram
    figure;
    histogram(sums, 'Normalization', 'probability', 'BinWidth', 1, 'EdgeColor',
'w');

    % Set plot labels and title
    title(sprintf('Central Limit Theorem for Tossing %d Fair Coins %d times', N,
num_tosses));
    xlabel('Heads');
    ylabel('Probability');
end
```

EXPERIMENT 3

ILLUSTRATION OF LOWPASS SAMPLING THEOREM FOR VARIOUS CASES

Undersampling oversampling and critical sampling

```
clc
close all

tfinal = 0.01;
t = 0:0.00001:tfinal;

xanalog = cos(2*pi*400*t) + cos(2*pi*700*t);

subplot(4,1,1);
plot(t, xanalog , 'r-');
xlabel("Time");
ylabel("Amplitude");
title("Analog signal");

% Critical Sampling (fs=2fm)
fs=1400;
tsamp = 0:1/fs:tfinal;
xsampled = cos(2*pi*400*tsamp) + cos(2*pi*700*tsamp);
subplot(4,1,2);
plot(tsamp, xsampled , 'b*-');
xlabel("Time");
ylabel("Amplitude");
title("Critical Sampling(fs=2fm)");

% Under Sampling (fs<2fm)
fs=700;
tsamp = 0:1/fs:tfinal;
xsampled = cos(2*pi*400*tsamp) + cos(2*pi*700*tsamp);
subplot(4,1,3);
plot(tsamp, xsampled , 'b*-');
xlabel("Time");
ylabel("Amplitude");
title("UnderSampling(fs<2fm)");

% Over Sampling (fs>2fm)
fs=2000;
tsamp = 0:1/fs:tfinal;
xsampled = cos(2*pi*400*tsamp) + cos(2*pi*700*tsamp);
subplot(4,1,4);
plot(tsamp, xsampled , 'b*-');
xlabel("Time");
ylabel("Amplitude");
title("OverSampling(fs>2fm)");
```

Time domain and frequency domain

```
clc
close all

tfinal = 0.01;
t = 0:0.00001:tfinal;

xanalog = cos(2*pi*400*t) + cos(2*pi*700*t);

subplot(4,1,1);
plot(t, xanalog , 'r-');
xlabel("Time");
ylabel("Amplitude");
title("Analog signal");

% Critical Sampling (fs=2fm)
fs=1400;
tsamp = 0:1/fs:tfinal;
xsampled = cos(2*pi*400*tsamp) + cos(2*pi*700*tsamp);
subplot(4,1,2);
plot(tsamp, xsampled , 'b*-');
xlabel("Time");
ylabel("Amplitude");
title("Critical Sampling(fs=2fm)");

% Under Sampling (fs<2fm)
fs=700;
tsamp = 0:1/fs:tfinal;
xsampled = cos(2*pi*400*tsamp) + cos(2*pi*700*tsamp);
subplot(4,1,3);
plot(tsamp, xsampled , 'b*-');
xlabel("Time");
ylabel("Amplitude");
title("UnderSampling(fs<2fm)");

% Over Sampling (fs>2fm)
fs=2000;
tsamp = 0:1/fs:tfinal;
xsampled = cos(2*pi*400*tsamp) + cos(2*pi*700*tsamp);
subplot(4,1,4);
plot(tsamp, xsampled , 'b*-');
xlabel("Time");
ylabel("Amplitude");
title("OverSampling(fs>2fm)");
```

EXPERIMENT 5
Uniform and non uniform pcm

uniform pcm

```
t = 0:0.01:10;
a = sin(t);

[sqnr8, aquan8, code8] = u_pcm(a, 8);
[sqnr16, aquan16, code16] = u_pcm(a, 16);

disp('SQNR for N = 8:');
disp(sqnr8);

disp('SQNR for N = 16:');
disp(sqnr16);

plot(t, a, '-', t, aquan8, '-.', t, aquan16, '-', t, zeros(1, length(t)));
legend('Original Signal', '8 Level Quantized Signal', '16 Level Quantized
Signal');
title('Signal and Quantized Versions');
xlabel('Time');
ylabel('Amplitude');

function [sqnr, a_quan, code] = u_pcm(a, n)
    amax = max(abs(a));
    a_quan = a / amax;
    b_quan = a_quan;
    d = 2 / n;
    q = d * (0:n-1) - (n-1)/2*d;

    for i = 1:n
        indices = (q(i) - d/2 <= a_quan) & (a_quan <= q(i) + d/2);
        a_quan(indices) = q(i) * ones(1, sum(indices));
        b_quan(a_quan == q(i)) = (i-1) * ones(1, sum(a_quan == q(i)));
    end

    a_quan = a_quan * amax;
    num_bits = ceil(log2(n));
    code = zeros(length(a), num_bits);

    for i = 1:length(a)
        for j = num_bits:-1:0
            if fix(b_quan(i) / 2^j) == 1
                code(i, (num_bits - j) + 1) = 1;
                b_quan(i) = b_quan(i) - 2^j;
            end
        end
    end

    sqnr = 20 * log10(norm(a) / norm(a - a_quan));
end
```

Non uniform PCM

```
%%
%%
% Algorithm Implementation
function mu_law_quantization()

    % Parameters
    sequence_length = 500;
    mu = 255;

    % Generate random variables from N(0,1) distribution
    input_sequence = randn(1, sequence_length);

    % Quantization levels
    quantization_levels = [16, 64, 128];

    % Plot input-output relation, error, and quantized output for each
quantization level
    for i = 1:length(quantization_levels)

        % Quantize using mu-law nonlinearity
        quantized_sequence = mu_law_quantize(input_sequence,
quantization_levels(i), mu);

        % Plot input-output relation, error, and quantized output
        figure;

        % Input-Output Relation
        subplot(2,1,1);
        plot(input_sequence, quantized_sequence, 'o');
        title(['Input-Output Relation - Quantization Levels: ',
num2str(quantization_levels(i))]);
        xlabel('Input');
        ylabel('Output');

        % Error
        error = input_sequence - quantized_sequence;
        subplot(2,1,2);
        plot(1:sequence_length, error, '-');
        title(['Error - Quantization Levels: ', num2str(quantization_levels(i))]);
        xlabel('Sample');
        ylabel('Error');


        % Determine SQNR
        sqnr = 10 * log10(sum(input_sequence.^2) / sum(error.^2));
        fprintf('SQNR for Quantization Levels %d: %.2f dB\n',
quantization_levels(i), sqnr);
    end

end

% Mu-law quantization function
function quantized_sequence = mu_law_quantize(input_sequence, num_levels, mu)

    % Normalize input sequence
    input_sequence = input_sequence / max(abs(input_sequence));
```

```
    % Mu-law compression
    compressed_sequence = sign(input_sequence) .* log(1 + mu *
abs(input_sequence)) / log(1 + mu);

    % Quantization
    quantized_sequence = round((num_levels - 1) * (compressed_sequence + 1) / 2);

end
```

EXPERIMENT 6
Delta Modulation and ADM

Delta Modulation

```
clc;
clear all;
close all;
a=2;
t=0:2*pi/50:2*pi; % Signal Generation
x=a*sin(t);
l=length(x);
plot(x,'r');
delta=0.2;
%delta1=2*delta;%Apply delta modulation with doubling the step size
%delta2=3*delta;
hold on
xn=0;
for i=1:l;
if x(i)>xn(i)
d(i)=1;
xn(i+1)=xn(i)+delta;
else
d(i)=0; xn(i+1)=xn(i)-delta;
end
end
stairs(xn)
hold on
legend('Analog signal','DM with step size=0.2')
title('DELTA MODULATION')
```

ADM

```
close all
clear all
clc
td = 0.01;
ts = 0.02;
t = 0:td:5;
x = 8*sin(2*pi*t);
delta = 0.1;
figure(1)
plot(t,x);
ADMout = adeltamod(x,delta,td,ts);
figure(2)
plot(t,ADMout);

%The working of the Advanced Delta Modulator is similar to the regular
% Delta Modulator. The only difference is that the amplitude step
% size is variable and it keeps getting doubled if the previous output/s
% don't seem to 'catch up' with the input signal. This problem is
% referred to as 'Slope overload' in textbooks.
% Usage
% ADMout = adeltamod(sig_in, Delta, fs);
% Delta -- min. step size. This will be multiplied 2nX if required
% sig_in -- the signal input, should be a vector
```

```matlab
% td -- the original sampling period of the input signal, sig_in
% ts -- the required sampling period for ADM output. Note that it
% should be an integral multiple of the input signal's period.
% If not, it will be rounded up to the nearest integer.
% Function output: ADMout
function [ADMout] = adeltamod(sig_in, Delta, td, ts)

    if (round(ts/td) >= 2)
    Nfac = round(ts/td); %Nearest integer
    xsig = downsample(sig_in,Nfac);
    Lxsig = length(xsig);
    Lsig_in = length(sig_in);

    ADMout = zeros(Lsig_in); %Initialising output

    cnt1 = 0; %Counters for no. of previous consecutively increasing
    cnt2 = 0; %steps
    sum = 0;
    for i=1:Lxsig

    if (xsig(i) == sum)
    elseif (xsig(i) > sum)
    if (cnt1 < 2)
    sum = sum + Delta; %Step up by Delta, same as in DM
    elseif (cnt1 == 2)
    sum = sum + 2*Delta; %Double the step size after
    %first two increase
    elseif (cnt1 == 3)
    sum = sum + 4*Delta; %Double step size
    else
    sum = sum + 8*Delta; %Still double and then stop
    %doubling thereon
    end
    if (sum < xsig(i))
    cnt1 = cnt1 + 1;
    else
    cnt1 = 0;
    end
    else
    if (cnt2 < 2)
    sum = sum - Delta;
    elseif (cnt2 == 2)
    sum = sum - 2*Delta;
    elseif (cnt2 == 3)
    sum = sum - 4*Delta;
    else
    sum = sum - 8*Delta;
    end
    if (sum > xsig(i))
    cnt2 = cnt2 + 1;
    else
    cnt2 = 0;
    end
    end
    ADMout(((i-1)*Nfac + 1):(i*Nfac)) = sum;
    end
    end
end
```

EXPERIMENT 7
Sigma Delta Modulation and Demodulation

```
clc
clear all
close all
t = -5:0.01:5; %basic time axis
f =2;
w = 2*pi*f;
osr = 250; %can vary
fs1 = w/pi;
fs = fs1*osr;
%% sampling time
ts = -5:(1/fs):5; %sampling times are defined
y = @(t)sin(w.*t); %signal is defined
%% sigma delta quantisation
[u,q] = SDQ(y(ts),ts);
%% reconstruction algorithm
z = 0;
for k = 1:length(ts)
 z = z + q(k).*sinc(w.*(t - ts(k)));
end
c = max(y(t))./max(z); %scaling is done as a consequence of oversampling
z = z.*c;
%% figures
figure(1)
subplot(3,1,1)
plot(t,y(t),'linewidth',2)
title('Original sinal')
xlabel('Time')
ylabel('Amplitude')
subplot(3,1,2)
plot(ts,q)
title('SDQ signal');
xlabel('Time');
ylabel('Amplitude');
subplot(3,1,3)
plot(t,z,'linewidth',2);
title('Reconstructed signal');
xlabel('Time');
ylabel('Amplitude');
figure(2);
plot(t,y(t),'linewidth',2)
hold on
plot(t,z,'linewidth',2);
title('Original vs Reconstructed');
figure(3);
plot(abs(z - y(t)),'linewidth',2);
title('Error');
figure(4);
subplot(3,1,1);
plot(abs(fftshift(fft(y(t)))));
xlabel('Frequency');
ylabel('Amplitude');
title('Spectrum of original signal');
```

```matlab
subplot(3,1,2);
plot(abs(fftshift(fft(q))));
xlabel('Frequency');
ylabel('Amplitude');
title('Spectrum of SDQ');
subplot(3,1,3);
plot(abs(fftshift(fft(z))));
title('Spectrum of recovered signal');
xlabel('Frequency');
ylabel('Amplitude');
%% mse computation
error = immse(z,y(t));
%% function
function [u,q] = SDQ(y,t)
%as per basic equations, models a sigma delta modulator
%% code logic
q = zeros(1,length(t));
u = zeros(1,length(t)); %quantizaton noise/state variable
u(1) = 0.9; %taken 0.9 as in between 0 and 1 for stability (non inclusive)
%recursive equations for SDQ
for k = 2:length(t)
 q(k) = sign(u(k-1) + y(k));
 u(k) = u(k-1) + y(k) - q(k);
end
end
```

EXPERIMENT 8
Generation of Line Codes, PSD, Probability of Error

Generation line codes

```
N=10;
a=floor(2*rand(1,N));
A=5;
Tb=1;
fs=100;

%Unipolar NRZ
U=[];
for k=1:N
    U=[U A*a(k)*ones(1,fs)];
end

%Unipolar RZ
U_rz=[];
for k=1:N
    c=ones(1,fs/2);
    b=zeros(1,fs/2);
    p=[c b];
    U_rz=[U_rz A*a(k)*p];
end

%Polar NRZ
P=[];
for k=1:N
    P=[P ((-1)^(a(k)+1))*A*ones(1,fs)];
end

%Polar RZ
P_rz=[];
for k=1:N
    c=ones(1,fs/2);
    b=zeros(1,fs/2);
    p=[c b];
    P_rz=[P_rz ((-1)^(a(k)+1))*A*p];
end

%Bipolar NRZ
B=[];
count=-1;
for k=1:N
    if a(k)==1
        if count==-1
            B=[B A*a(k)*ones(1,fs)];
            count=1;
        else
            B=[B -A*a(k)*ones(1,fs)];
            count=-1;
        end
    else
        B=[B A*a(k)*ones(1,fs)];
```

```
    end
end

%Bipolar RZ
B_rz=[];
count=-1;
for k=1:N
    if a(k)==1
        if count==-1
            B_rz=[B_rz A*a(k)*ones(1,fs/2) zeros(1,fs/2)];
            count=1;
        else
            B_rz=[B_rz -A*a(k)*ones(1,fs/2) zeros(1,fs/2)];
            count=-1;
        end
    else
        B_rz=[B_rz A*a(k)*ones(1,fs)];
    end
end

%Manchster Code
M=[];
for k=1:N
    c=ones(1,fs/2);
    b=-1*ones(1,fs/2);
    p=[c b];
    M=[M ((-1)^(a(k)+1))*A*p];
end

T=linspace(0,N*Tb,length(U));

figure(1);

subplot(4,1,1);
plot(T,U,'LineWidth',2);
axis([0 N*Tb -6 6]);
title('Unipolar NRZ');
grid on;

subplot(4,1,2);
plot(T,U_rz,'LineWidth',2);
axis([0 N*Tb -6 6]);
title('Unipolar RZ');
grid on;

subplot(4,1,3);
plot(T,P,'LineWidth',2);
axis([0 N*Tb -6 6]);
title('Polar NRZ');
grid on;

subplot(4,1,4);
plot(T,P_rz,'LineWidth',2);
axis([0 N*Tb -6 6]);
title('Polar RZ');
grid on;

figure(2);
```

```
subplot(3,1,1);
plot(T,B,'LineWidth',2);
axis([0 N*Tb -6 6]);
title('Bipolar NRZ');
grid on;

subplot(3,1,2);
plot(T,B_rz,'LineWidth',2);
axis([0 N*Tb -6 6]);
title('Bipolar RZ');
grid on;

subplot(3,1,3);
plot(T,M,'LineWidth',2);
axis([0 N*Tb -6 6]);
title('Manchester Code');
grid on;
```

PSD of Line Codes

```
v = 1; % voltage level of a bit
R = 1; % Bitrate
T = 1/R; % Bit period
f = 0:0.001*R:2*R; % frequency vector in terms of bit rate
f = f + 1e-10; % Otherwise, sin(0)/0 is undefined

% PSD curves are plotted for Bitrate=1bps and Pulse amplitude=1V

% Unipolar NRZ
s_unipolar_nrz = ((v^2*T/4).*(sin(pi.*f*T)./(pi.*f*T)).^2);
s_unipolar_nrz(1) = s_unipolar_nrz(1) + (v^2/4); % corresponds to an impulse
function of weight v^2/4 at f=0 added to s(f) at f=0

% Manchester code
s_manchester = (v^2*T).*((sin(pi.*f*T/2)./(pi.*f*T/2)).^2).*(sin(pi.*f*T/2).^2);

% Polar NRZ
s_polar_nrz = ((v^2*T).*(sin(pi.*f*T)./(pi.*f*T)).^2);

% Bipolar RZ
s_bipolar_rz = (v^2*T/4).*((sin(pi.*f*T/2)./(pi.*f*T/2)).^2).*(sin(pi.*f*T).^2);

% Plotting
figure;
plot(f, (s_unipolar_nrz), '-r', 'LineWidth', 2);
hold on;
plot(f, (s_manchester), '--g', 'LineWidth', 2);
plot(f, (s_polar_nrz), '--b', 'LineWidth', 2);
plot(f, (s_bipolar_rz), '--k', 'LineWidth', 2);

legend('Unipolar NRZ', 'Manchester code', 'Polar NRZ', 'Bipolar RZ/ RZ-AMI');
xlabel('Normalized frequency');
ylabel('Power spectral density (dB)');
title('Power Spectral Densities for Different Modulation Schemes');
grid on;
```

Probability of error

```
E=[0:1:25]; % Eb/N0=SNR of the recieved signal

%Unipolar NRZ
P1=(1/2)*erfc(sqrt(E/2));

%polar NRZ and Manchester code has same Pe for equiprobable 1's and 0's
P2=(1/2)*erfc(sqrt(E));

%Bipolar RZ/ RZ-AMI
P3=(3/4)*erfc(sqrt(E/2));

E=10*log10(E); % SNR in dB

semilogy(E,P1,'-k',E,P2,'-r',E,P3,'-b','LineWidth',2)
legend('Unipolar NRZ','Polar NRZ and Manchester','Bipolar RZ/ RZ-
AMI','Location','best');
xlabel('SNR per bit, Eb/No(dB)');
ylabel('Bit error probality Pe');
```