PSD:

```
%psd
clc;
clear all;
close all
L=32;
Fs=8*L;
voltageLevel=5;
data=rand(10000,1)>0.5;
clk= mod(0:2*numel(data)-1,2).';
clk_sequence=reshape(repmat(clk,1,L).',1,length(clk)*L);
data_sequence=reshape(repmat(data,1,2*L).',1,length(data)*2*L);
unipolar_nrz_l=voltageLevel*data_sequence;
nrz_encoded=voltageLevel*(2*data_sequence-1);
unipolar_rz=voltageLevel*and(data_sequence,not(clk_sequence));
manchester_encoded=voltageLevel*(2*xor(data_sequence,clk_sequence)-1);
figure(1)
subplot(2,1,1)
plot(clk_sequence(1:800));
grid on
title('Clock')
ylim([-1 2]);
subplot(2,1,2)
plot(data_sequence(1:800))
grid on
title('Data')
ylim([-1 2]);
figure(2)
subplot(2,1,1);
plot(unipolar_nrz_l(1:800))
grid on
```

```
title('Unipolar_nrz_l')
ylim([-1 7]);
subplot(2,1,2)
plot(nrz_encoded(1:800));
grid on
title('Bipolar nrzl')
ylim([-6 6])
figure(3)
subplot(2,1,1)
plot(unipolar_rz(1:800));
grid on
title('Unipolar return to zero')
ylim([-1 7])
subplot(2,1,2)
plot(manchester_encoded(1:800))
grid on
title('Manchester Encoded-IEEE 802.3')
ylim([-6,6])
Rb=1;
Tb=1/Rb;
f=0:0.025:2*Rb;
x=f*Tb;
%Power spectral density of polar signal
P=0.25*Tb*sin(sinc(x).*sinc(x));
%Power spectral density of Unipolar signal
P1=0.0625*Tb*(sinc(x).*sinc(x))+0.125*dirac(f);
%Power spectral density Manchester Signal
P2=0.5*Tb*(sinc(x/2)).^2.*(sin(pi*x/2).^2);
%Power spectral density of Bipolar Signal
P3=0.25*Tb*(sinc(x/2)).^2.*(sin(pi*x).^2);
figure(4)
```

```
plot(f,P,'r')

hold on

plot(f,P1,'g')

hold on

plot(f,P2,'b')

hold on

plot(f,P3,'m')

grid on

box on

xlabel('frequency as a multiple of Bitrate(fRb)--->')

ylabel('Power Spectral Density--->')

title('PSD for various Binary LIne Codes')

legend('PSD for Polar Signal','PSD for unipolar Signal','PSD for Manchester Signal','PSD for Bipolar Signal')
```

BER:

```
%ber
clc;
clear all;
close all;
ac=1; fc=8; b = 10;
bs = input('Enter the message bits: ')
t=0.001:0.001:b;
%MODULATION
%ASK
for i=1:b
mt((i-1)*1000+1:i*1000)=bs(i);
end
ct=ac*cos(2*pi*fc.*t);
st=mt.*ct;
snr = 10;
rt=awgn(st, snr);
for i=1:b
x=sum(st((i-1)*1000+1:i*1000).*ct((i-1)*1000+1:i*1000));
if (x/1000) > 0
d((i-1)*1000+1:i*1000)=1;
else
d((i-1)*1000+1:i*1000)=0;
end
end
figure (1)
subplot(4,1,1)
plot(t,mt)
title('Modulating signal')
xlabel('Time(sec)')
ylabel('Amplitude(volts)')
```

```
subplot(4,1,2)
plot(t,st)
title('ASK Modulated Signal')
xlabel('Time(sec)')
ylabel('Amplitude(volts)')
subplot(4,1,3)
plot(t,rt)
title('Noise Introduced Signal')
xlabel('Time(sec)')
ylabel('Amplitude(volts)')
subplot(4,1,4)
plot(t,d)
title('Demodulated Signal')
xlabel('Time(sec)')
ylabel('Amplitude(volts)')
%BFSK
for i=1:b
mt((i-1)*1000+1:i*1000)=bs(i);
end
ct = ac*cos(2*pi*fc.*t);
ct1= cos(2*pi*fc*(mt+1).*t);
st= ac*cos(2*pi*fc*(mt+1).*t);
snr = 10;
rt= awgn(st, snr);
for i=1:b
x=sum(st((i-1)*1000+1: i*1000).*ct((i-1)*1000+1: i*1000));
if (x/1000)>0.5
d((i-1)*1000+1:i*1000)=0;
else
d((i-1)*1000+1:i*1000)=1;
end
```

```
end
figure(2)
subplot(4,1,1)
plot(t,mt)
title('Modulating signal')
xlabel('Time(sec)')
ylabel('Amplitude(volts)')
subplot(4,1,2)
plot(t,st)
title('FSK Modulated Signal')
xlabel('Time(sec)')
ylabel('Amplitude(volts)')
subplot(4,1,3)
plot(t,rt)
title('Noise Introduced Signal')
xlabel('Time(sec)')
ylabel('Amplitude(volts)')
subplot(4,1,4)
plot(t,d)
title('Demodulated Signal')
xlabel('Time(sec)')
ylabel('Amplitude(volts)')
%BPSK
for i=1:b
if (bs(i)==0)
bs(i) = -1;
else
bs(i)= 1;
end
end
for i=1:b
```

```
mt((i-1)*1000+1:i*1000)=bs(i);
end
ct=ac*sin(2*pi*fc.*t);
st=mt.*ct;
snr = 10;
rt =awgn(st, snr);
for i=1:b
x=sum(st((i-1)*1000+1:i*1000).*ct((i-1)*1000+1:i*1000));
if(x/1000)>0
d((i-1)*1000+1:i*1000)=1;
else
d((i-1)*1000+1:i*1000)=-1;
end
end
figure(3)
subplot(4,1,1)
plot(t,mt)
title('Modulating signal')
xlabel('Time(sec)')
ylabel('Amplitude(volts)')
subplot(4,1,2)
plot(t,st)
title('PSK Modulated Signal')
xlabel('Time(sec)')
ylabel('Amplitude(volts)')
subplot(4,1,3)
plot(t,rt)
title('Noise Introduced Signal')
xlabel('Time(sec)')
ylabel('Amplitude/volts)')
subplot(4,1,4)
```

```
plot(t,d)
title('Demodulated Signal')
xlabel('Time(sec)')
ylabel('Amplitude(volts)')
%BIT-ERROR-RATE
%ASK
num_bit = 1000;
BER_iter = 20;
Eb=1;
SNRdB=0:0.2:10;
SNR=10.*(SNRdB/10);
for count=1:length(SNR)
avgError=0;
No=Eb/SNR(count);
for run_time=1:BER_iter
Error=0;
data= randi([0 1],1,num_bit);
Y=awgn(complex(data),SNRdB(count));
for k=1:num_bit
if ((Y(k)>0.5 && data(k)==0)||(Y(k)<0.5 && data(k)==1))
Error=Error+1;
end
end
Error =Error/num_bit;
avgError=avgError+Error;
end
BER_sim(count)= avgError/BER_iter;
end
figure (4)
semilogy(SNRdB,BER_sim,'g', 'linewidth',2.5);
grid on;
```

```
hold on;
BER_th= (1/2)*erfc(0.5*sqrt(SNR));
semilogy(SNRdB,BER_th,'r', 'linewidth', 2.5)
title('For Bit Error Rate verses SNR for ASK modulation');
grid on; hold on;
xlabel ('SNR(dB)');
ylabel('BER');
legend('Simulation, Theoretical')
%BFSK
for count=1:length(SNR)
avgError=0;
No=Eb/SNR(count);
for run_time=1:BER_iter
Error=0;
data = randi ([0 1],1,num_bit);
s=data+ j*(~data);
Nimg = sqrt(No/2)*randn(1,num_bit);
Nreal = sqrt(No/2)*randn(1,num_bit);
N = Nimg+ j*Nreal;
Y = s+N;
for k=1:num_bit
Z(k)=real(Y(k))-imag(Y(k));
if ((Z(k)>0 && data(k)==0) || (Z(k) <0 && data(k)==1))
Error=Error+1;
end
end
Error=Error/num_bit;
avgError=avgError+Error;
end
BER_sim(count)=avgError/BER_iter;
end
```

```
figure (5)
semilogy(SNRdB,BER_sim,'g', 'linewidth',2.5)
grid on;hold on;
BER_th=(1/2)*erfc(sqrt(SNR/2));
semilogy(SNRdB,BER_th, 'r', 'linewidth',2.5);
grid on;hold on;
title('Curve for Bit Error Rate verses SNR for BFSK modulation');
xlabel('SNR(dB)'); ylabel('BER');
legend('Simulation', 'Theoretical')
%BPSK
for count= 1:length(SNR)
avgError=0;
No=Eb/SNR(count);
for run_time=1:BER_iter
Error=0;
data = randi([0 1],1,num_bit);
s=2*data-1;
N = sqrt(No/2)*randn(1,num_bit);
Y = s+N;
for k=1:num_bit
if ((Y(k)>0 && data(k)==0) || (Y(k) < 0 && data(k)==1))
Error=Error+1;
end
end
Error=Error/num_bit;
avgError=avgError+Error;
end
BER_sim(count)= avgError/BER_iter;
end
figure (6)
semilogy(SNRdB,BER_sim,'g', 'linewidth',2.5);
```

```
grid on; hold on;
BER_th =(1/2)*erfc(sqrt(SNR));
semilogy(SNRdB,BER_th, 'r', 'linewidth',2.5);
grid on;hold on;
title(' Curve for Bit Error Rate verses SNR for BPSK modulation');
xlabel('SNR(dB)');
ylabel('BER'); legend('Simulation', 'Theoretical')
```

VITERBI:

```
%viterbi
clc;
close all;
clear vars;
m=input('Enter the message bits');
m1=[m,0,0];
s1=0;
s2=0;
s3=0;
u=[];
l=4;
k=6;
for i=m1
s3=s2;
s2=s1;
s1=i;
u(end+1)=bitxor(bitxor(s1,s2),s3);
u(end+1)=bitxor(s1,s3);
end
disp('The Encoded Code Word is: ');
disp(u)
trellis=poly2trellis(3,[6,7]);
decoded_msg=vitdec(u,trellis,4,'trunc','hard');
disp('Decoded using inbuilt functions');
disp(decoded_msg(1:4));
for i=1:k
if(i==4)
u(i)=~u(i);
end
end
```

```
disp('The received code word with one bit error is : ');
disp(u);
path_metric_1(1)=0;
path_metric_2(1)=1000;
path_metric_3(1)=1000;
path_metric_4(1)=1000;
u=[u,0,0,0,0]
for n=1:l
bm11=sum(abs([u(2*n-1),u(2*n)]-[0,0]));
bm13=sum(abs([u(2*n-1),u(2*n)]-[1,1]));
bm21=sum(abs([u(2*n-1),u(2*n)]-[1,1]));
bm23=sum(abs([u(2*n-1),u(2*n)]-[0,0]));
bm32=sum(abs([u(2*n-1),u(2*n)]-[1,0]));
bm34=sum(abs([u(2*n-1),u(2*n)]-[0,1]));
bm42=sum(abs([u(2*n-1),u(2*n)]-[0,1]));
bm44=sum(abs([u(2*n-1),u(2*n)]-[1,0]));
pm1_1=path_metric_1(n)+bm11;
pm1_2=path_metric_2(n)+bm21;
pm2_1=path_metric_3(n)+bm32;
pm2_2=path_metric_4(n)+bm42;
pm3_1=path_metric_1(n)+bm13;
pm3_2=path_metric_2(n)+bm23;
pm4_1=path_metric_3(n)+bm34;
pm4_2=path_metric_4(n)+bm44;
if pm1_1<=pm1_2
path_metric_1(n+1)=pm1_1;
tb_path(1,n)=0;
else
path_metric_1(n+1)=pm1_2;
tb_path(1,n)=1;
end
```

```
if pm2_1<=pm2_2
path_metric_2(n+1)=pm2_1;
tb_path(2,n)=0;
else
path_metric_2(n+1)=pm2_2;
tb_path(2,n)=1;
end
if pm3_1<=pm3_2
path_metric_3(n+1)=pm3_1;
tb_path(3,n)=0;
else
path_metric_3(n+1)=pm3_2;
tb_path(3,n)=1;
end
if pm4_1<=pm4_2
path_metric_4(n+1)=pm4_1;
tb_path(4,n)=0;
else
path_metric_4(n+1)=pm4_2;
tb_path(4,n)=1;
end
end
[last_pm,last_state]=min([path_metric_1(n+1),path_metric_2(n+1),path_metric_3(n+1),path_
metric_4(n+1)]);
m=last_state;
for n=l:-1:1
if(m==1)
if tb_path(m,n)==0
decoded(n)=0;
m=1;
elseif(tb_path(m,n)==1)
```

```
decoded(n)=0;
m=2;
end
elseif(m==2)
if tb_path(m,n)==0
decoded(n)=0;
m=3;
elseif(tb_path(m,n)==1)
decoded(n)=0;
m=4;
end
elseif(m==3)
if tb_path(m,n)==0
decoded(n)=1;
m=1;
elseif(tb_path(m,n)==1)
decoded(n)=1;
m=2;
end
elseif(m==4)
if tb_path(m,n)==0
decoded(n)=1;
m=3;
elseif(tb_path(m,n)==1)
decoded(n)=1;
m=4;
end
end
end
disp('Decoded without using built in functions ');
disp('The corrected dataword is: ');
```

```
disp(decoded);
```

DSSS:

```
%dsss
clc;
clear all;
close all;
m = input('Enter the 4 bit input: ');
t = 0:0.01:28;
PN = [];
s1 = 1;
s2 = 0;
s3 = 0;
for i = 1:7
   PN = [PN s3];
   ss3 = s2;
   ss2 = s1;
   ss1 = xor(s1,s3);
   s1 = ss1;
   s2 = ss2;
   s3 = ss3;
end
for i = 1:7
   if (PN(i) == 0)
      PN(i) = -1;
   end
end
PN1 = repmat(repelem(PN, 100), 1, 4);
figure(1);
subplot(3,1,1);
plot(t(1:2800), PN1);
axis([0 28 -1.3 1.3]);
grid on;
```

```
box on;
xlabel('Time');
ylabel('Amplitude');
title('Pseudo Noise Sequence');
m1 = [];
for i = 1:4
    if (m(i) == 0)
        m1 = [m1 repmat(-1,1,7)];
    else
        m1 = [m1 ones(1,7)];
    end
end
m2 = repelem(m1,100);
figure(1);
subplot(3,1,2);
plot(t(1:2800), m2);
axis([0 28 -1.3 1.3]);
grid on;
box on;
xlabel('Time');
ylabel('Amplitude');
title('Input Sequence');
d = [];
for i = 1:7:28
    for j = 1:7
        d = [d m2(i+j-1)*PN(j)];
    end
end
d1 = repelem(d,100);
figure(1);
subplot(3,1,3);
```

```
plot(t(1:2800), d1);
axis([0 28 -1.3 1.3]);
grid on;
box on;
xlabel('Time');
ylabel('Amplitude');
title('Input Sequence multiplied by PN Sequence');
t = 0:0.01:56;
b = repelem(d1, 2);
figure(2);
subplot(3,1,1);
plot(t(1:5600), b);
axis([0 56 -1.3 1.3]);
grid on;
box on;
xlabel('Time');
ylabel('Amplitude');
title('Input Bits');
c = sin(2*pi*t);
figure(2);
subplot(3,1,2);
plot(t, c);
axis([0 56 -1.3 1.3]);
grid on;
box on;
xlabel('Time');
ylabel('Amplitude');
title('Carrier Signal');
op = c(1:5600).*b;
figure(2);
subplot(3,1,3);
```

```
plot(t(1:5600), op);

axis([0 56 -1.3 1.3]);

grid on;

box on;

xlabel('Time');

ylabel('Amplitude');

title('DSSS Wave');
```

FHSS:

```
%fhss
clc;
close all;
clear all;
sequence = input('enter the input bit sequence: ');
i=length(sequence);
input_signal=[];
carrier_signal=[];
time=[0:2*pi/119:2*pi];
for k=1:i
if sequence(1,k)==0
sig=-ones(1,120);
else
sig=ones(1,120);
end
c=cos(time);
carrier_signal = [carrier_signal c];
input_signal = [input_signal sig];
end
figure();
subplot(4,1,1);
plot(input_signal,'k','linewidth',1);
axis([-100 2400 -1.5 1.5]);
title('Input Sequence');
grid on;
bpsk_mod_signal=input_signal.*carrier_signal;
subplot(4,1,2);
plot(bpsk_mod_signal,'k','linewidth',1);
axis([-100 2400 -1.5 1.5]);
title('BPSK Modulated Signal');
```

```
grid on;

time1=[0:2*pi/9:2*pi];

time2=[0:2*pi/19:2*pi];

time3=[0:2*pi/29:2*pi];

time4=[0:2*pi/39:2*pi];

time5=[0:2*pi/59:2*pi];

time6=[0:2*pi/119:2*pi];

carrier1=cos(time1);

carrier1=[carrier1 carrier1 carrier1 carrier1 carrier1 carrier1 carrier1 carrier1 carrier1 carrier1
carrier1 carrier1];

carrier2= cos(time2);

carrier2=[carrier2 carrier2 carrier2 carrier2 carrier2 carrier2];

carrier3=cos(time3);

carrier3=[carrier3 carrier3 carrier3 carrier3];

carrier4=cos(time4);

carrier4 =[carrier4 carrier4 carrier4];

carrier5= cos(time5);

carrier5=[carrier5 carrier5];

carrier6=cos(time6);

spread_signal=[];

for n=1:20

c=randi([1 6],1,1);

switch(c)

 case(1)

spread_signal=[spread_signal carrier1];

case(2)

spread_signal=[spread_signal carrier2];

case(3)

spread_signal=[spread_signal carrier3];

case(4)

spread_signal=[spread_signal carrier4];
```

```
case(5)
spread_signal=[spread_signal carrier5];
case(6)
spread_signal =[spread_signal carrier6];
end
end
subplot(4,1,3);
plot([1:2400],spread_signal,'k','linewidth',1);
axis([-100 2400 -1.5 1.5]);
title('Spread Signal with 6 frequencies');
grid on;
freq_hopped=bpsk_mod_signal.*spread_signal;
subplot(4,1,4);
plot([1:2400],freq_hopped,'k','linewidth',1);
axis([-100 2400 -1.5 1.5]);
title('Frequency Hopped Spread Spectrum Signal');
grid on;
bpsk_demodulated=freq_hopped./spread_signal;
figure();
subplot(2,1,1);
plot([1:2400],bpsk_demodulated,'k','linewidth',1);
axis([-100 2400 -1.5 1.5]);
title('Demodulated BPSK Signal from Wide Spread');
grid on;
original_signal=bpsk_demodulated./carrier_signal;
subplot(2,1,2);
plot([1:2400],original_signal,'k', 'linewidth',1);
axis([-100 2400 -1.5 1.5]);
title('Transmitted Original Bit Sequence');
grid on;
```

EARLY LATE GATE:

```
%early late gate

clock_period = 1;

sample_delay = 0.2 * clock_period;

sampling_frequency = 1000;

t = 0:0.01:10*clock_period;

clock_signal = square(2*pi*t/clock_period, 50);

received_signal =sin(2*pi*t/clock_period);

early_clock = square(2*pi*(t -sample_delay)/clock_period, 50);

late_clock = square(2*pi*(t +sample_delay)/clock_period, 50);

early_output = early_clock .*received_signal;

late_output = late_clock .*received_signal;

early_integral = trapz(t,early_output);

late_integral = trapz(t,late_output);

disp(['Early Integral: ',num2str(early_integral)]);

disp(['Late Integral: ',num2str(late_integral)]);

sampling_instants =0:1/sampling_frequency:t(end);

early_samples = interp1(t,early_output, sampling_instants,'linear', 0);

late_samples = interp1(t,late_output, sampling_instants,'linear', 0);

early_magnitude = abs(early_samples);

late_magnitude = abs(late_samples);

combined_magnitude = early_magnitude+ late_magnitude;

combined_magnitude =interp1(sampling_instants,combined_magnitude, t, 'linear', 0);

modulated_clock_signal = clock_signal.* (1 + combined_magnitude);

figure;

subplot(4,1,1);

plot(t, clock_signal);

title('Clock Signal');

subplot(4,1,2);

plot(t, received_signal);

title('Received Signal');
```

```
subplot(4,1,3);
plot(t, early_output);
title('Early Output (Multiplied with Received Signal)');
subplot(4,1,4);
plot(t, late_output);
title('Late Output (Multiplied with Received Signal)');
figure;
subplot(3,1,1);
stem(sampling_instants,early_samples);
title('Sampled Early Output');
subplot(3,1,2);
stem(sampling_instants,late_samples);
title('Sampled Late Output');
subplot(3,1,3);
plot(t, modulated_clock_signal);
title('Modulated Clock Signal');
```

TAPPED DELAY EQUALIZER:

```
%tapped delay equalizer
clc;
clear all;
close all;
numTaps = 5;
channelDelay = 5;
SNR = 20;
symbolRate = 10e3;
sincDuration = 2;
numSamples = symbolRate * sincDuration;
t = linspace(-2, sincDuration, numSamples);
sincSignal = sinc(10 * t);
channel = zeros(1, channelDelay + 1);
channel(channelDelay + 1) = 1;
receivedSignal = filter(channel, 1, sincSignal);
receivedSignal = awgn(receivedSignal, SNR, 'measured');
equalizerOutput = zeros(1, numSamples);
for i = numTaps + 1:numSamples
 equalizerOutput(i) = receivedSignal(i - numTaps:i) * sincSignal(i - numTaps:i).';
end
figure;
subplot(2, 1, 1);
plot(t, receivedSignal, 'b', 'LineWidth', 1.5);
title('Received Sinc Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
subplot(2, 1, 2);
plot(t, equalizerOutput, 'r', 'LineWidth', 1.5);
title('Signal after Tapped Delay Equalization');
```

```
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
```

WIRELESS FADING CHANNEL:

```
%wireless channel modelling
N = 1000;
E0 = 1;
fc = 2e9;
Cn = rand(1, N);
Cn = Cn / sum(Cn);
t = linspace(0, 1, 1000);
Tc = zeros(size(t));
Ts = zeros(size(t));
for n = 1:N
 phase_n = rand * 2 * pi;
 Tc = Tc + E0 * Cn(n) * cos(2 * pi * fc * t + phase_n);
 Ts = Ts + E0 * Cn(n) * sin(2 * pi * fc * t + phase_n);
end
Ez_field = Tc .* cos(2 * pi * fc * t) - Ts .* sin(2 * pi * fc * t);
v = 30;
angle_of_arrival_deg = 30;
angle_of_arrival_rad = deg2rad(angle_of_arrival_deg);
c = 3e8;
doppler_shift = (v / c) * fc * cos(angle_of_arrival_rad);
N = 10^6;
x = randn(1, N);
y = randn(1, N);
z = (x + 1i * y);
zBin = [0:0.01:7];
sigma2 = 1;
pzTheory = (zBin / sigma2) .* exp(-(zBin.^2) / (2 * sigma2));
[nzSim, zBinSim] = hist(abs(z),zBin);
thetaBin = [-pi:0.01:pi];
pThetaTheory = 1 / (2 * pi) * ones(size(thetaBin));
```

```
[nThetaSim, thetaBinSim] = hist(angle(z), thetaBin);

figure;

subplot(2, 1, 1);

plot(zBinSim, nzSim / (N * 0.01), 'm', 'LineWidth', 2);

hold on;

plot(zBin, pzTheory, 'b.-');

xlabel('z');

ylabel('Probability Density, p(z)');

legend('Simulation', 'Theory');

title('Probability Density Function of |z|');

axis([0 7 0 0.7]);

grid on;

subplot(2, 1, 2);

plot(thetaBinSim, nThetaSim / (N *0.01), 'm', 'LineWidth', 2);

hold on;

plot(thetaBin, pThetaTheory, 'b.-');

xlabel('θ');

ylabel('Probability Density, p(θ)');

legend('Simulation', 'Theory');

title('Probability Density Function of θ');

axis([-pi pi 0 0.2]);

grid on;

Fc = 0;

Fm = doppler_shift;

fs = 1 / (t(2) - t(1));

f_axis = linspace(-fs / 2, fs / 2, length(t));

frequency_response = (1.5 / (pi * Fm)) * sqrt(1 - ((f_axis - Fc) / Fm).^2);

filtered_signal = ifft(fft(Ez_field).* fftshift(frequency_response));

figure;

plot(t, abs(filtered_signal));

xlabel('Time (s)');
```

```
ylabel('Received Signal Amplitude(r)');
title('Received Signal (Magnitude of Ez\_field)');
figure;
plot(t, doppler_shift * ones(size(t)), 'r--');
xlabel('Time (s)');
ylabel('Doppler Shift (Hz)');
title('Doppler Shift Over Time');
psd_filtered = (1 / (fs * length(filtered_signal))) * abs(fft(filtered_signal)).^2;
f_axis_filtered = linspace(-fs / 2, fs / 2, length(psd_filtered));
figure;
plot(f_axis_filtered, psd_filtered);
xlabel('Frequency (Hz)');
ylabel('Power/Frequency (dB/Hz)');
title('Power Spectrum of Filtered Received Signal');
N = 1000;
E0 = 1;
fc = 2e9;
fs = 1000;
t = linspace(0, 1, fs);
num_waveforms = 3;
cross_corr_matrix = zeros(num_waveforms, num_waveforms);
rt_waveforms = cell(num_waveforms, 1);
for waveform_idx = 1:num_waveforms
 Tc = zeros(size(t));
 Ts = zeros(size(t));
 for n = 1:N
 phase_n = rand * 2 * pi;
 Tc = Tc + E0 * rand * cos(2 * pi * fc * t + phase_n);
 Ts = Ts + E0 * rand * sin(2 * pi * fc * t + phase_n);
 end
 Ez_field = Tc .* cos(2 * pi * fc * t) - Ts .* sin(2 * pi * fc * t);
```

```
 Fc = 0;
 Fm = doppler_shift;
 f_axis = linspace(-fs / 2, fs / 2, length(t));
 frequency_response = (1.5 / (pi * Fm)) * sqrt(1 - ((f_axis - Fc) / Fm).^2);
 filtered_signal = ifft(fft(Ez_field) .* fftshift(frequency_response));
 rt_waveforms{waveform_idx} = abs(filtered_signal);
end
for i = 1:num_waveforms
 for j = 1:num_waveforms
 cross_corr = xcorr(rt_waveforms{i}, rt_waveforms{j});
 cross_corr_matrix(i, j) = max(cross_corr);
 end
end
disp('Cross-Correlation Matrix:');
disp(cross_corr_matrix);
```