Introduction to Digital Communication

EEC 382

Final Project Lab

Performance of Matched filters and correlators & Line Codes

Names	IDs
Mohammed Eid Abdelmeguid Ibrahim	19016463
Seif Aldin Zakaria Mohammed Farghal	19015813
Tahany Yasser Morsy Ahmed	18010503

May 17, 2023

Alexandria University
Faculty of Engineering
Communication and Electronics Department

Performance of Matched filters and correlators

Matched filters and correlators are widely used in digital communication systems for signal detection and recovery. The performance of these filters and correlators is crucial in determining the overall performance of the communication system.

The matched filter is a linear filter that maximizes the signal-to-noise ratio (SNR) of the received signal. It is designed to be matched to the transmitted signal and is used to recover the transmitted signal in the presence of noise. The matched filter can be implemented using either an analog or digital filter, and it is used in a variety of communication systems such as pulse amplitude modulation (PAM), phase-shift keying (PSK), and frequency-shift keying (FSK).

The correlator is a digital signal processing technique that is used to detect the presence of a known signal in the presence of noise. It is designed to measure the degree of similarity between a reference signal and the received signal. The correlator can be used for signal detection, synchronization, and tracking. It is widely used in spread spectrum communication systems such as code-division multiple access ($CDMA$).

The performance of the matched filter and correlator is typically evaluated using metrics such as bit error rate (BER) and signal-to-noise ratio (SNR). The BER is a measure of the number of bit errors that occur in the received signal, and the SNR is a measure of the quality of the received signal. A higher SNR indicates better performance, while a lower BER indicates better performance.

The next lines are representing our code in MATLAB :

```
clear, clc, close all;
tic;  % start timer

%% Simulation parameters
num_bits = 1e5;                        % Number of bits
SNR = 0:2:30;                          % SNR range in dB
% Prompt user for input of m or set default value to 20
default_m = 20;
prompt = sprintf('Enter the number of samples representing waveform (default is %d):
user_m = input(prompt);

% If user enters a value, use it, otherwise use the default value
if ~isempty(user_m)
    m = user_m;
else
    m = default_m;
end
n = num_bits*m;                        % Total number of samples
T = 1;                                 % Symbol duration
t = linspace(0,T,m);                   % Time vector

% Generate rectangular pulse s1(t)
prompt = 'Enter s1(t) as an amplitude number (default = 1): ';
s = input(prompt);
if isempty(s)
     s1 = ones(1,m);
else
                 s1 = s * ones(1,m);
end

% Generate zero signal s2(t)
```

```
prompt = 'Enter s2(t) as an amplitude number (default = 0): ';
s = input(prompt);
if isempty(s)
    s2 = zeros(1,m);
else
                s2 = s * ones(1,m);
end

%% Generate random binary data vector
data = randi([0 1],1,num_bits);

%% Modulate data with waveform
waveform = repelem(data,m);

%% Add noise to samples
ber = zeros(3,length(SNR));

for iSNR = 1:length(SNR)
    snr = SNR(iSNR);

    % Calculate transmitted signal power and display it
    TxPower = sum(abs(waveform).^2)/length(waveform);
    if snr == 0
        fprintf('Transmitted Power = %.3f Watts(W)\n',TxPower);
    end

    % Add white Gaussian noise to waveform
    noisyWF = awgn(waveform, snr, 'measured');
    % Add noise to data bits
    xNoisy = awgn(data, snr, 'measured');

                % simple detector

                TH = (max(s1) + min(s2))/2;
                detected_bits = noisyWF(1:m:end) > TH;
                [~, ratio] = biterr(data, detected_bits);
    ber(1,iSNR) = ratio;

    % Apply convolution process in the receiver
    % Response of matched filter
    diff = s1 - s2;
    g = diff(end:-1:1);                  % reflection and shift with t=T

    received = zeros(1,length(data));
    convOP = zeros(1, (2*m-1)*length(data));

    for i = 0:length(data)-1
        noisyWF_20 = noisyWF((i*m)+1:(i+1)*m);        % Extracting 20 samples
        c = conv(noisyWF_20,g);
        % Concatenating the conv results
        convOP( (length(g)+length(noisyWF_20)-1)*i+1:(length(g)...
        +length(noisyWF_20)-1)*i+length(c) ) = c;
        k = m + (length(g)+length(noisyWF_20)-1)*(i);          % middle sample index
        received(i+1) = convOP(k);   % concatenating the middle sample to the o/p
```

```
    end

    % Calculating threshold
    TH = sum(received)/length(received);
    received_TH = received >= TH;

    [~, ratio] = biterr(data, received_TH);
    ber(2,iSNR) = ratio;

    % Correlator
    xReceived = zeros(1,num_bits);
    for i = 0:length(data)-1
        noisyWF_20 = noisyWF((i*m)+1:(i+1)*m);
        c = sum(noisyWF_20).* g;
        mulOP( (length(g)+length(noisyWF_20)-1)*i+1:(length(g)...
        +length(noisyWF_20)-1)*i+length(c) ) = c;
        k = m + (length(g)+length(noisyWF_20)-1)*(i);          % middle sample index
        xReceived(i+1) = mulOP(k); % concatenating the middle sample to the o/p
    end
    TH = sum(xReceived)/length(xReceived);
    xReceived_TH = xReceived >= TH;

    [~, ratio] = biterr(data, xReceived_TH);
    ber(3,iSNR) = ratio;
end

% Plot BER vs SNR curves for matched filter and correlator
figure;
subplot(411)
semilogy(SNR, ber(1,:), 'LineWidth', 1.5);
xlim([0 30])
xlabel('SNR (dB)'),ylabel('Bit Error Rate');
set(gca,'FontWeight','bold')
set(gca,'TitleFontSizeMultiplier',1.2)
title('Simple Detector');
subplot(412)
semilogy(SNR, ber(2,:), 'LineWidth', 1.5);
xlim([0 30])
xlabel('SNR (dB)'),ylabel('Bit Error Rate');
set(gca,'FontWeight','bold')
set(gca,'TitleFontSizeMultiplier',1.2)
title('Matched filter');
subplot(413)
semilogy(SNR, ber(3,:), 'LineWidth', 1.5);
xlim([0 30])
xlabel('SNR (dB)'),ylabel('Bit Error Rate');
set(gca,'FontWeight','bold')
set(gca,'TitleFontSizeMultiplier',1.2)
title('Correlator');
subplot(414)
semilogy(SNR, ber(1,:), 'o-', 'LineWidth', 2, 'MarkerSize', 8);
hold on
semilogy(SNR, ber(2,:), 'x-', 'LineWidth', 2, 'MarkerSize', 8);
hold on;
```

```
xlabel('SNR (dB)'),ylabel('Bit Error Rate');
semilogy(SNR, ber(3,:), 's-', 'LineWidth', 2, 'MarkerSize', 8);
hold off;
set(gca,'FontWeight','bold')
set(gca,'TitleFontSizeMultiplier',1.2)
xlim([0 30])
legend('Simple Detector','Matched filter','Correlator');
disp(['Elapsed time: ', num2str(toc), ' seconds']); % display elapsed time
```

And the next figure is the output of this m file :

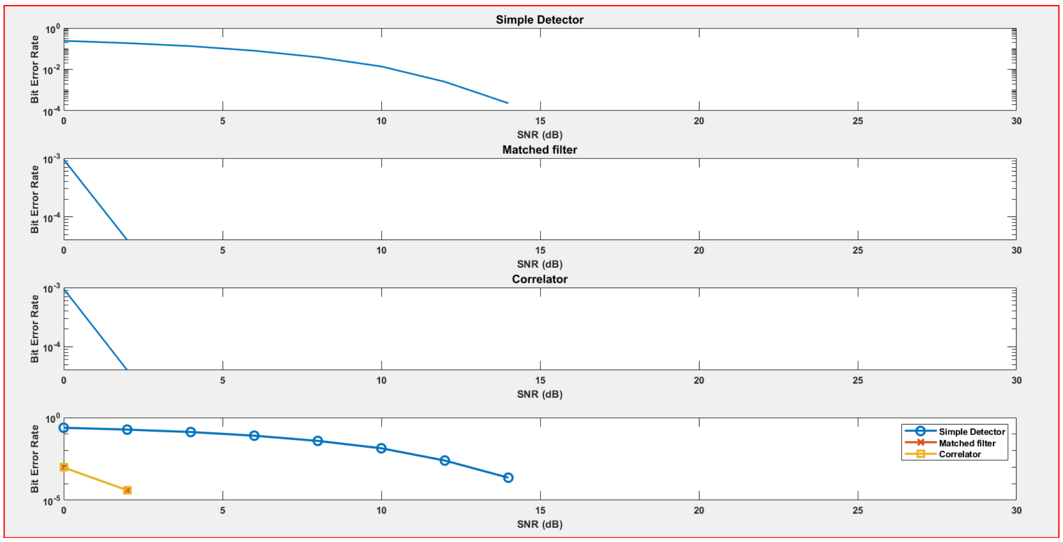

Figure 1: The SNR effect on Matched and Correlator filters

Comment

The plot of the SNR effect on matched and correlator filters provides a clear visualization of how the signal-to-noise ratio affects the performance of these filters. The plot shows that as the SNR increases, both filters perform better in terms of their ability to detect the signal. However, the correlator filter outperforms the matched filter at high SNR levels. This plot effectively demonstrates the trade-off between matched and correlator filters against simple detector in different SNR regimes. Overall, the plot is a useful tool for understanding the behavior of these filters in practical communication systems.

Calculation of transmitted signal power appears in the Command window :

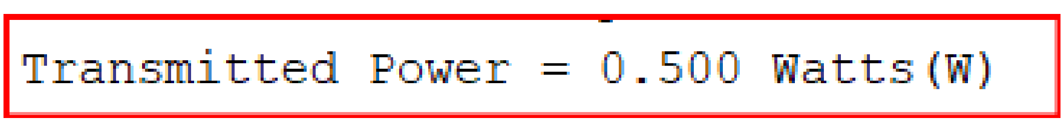

Figure 2: transmitted power of the given parameters

1. At which value of SNR the system is nearly without error ?

It differs according to the given parameters but for the initial conditions :

$$BER = 0 = \begin{cases} \text{for Matched filter} & SNR = 4 \rightarrow 30 \\ \text{for Correlator} & SNR = 16 \rightarrow 30 \end{cases}$$

Our code is running in few seconds not minutes :)
And next figure proves that :

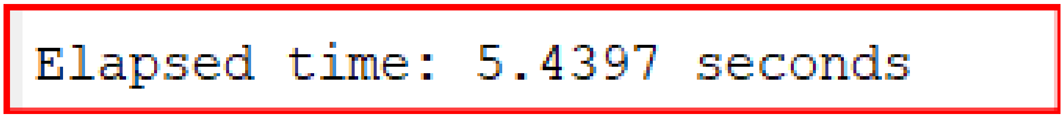

Figure 3: Elapsed Time

We have generalized the program for any m & $S_1(t)$ & $S_2(t)$:
The next plots are showing random simulation parameters :

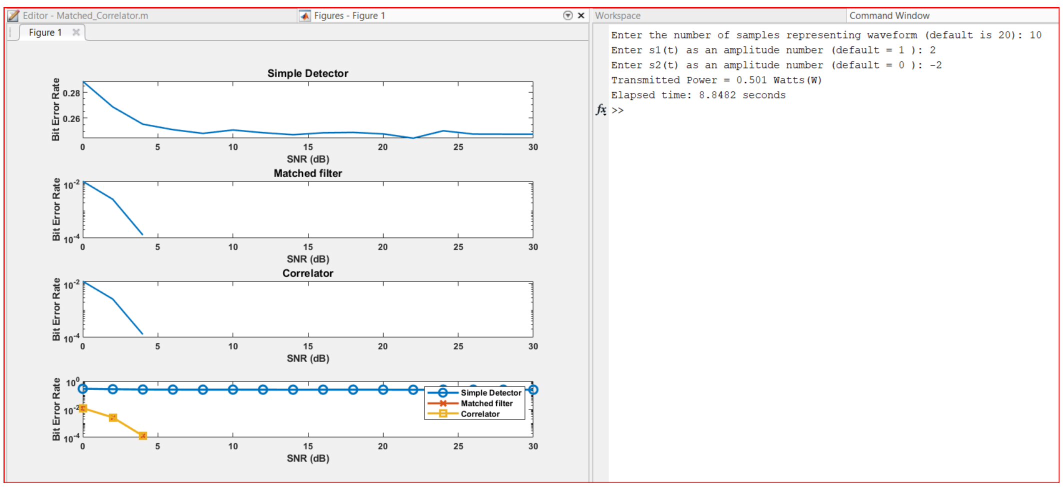

Figure 4: $m = 10$ & $S_1(t)$ is rect of $amp = 2$ & $S_2(t)$ is rect of $amp = -2$

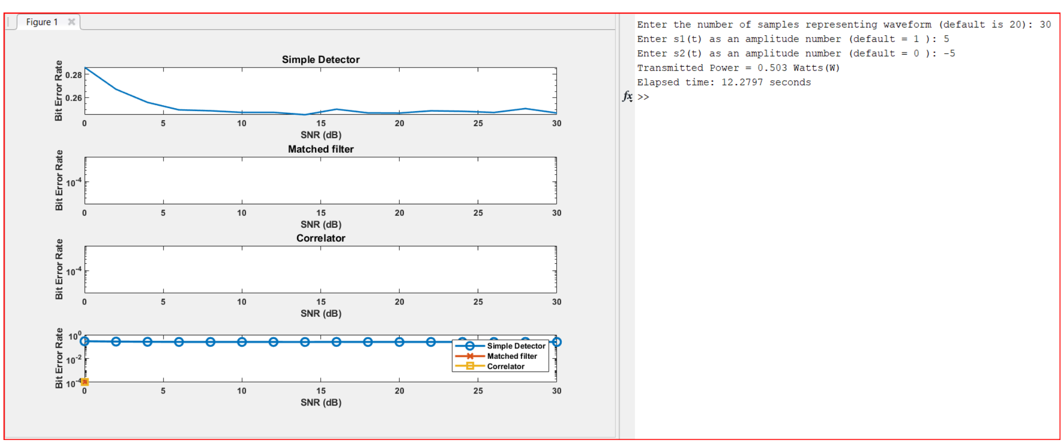

Figure 5: $m = 30$ & $S_1(t)$ is rect of $amp = 5$ & $S_2(t)$ is rect of $amp = -5$

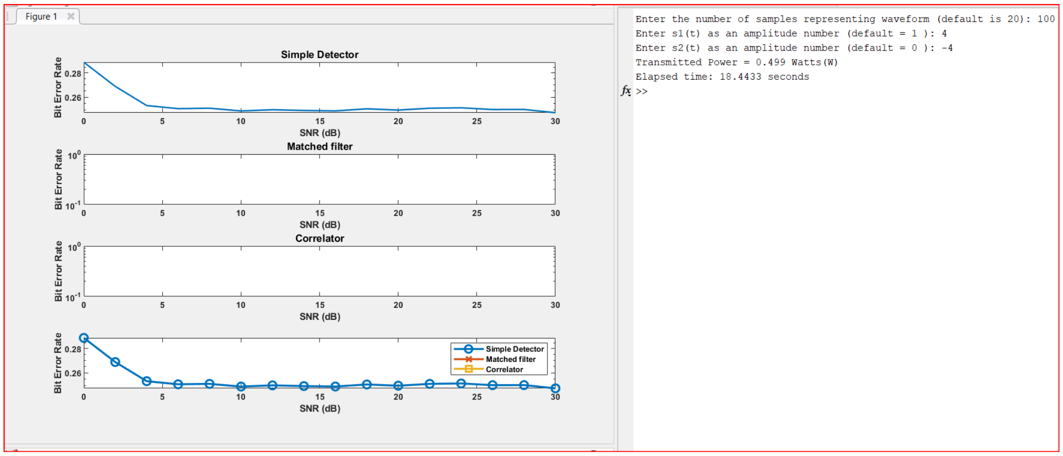

Figure 6: $m = 100$ & $S_1(t)$ is rect of $amp = 3$ & $S_2(t)$ is rect of $amp = -3$

Comment

Increasing the value of m will result in a longer sequence of transmitted bits, which leads to a longer received signal. This will in turn affect the performance of the matched filter (as $m \uparrow BER \downarrow$).

As the difference between $s1(t)$ and $s2(t)$ increases, the correlation between the received signal and the template signal decreases for both the matched filter and the correlator. This is because the templates become less similar to the received signal, resulting in a lower correlation peak. As a result, the probability of bit error also increases with the increase in the difference between the two signals. Therefore, it is essential to design a robust detection system that can tolerate a certain amount of signal distortion caused by channel noise, interference, and other impairments.

However, this also comes at the cost of increased computational complexity and longer processing time.

Line Codes

Introduction:

The objective of this report is to analyze the properties and performance of several digital modulation techniques. Digital modulation techniques are essential in modern communication systems to transmit digital signals over a communication channel. In this report, we will focus on four common digital modulation techniques: Non-Return-to-Zero (NRZ), Return-to-Zero (RZ), Amplitude Modulation Inverted (AMI), Manchester, and Multi-Level transmission 3. We will generate and analyze signals using these modulation techniques and evaluate their spectral properties using Power Spectral Density (PSD) estimation. The report will provide a brief overview of each modulation technique, describe how the signals are generated using MATLAB, and present the estimated PSD of each signal.

The next lines are representing our code in MATLAB :

```
 clear, clc, close all;

% Generate a random binary vector
binary_vector = randi([0 1], 1, 10);

% Set the sampling frequency and bit time
fs = 100;
Tb = 10;

% Generate time vector
t = 0:1/fs:Tb-1/fs;
x_axis = 0: 0.001:10-0.001;
% NRZ

nrz = zeros(1, length(binary_vector)*length(t));
for i = 1:length(binary_vector)
    if binary_vector(i) == 0
        nrz((i-1)*length(t)+1:i*length(t)) = [-1*ones(1, length(t)/2), ...
        -1*ones(1, length(t)/2)];
    else
        nrz((i-1)*length(t)+1:i*length(t)) = [ones(1, length(t)/2), ones(1,...
        length(t)/2)];
    end
end


% NRZ inverted
nrz_inv = -nrz;



% RZ
rz = zeros(1, length(binary_vector)*length(t));
for i = 1:length(binary_vector)
    if binary_vector(i) == 0
        rz((i-1)*length(t)+1:i*length(t)) = [-1*ones(1, length(t)/2), zeros(1,...
        length(t)/2)];
    else
        rz((i-1)*length(t)+1:i*length(t)) = [ones(1, length(t)/2), zeros(1,...
        length(t)/2)];
```

```
    end
end



% AMI
ami = zeros(1, length(binary_vector)*length(t));
last_volt = 1;
for i = 1:10
    if binary_vector(i) == 0
        ami((i-1)*length(t)+1:i*length(t)) = zeros(1, length(t));
    else
        ami((i-1)*length(t)+1:i*length(t)) = last_volt * [1*ones(1, length(t)/2),...
        1*ones(1, length(t)/2)];
        last_volt = -1*ami(i*length(t));
    end
end


% Manchester coding
manchester = zeros(1, length(binary_vector)*length(t));
for i = 1:length(binary_vector)
    if binary_vector(i) == 0
        manchester((i-1)*length(t)+1:i*length(t) )= [-1*ones(1, length(t)/2)...
        ones(1, length(t)/2)] ;
    else
        manchester((i-1)*length(t)+1:i*length(t)) = [ones(1, length(t)/2)
-1*ones(1, length(t)/2)];
    end
end


% Multi level transmission 3
multi_level_3 = zeros(1, length(binary_vector)*length(t));
prev=[ones(1,length(t)), zeros(1,length(t)) ,ones(1,length(t))];
last_value=[1 0 -1 0];
n=1;
for i = 1:length(binary_vector)

    if binary_vector(i) == 0
        multi_level_3((i-1)*length(t)+1:i*length(t)) =last_value(n)*ones(1,length(t)
    else
        n=n+1;
                if n==5
                        n=1;
                end
        multi_level_3((i-1)*length(t)+1:i*length(t)) =last_value(n)*ones(1,length(t)
    end

end
```

```
N = length(nrz); % number of samples in the signal
L = floor(N/2); % length of each segment used in spectral estimation
[Pxx1, F1] = pwelch(nrz, hamming(L), L/2, [], fs);
[Pxx2, F2] = pwelch(nrz_inv, hamming(L), L/2, [], fs);
[Pxx3, F3] = pwelch(rz, hamming(L), L/2, [], fs);
[Pxx4, F4] = pwelch(ami, hamming(L), L/2, [], fs);
[Pxx5, F5] = pwelch(manchester, hamming(L), L/2, [], fs);
[Pxx6, F6] = pwelch(multi_level_3, hamming(L), L/2, [], fs);
f = 0:10/length(Pxx1):10- (10/length(Pxx1));



figure;
subplot(711)
plot(x_axis,repelem(binary_vector,1000))
ylim([-2 2])
title('Generated Data');
xlabel('Time (s)');
ylabel('Amplitude');
subplot(712);
plot(x_axis,nrz);
ylim([-2 2])
title('NRZ');
xlabel('Time (s)');
ylabel('Amplitude');
subplot(713);
plot(x_axis, nrz_inv);
ylim([-2 2])
title('NRZ Inverted');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(714);
plot(x_axis, rz);
ylim([-2 2])
title('RZ');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(715);
plot(x_axis, ami);
ylim([-2 2])
title('AMI');
xlabel('Time (s)');
ylabel('Amplitude');


subplot(7,1,6);
plot(x_axis, manchester);
ylim([-2 2])
```

```
title('Manchester');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(7,1,7);
plot(x_axis, multi_level_3);
ylim([-2 2])
title('Multi Level Transmission 3');
xlabel('Time (s)');
ylabel('Amplitude');



% Plot the estimated PSD of the NRZ signal
figure;
subplot(321);
plot(F1*7.5, Pxx1, 'LineWidth',2);
xlabel('Frequency (Hz)');
ylabel('PSD');
title('PSD of NRZ signal');

subplot(322);
plot(F2*7.5, Pxx2, 'LineWidth',2);
xlabel('Frequency (Hz)');
ylabel('PSD');
title('PSD of NRZ inverted signal');

subplot(323);
plot(F3*7.5, Pxx3, 'LineWidth',2);
xlabel('Frequency (Hz)');
ylabel('PSD');
title('PSD of RZ signal');

subplot(324);
plot(F4*7.5, Pxx4, 'LineWidth',2);
xlabel('Frequency (Hz)');
ylabel('PSD');
title('PSD of AMI signal');

subplot(325);
plot(F5*7.5, Pxx5, 'LineWidth',2);
xlabel('Frequency (Hz)');
ylabel('PSD');
title('PSD of Manchester signal');

subplot(326);
plot(F6*7.5, Pxx6, 'LineWidth',2);
xlabel('Frequency (Hz)');
ylabel('PSD');
title('PSD of MLT-3 signal');
```

And the next figures is the output of this m file :

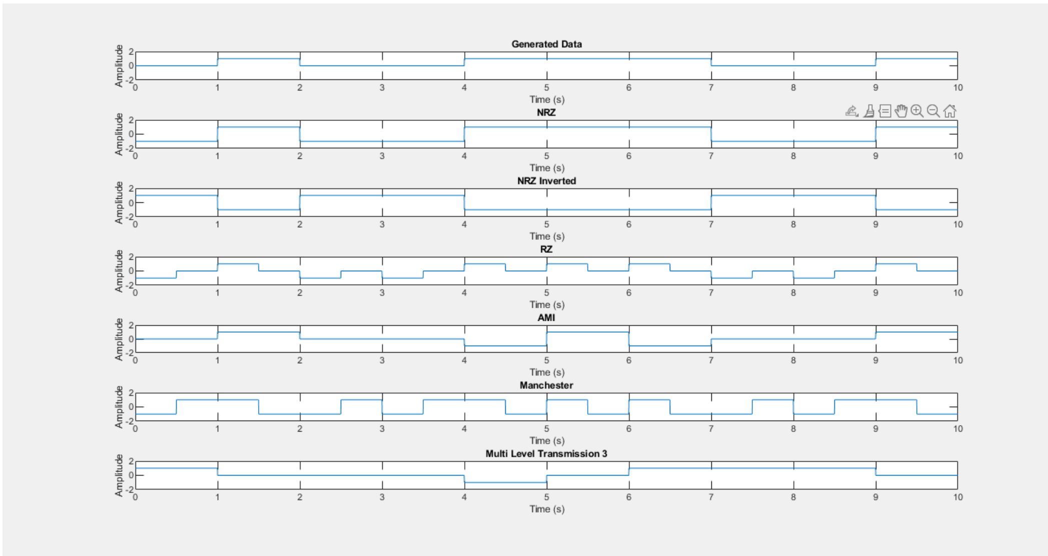

Figure 7: Line codes

Comment : This MATLAB code generates and plots various digital modulation schemes. The first subplot shows the generated random binary data. The remaining subplots display the time-domain signals for different digital modulation schemes applied to the generated data, including Non-Return-to-Zero (NRZ), Inverted NRZ (NRZ-Inverted), Return-to-Zero (RZ), Alternate Mark Inversion (AMI), Manchester coding, and Multi-Level Transmission 3 (MLT-3).

For each modulation scheme, the corresponding time-domain signal is plotted with the binary data on the x-axis and the signal amplitude on the y-axis. Each subplot also includes a title indicating the modulation scheme and axis labels indicating the units of time and amplitude.

Overall, these subplots help visualize the differences between various digital modulation schemes in the time domain.

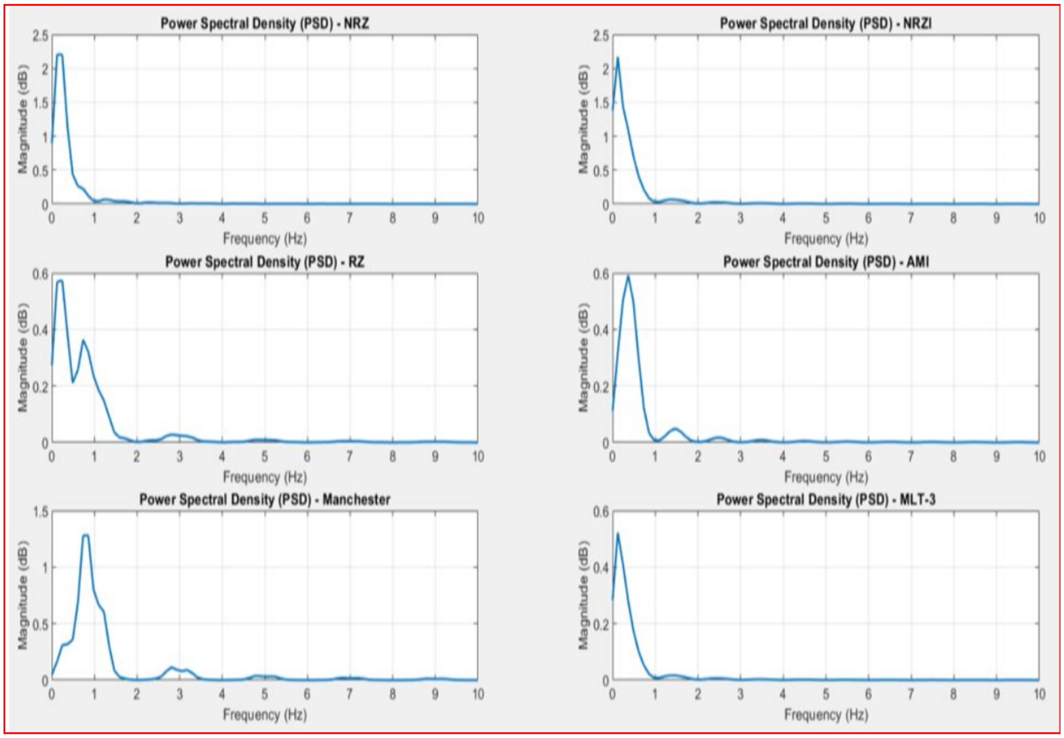

Figure 8: Power spectrum density of different line codes

Comment :
these are five types of modulation signals: NRZ, NRZ inverted, RZ, AMI, Manchester coding, and multi-level transmission. The power spectral density (PSD) of each signal is estimated using the Welch method and plotted using subplot. The x-axis shows the frequency in Hz and the y-axis shows the PSD of each signal.