Communication Project
Project 1
3rd Year Comm. | Spring 2025

NAME	SECTION	ID
Youssef Khaled Omar Mahmoud	4	9220984
Omar Ayman Amin Mohamed	3	9220528
Ahmed Mohamed Ahmed Sultan	1	9220080
Shahd Hamad Shaban Saleh	2	9220401
Mohamed Ahmed Abd El Hakam	3	9220647

Instructor: Eng. Mohamed Khaled
Dr. Mohammed Nafie & Dr. Mohamed Khairy

Contents

Table of Content

A. Project Description

Using software radio technique (SDR) to transmit stream of randomness bits through an ideal channel (which performing a small delay) using Matlab. Performing measures and analysis to see the performance of the system through three main line codes (unipolar, polar nrz and polar rz).

B. Introduction

Software radio is a revolutionary approach that brings the programming code directly to the antenna, minimizing reliance on traditional radio hardware.
By doing so, it transforms challenges associated with radio hardware into software- related issues.
Unlike conventional radios, where signal processing primarily relies on analog circuitry or a combination of analog and digital chips, software radio operates by having software dictate both the transmitted and received waveforms.
This paradigm shift allows for greater flexibility and adaptability in radio systems, as they can be easily reconfigured and optimized through software updates, rather than hardware modifications.

C. Control Variables

A = 4 → amplitude of line code
rows_num = 500 → number of realizations
real_length = 100 → length of realization (in bits)
samples_num = 7→ number of samples per bit
samples_per_real → total samples per realization (700 samples)
fs = 100 → sampling frequency

D. Generation of Data

Using the function: "**Randi**" to generate random binary data of size 500x101
(500 waveforms each with 101 bits). This data represents the binary bits that need to be encoded.

```
Data = randi([0,1], rows_num, real_length+1);
```

E. polar NRZ ensemble creation

- The data consists of 0s and 1s. We converted these values to A and -A respectively.
- Then, we utilized the "**repelem**" function to repeat each element seven times.

```
polar_NRZ_amplitude = (2 * Data - 1) * A;
polar_NRZ_reals = repelem (polar_NRZ_amplitude, 1, samples_num);
```

F. Uni polar NRZ ensemble creation

- We then generate unipolar NRZ amplitudes along with its realization.
- We convert data (1,0) to 1 →A ,0→ 0 to have uni_polar_NRZ.

```
uni_polar_NRZ_amplitude = Data * A;
uni_polar_NRZ_reals = repelem (uni_polar_NRZ_amplitude, 1, samples_num);
```

G. Generation of Polar realization

```
polar_RZ_reals = polar_NRZ_reals;
       for i = 1:rows_num
           for j = 5:7:705
      polar_RZ_reals(i, j:j+2) = 0;
        end
     end
```

- we then generate polar RZ amplitudes along with its realization.
- We generate polar_NRZ By setting samples to zero during the middle of each bit period, the code effectively generates the Polar RZ signal from the Polar NRZ signal.
- The interval [j:j+2] represents the middle part of the bit period, and setting those samples to zero creates the RZ waveform.

H. Random initial time shift

- Generating a single random initial time delay that can range from ‘0’ to ‘6’ samples for each waveform using the function “randi”.

```
start_indices = randi([0, 6], rows_num, 1);
```

- Then, we utilized the randi function to generate a random number ranging from 0 to 6, which represents the delay or start time, then we take the elements from this random index (start_indices) to 700+(start_indices)

```
for i = 1:rows_num
    start_index = start_indices(i);
    end_index = samples_per_real + start_index;
    Rndm_shift_polar_NRZ(i, :) = polar_NRZ_reals(i,
start_index+1:end_index);
    Rndm_shift_uni_polar_NRZ(i, :) =
uni_polar_NRZ_reals(i, start_index+1:end_index);
    Rndm_shift_polar_RZ(i, :) = polar_RZ_reals(i,
start_index+1:end_index);
end
```

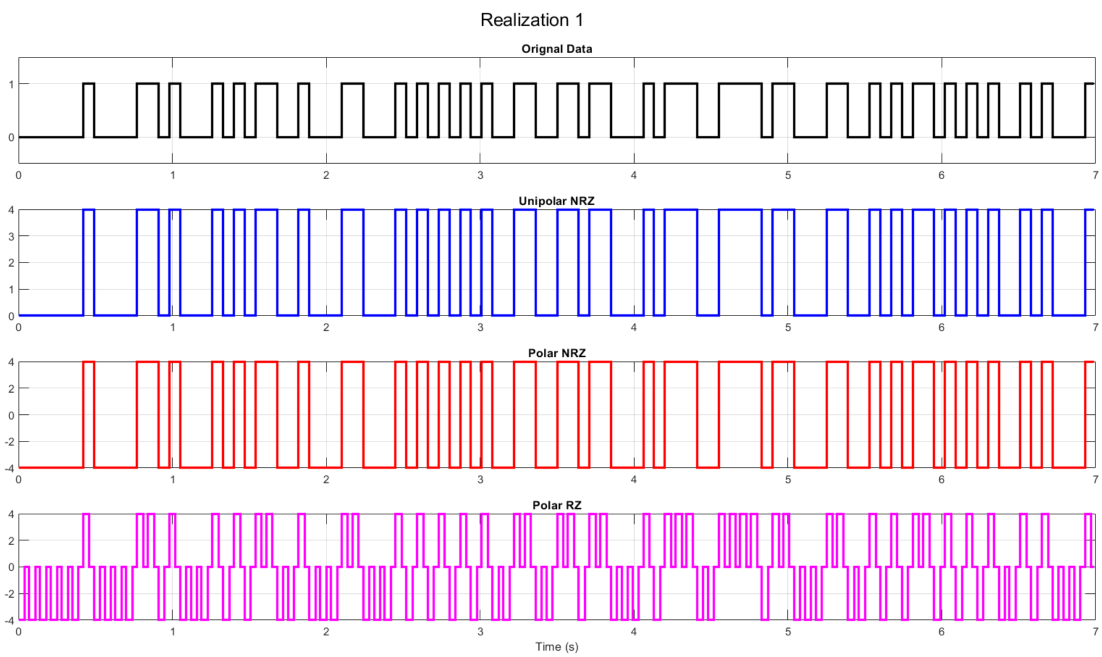


Figure 1 Realization

I. Questions

1. Statistical Mean

1.1. Hand Analysis

For the "Statistical Mean" which represents the average of all the realizations at the same time instant, let us consider the first line code method "Unipolar NRZ"
$\mu X(t) = 0 * 0.5 + 4 * 0.5 = 2\ (Constant\ across\ time)$
And in the same matter, we can calculate the "Statistical Mean" for both "Polar NRZ" and "Polar RZ" as following:
$\mu X\_PNRZ(t) = 4 * 0.5 + (-4) * 0.5 = 0\ (Constant\ across\ time)$

$$\mu X\_PRZ(t) = 4 * 0.5 + (-4) * 0.5 = 0\ (Constant\ across\ time)$$

1.2. Code Snippet

```
Col_Sum_pol_NRZ = zeros(1, samples_per_real); % initialization of sum matrix
Col_Sum_uni_pol_NRZ = zeros(1, samples_per_real); % initialization of sum
matrix
Col_Sum_pol_RZ = zeros(1, samples_per_real); % initialization of sum matrix

for k = 1 : samples_per_real
    for j = 1 : rows_num
        Col_Sum_pol_NRZ(k) = Col_Sum_pol_NRZ(k) + Rndm_shift_polar_NRZ(j, k);
        Col_Sum_uni_pol_NRZ(k) = Col_Sum_uni_pol_NRZ(k) +
Rndm_shift_uni_polar_NRZ(j, k);
        Col_Sum_pol_RZ(k) = Col_Sum_pol_RZ(k) + Rndm_shift_polar_RZ(j, k);
    end
end
```

- we have two loops, the outer one for the time instants (K) and the inner for the number of realizations (j).

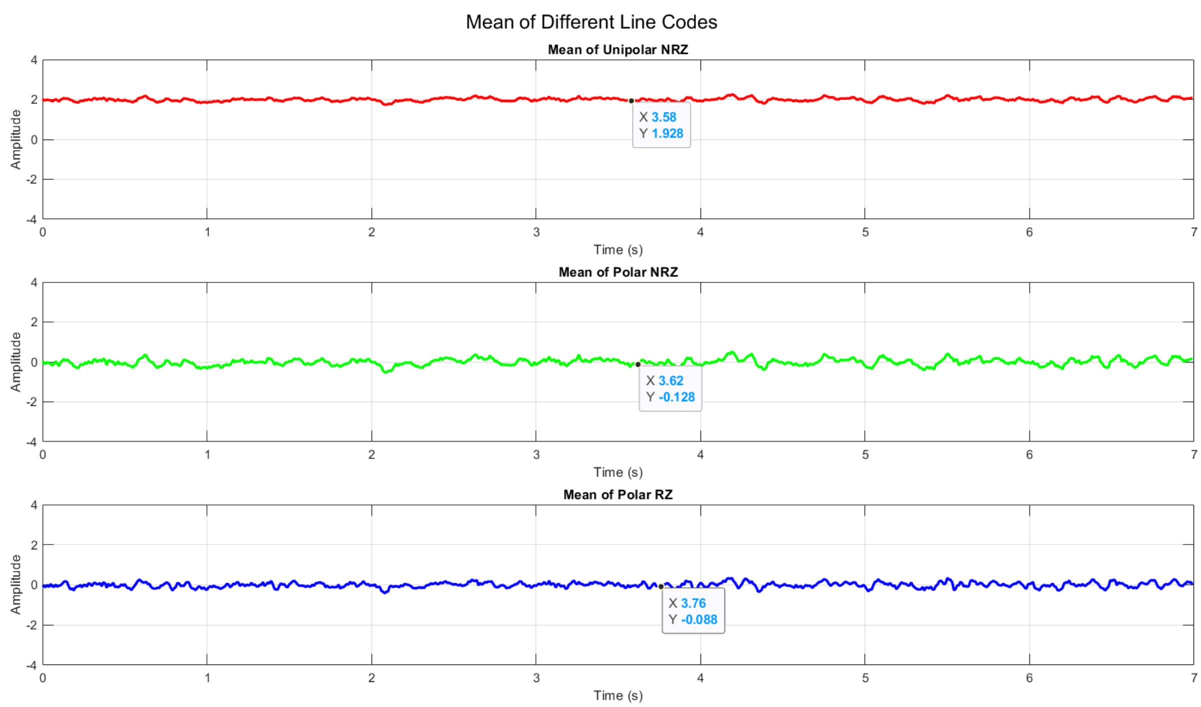


Figure 2 Plot of Statistical Mean

- As expected, polar RZ & NRZ have almost zero mean and the uni polar has mean around 2 Bec its amplitude ranges from 0:4

2. Statistical Autocorrelation

2.1. Hand Analysis

$$R_X(\tau) = E[X(\tau)\,X(t+\tau)] = \sum X(\tau)\,X(t+\tau)\,P(X(\tau)\,X(t+\tau))$$

- **For Unipolar NRZ:**
  We have 2 cases **(Considering T to be 70ms or 7 samples),**
  1. $|\tau| < T$

$$\begin{aligned} R_X(\tau) &= E[X(\tau)\,X(t+\tau)] \\ &= 4^2 * P(4{,}4) + 0^2 * P(0{,}0) + 4 * 0 * P(0{,}4) + 0 * 4 * P(4{,}0) \\ &= 4^2 * P(4{,}4) \end{aligned}$$

$$P(4{,}4) = P(X(t+\tau) = 4 \mid X(t) = 4) * P(X(t) = 4)$$

$$P(X(t+\tau) = 4 \mid X(t) = 4) = P(T) + P(T) * P(X(t+\tau) = 4)$$

$$P(T) = \int_{t_1}^{t_2} \frac{1}{T}\,dt = \frac{\tau}{T} \Rightarrow P(T') = 1 - P(T') = 1 - \frac{\tau}{T}$$

$$R_X(\tau) = \frac{4^2}{2} \cdot \left(1 - \frac{|\tau|}{2T}\right) = 8\left(1 - \frac{|\tau|}{2T}\right)$$

  2. $|\tau| > T$

$$\begin{aligned} R_X(\tau) &= E[X(\tau)\,X(t+\tau)] \\ &= 4^2 * 0.5 * 0.5 + 0^2 * 0.5 * 0.5 + 4 * 0 * 0.5 * 0.5 + 0 * 4 * 0.5 * 0.5 \\ &= 4^2 * 0.5 * 0.5 \\ &= 4 \end{aligned}$$

  - And using the same flow, we can find that the ACF for "**Polar NRZ**" is

$$\text{if } |\tau| < T \rightarrow R_X(\tau) = 4^2 \cdot \left(1 - \frac{\tau}{T}\right) = 16\left(1 - \frac{|\tau|}{T}\right)$$

$$\text{if } |\tau| > T \rightarrow R_X(\tau) = \text{Zero}$$

- And similarly, the ACF for "**Polar RZ**" is

$$\text{if } |\tau| < \frac{T}{2} \rightarrow R_X(\tau) = \frac{4^2}{2} \cdot \left(1 - \frac{2|\tau|}{T}\right) = 8\left(1 - \frac{2|\tau|}{T}\right)$$

$$\text{if } |\tau| > \frac{T}{2} \rightarrow R_X(\tau) = \text{Zero}$$

And as we know:

$$Total\ Power = RX(0)\ \&\ DC\ Power = RX(\infty)$$

$$AC\ Power = Total\ Power - DC\ Power$$

	Unipolar NRZ	Polar NRZ	Polar RZ
Total Power	**8**	**16**	**8**
DC Power	**4**	**0**	**0**
AC Power	**4**	**16**	**8**

2.2. Code Snippet

```
averages_pol_NRZ = zeros(1, 700);
averages_uni_pol_NRZ = zeros(1, 700);
averages_pol_RZ = zeros(1, 700);

for i = 0:699
    result_column_pol_NRZ = Rndm_shift_polar_NRZ(:, 1) .*
Rndm_shift_polar_NRZ(:, i+1);
    averages_pol_NRZ(i+1) = mean(result_column_pol_NRZ);
end

for i = 0:699
    result_column_uni_pol_NRZ = Rndm_shift_uni_polar_NRZ(:, 1) .*
Rndm_shift_uni_polar_NRZ(:, i+1);
    averages_uni_pol_NRZ(i+1) = mean(result_column_uni_pol_NRZ);
end

for i = 0:699
    result_column_pol_RZ = Rndm_shift_polar_RZ(:, 1) .*
Rndm_shift_polar_RZ(:, i+1);
    averages_pol_RZ(i+1) = mean(result_column_pol_RZ);
end

t = -699 : 699;
figure("name", "Statistical Autocorrelation");
```

Annotations

- The Statistical Autocorrelation is created by taking the element-wise product of each column with the first column of a selected matrix of data points, then averaging the resulting column-wise products.

- To guarantee that Autocorr is an even fun we concatenate between the result of fliplr fun & the averages vector before flipping (2:700 to ensure no repeated value at zero)

- The resulting autocorrelation values are plotted against the corresponding time delays (τ). We observe that at $\tau = 0$ the autocorrelation with the point itself is maximum, indicating perfect correlation. However, the other points are around zero to be exactly zero it requires an infinite number of samples.

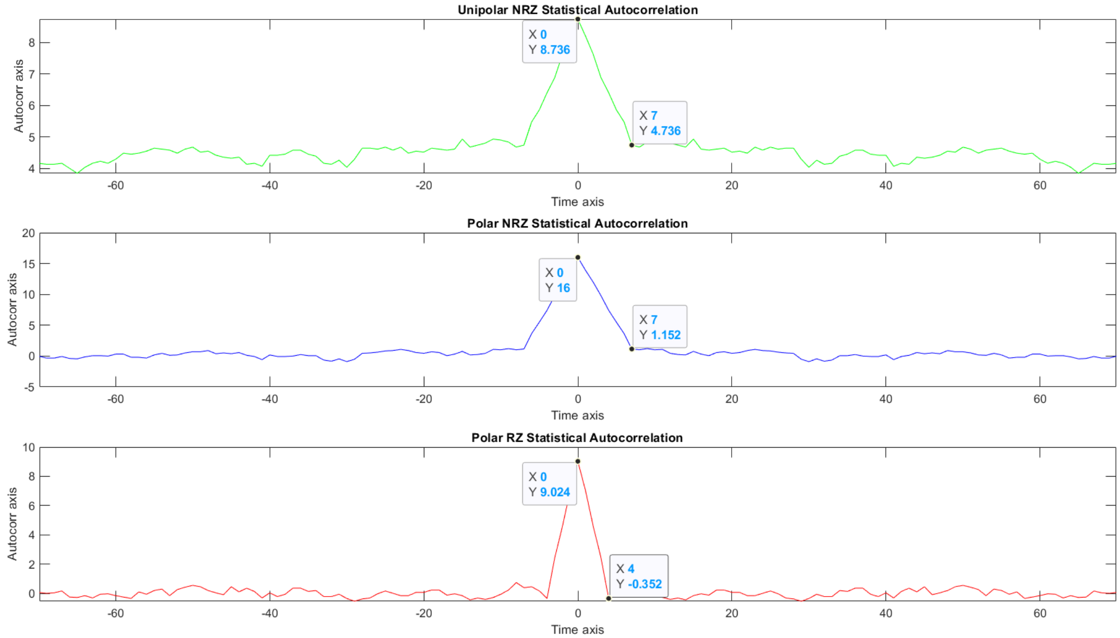


Figure 3 Auto Correction plot

3. Is the Process Stationary

- Yes, the process is stationary (WSSP) because the mean is constant function in time as shown in Figure 2 Plot of Statistical Mean and theoretically the autocorrelation depends only on the time difference not the absolute time.

4. The time mean and autocorrelation function for one waveform

4.1. Average Time Mean

```
polar_NRZ_time_mean = zeros(500, 1);
uni_polar_NRZ_time_mean = zeros(500, 1);
polar_RZ_time_mean = zeros(500, 1);

for i = 1: 700
    polar_NRZ_time_mean = polar_NRZ_time_mean + 
Rndm_shift_polar_NRZ (: , i);
    uni_polar_NRZ_time_mean = uni_polar_NRZ_time_mean + 
Rndm_shift_uni_polar_NRZ (: , i);
    polar_RZ_time_mean = polar_RZ_time_mean + Rndm_shift_polar_RZ 
(: , i);
end
```

- We add the value of all realizations at each time instant then divide by the number of samples (700 sample per realization).

- Then the average value of each realization is added to the corresponding row in the result vector (line_code_time_mean).

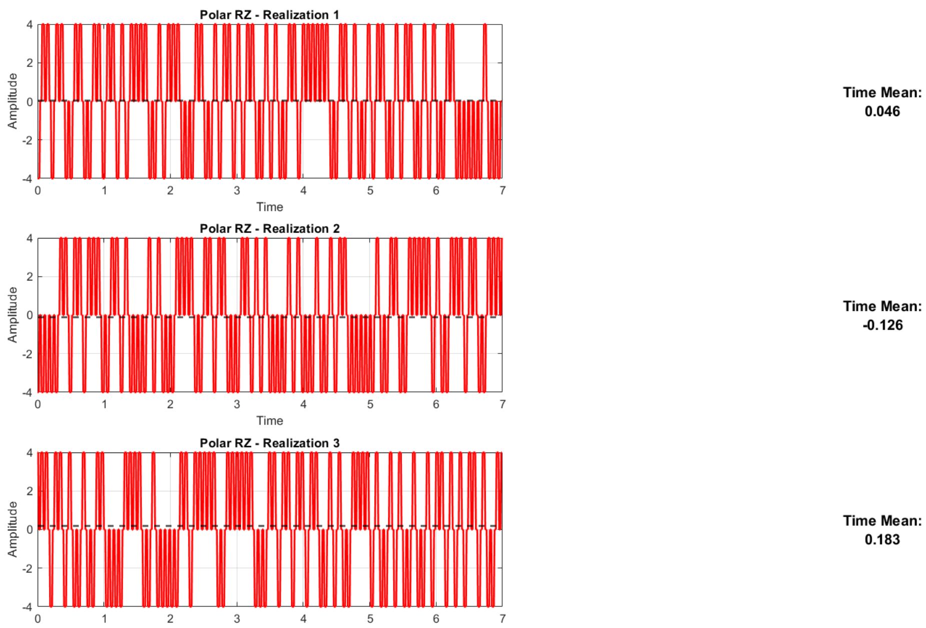


Figure 5 Time Mean Ploar RZ

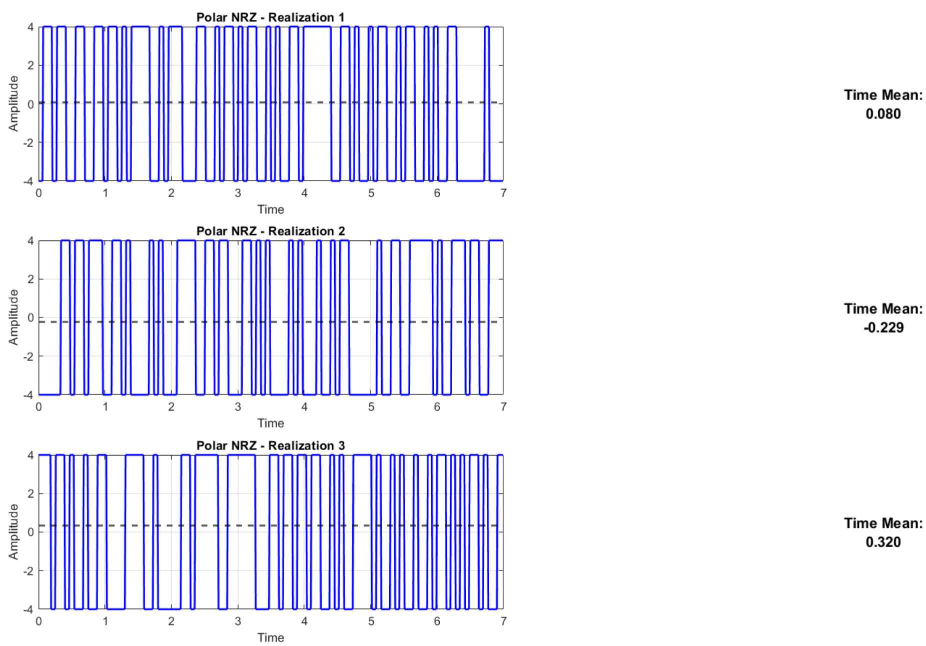


Figure 4Time Mean Ploar NRZ

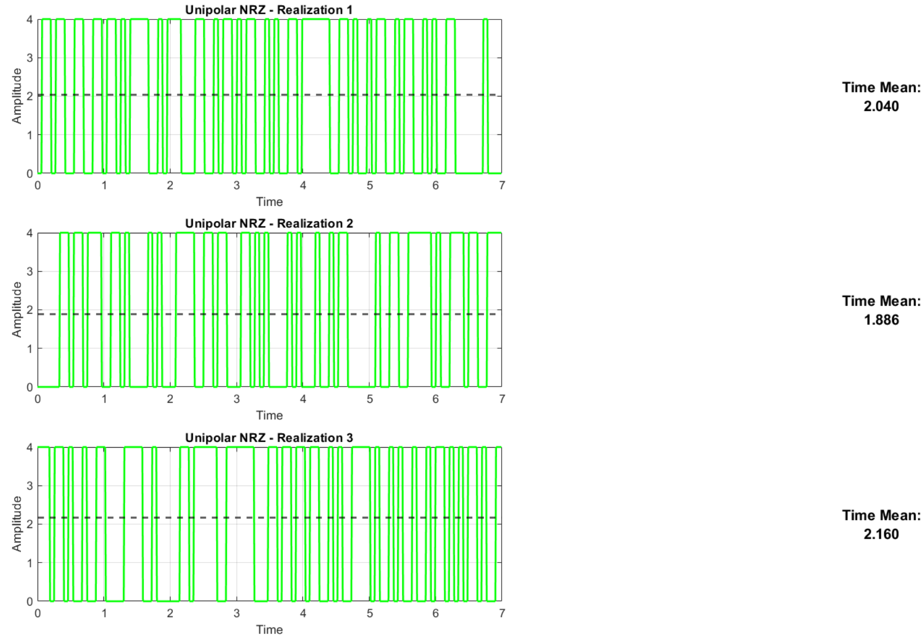


Figure 7 Time Mean Uni polar NRZ

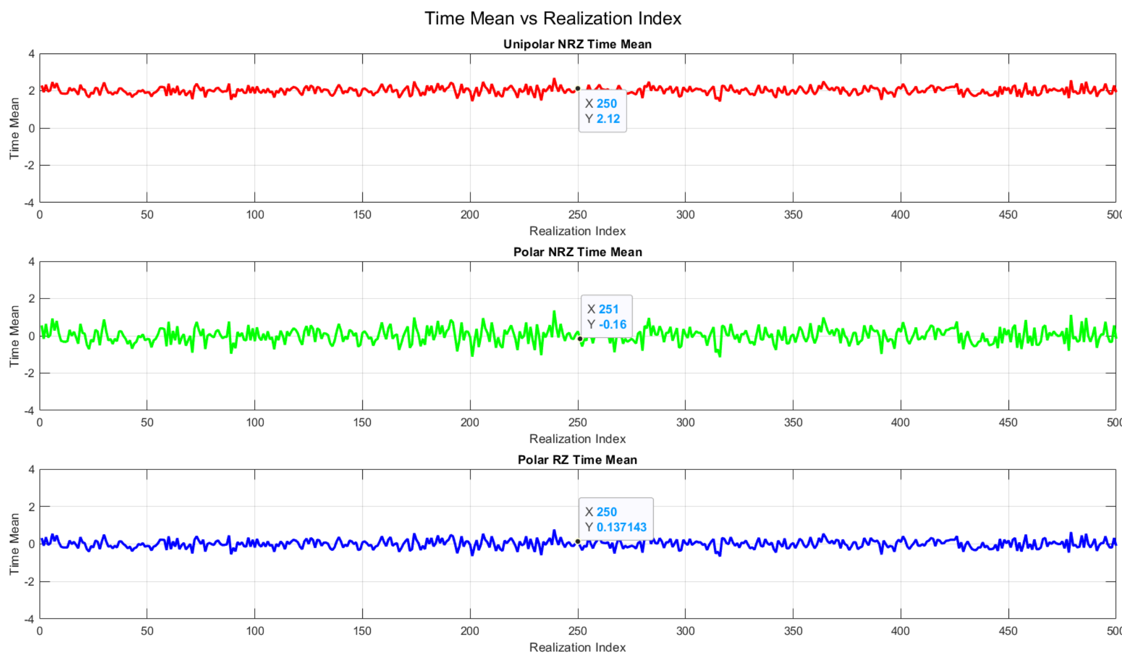


Figure 6 Average time mean plot for all line codes

- As expected, polar RZ & NRZ have almost zero mean and the uni polar has mean around 2 Bec its amplitude ranges from 0 : 4

4.2. Average Time Auto Correction

```
% Initialize autocorrelation matrices (each row for a realization)
    R_unipolar_nrz_t = zeros(num_realizations, length(taw2));
    R_polar_nrz_t = zeros(num_realizations, length(taw2));
    R_polar_rz_t = zeros(num_realizations, length(taw2));

    % Compute autocorrelation for each realization separately
    for r = 1:num_realizations
        for i = taw2
            M = i + max_lag + 1; % Shift index to fit within array bounds

            % Compute sum for each lag (dot product of signal with shifted
version)
            if i >= 0
                valid_samples = num_samples - i;
                R_unipolar_nrz_t(r, M) = sum(UnipolarNRZ(r, 1:valid_samples)
.* UnipolarNRZ(r, i+1:num_samples)) / valid_samples;
                R_polar_nrz_t(r, M) = sum(PolarNRZ(r, 1:valid_samples) .*
PolarNRZ(r, i+1:num_samples)) / valid_samples;
                R_polar_rz_t(r, M) = sum(PolarRZ(r, 1:valid_samples) .*
PolarRZ(r, i+1:num_samples)) / valid_samples;
            else
                valid_samples = num_samples + i;
                R_unipolar_nrz_t(r, M) = sum(UnipolarNRZ(r, -i+1:num_samples)
.* UnipolarNRZ(r, 1:valid_samples)) / valid_samples;
                R_polar_nrz_t(r, M) = sum(PolarNRZ(r, -i+1:num_samples) .*
PolarNRZ(r, 1:valid_samples)) / valid_samples;
                R_polar_rz_t(r, M) = sum(PolarRZ(r, -i+1:num_samples) .*
PolarRZ(r, 1:valid_samples)) / valid_samples;
            End
```

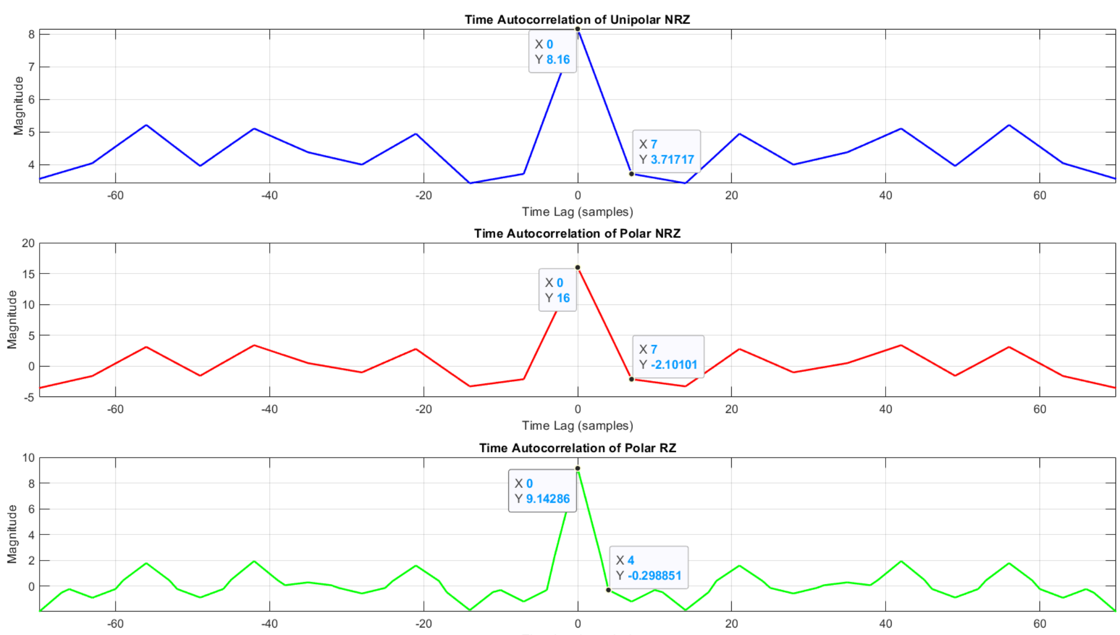
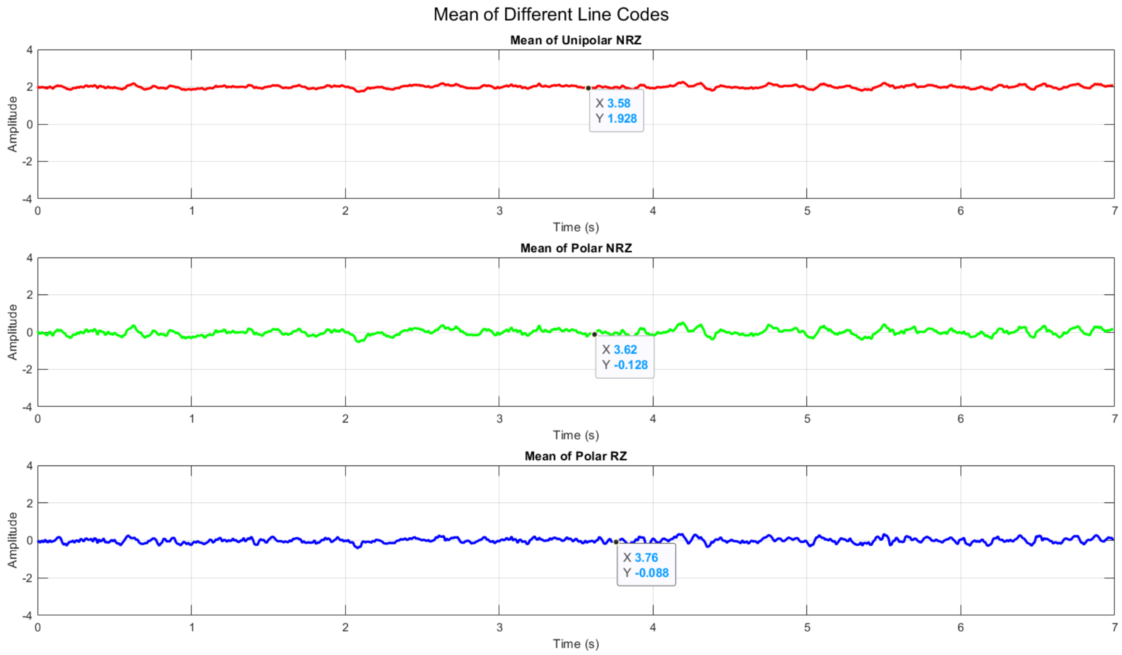


Figure 8 Time Auto Correlation

- The time autocorrelation is calculated by $r_{xx} = \frac{1}{N}\sum_{n=0}^{N-1} x(n)x(n-k)$ of the first waveform.
    - The autocorrelation function has maximum at $\boldsymbol{\tau} = \mathbf{0}$ and it is an even function.

5. Is The Random Process Ergodic



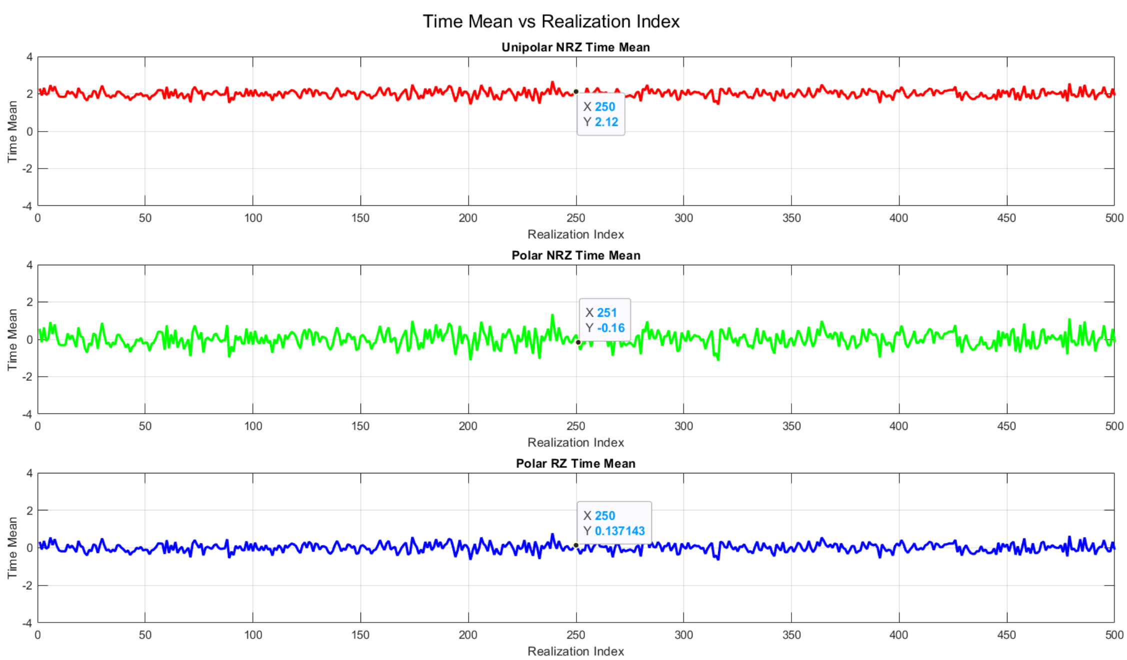


Figure 9 Comparison Between Time and Statistical Mean

- Yes, because the time mean equal to the ensemble mean and the time autocorrelation is equal to the ensemble autocorrelation.
  Then this process is ergodic

6. the PSD & Bandwidth of the Ensemble

```
function BW = find_bandwidth(PSD, freq_axis)
    % Function to find the -3dB bandwidth from PSD
    PSD_max = max(PSD);         % Find max power
    threshold = PSD_max / 2;    % -3dB threshold (half power)

    % Find indices where PSD is above threshold
    valid_indices = find(PSD >= threshold);

    % Check if we found any valid points
    if isempty(valid_indices)
        BW = 0; % If no valid range is found, return 0
        return;
    end
```

- We take the Fourier transform of the statistical autocorrelation then centralize the graph around zero.

- since $T_s = \frac{\text{Bit time}}{\text{no of samples per bit}} = \frac{70\ ms}{7} = 10\ ms \rightarrow F_s = 100$

- Then we normalize the PSD Amplitude through dividing by Fs

For the BW

- the BW is the frequency of the first zero of sinc^2 function (intersection with frequency-axis)

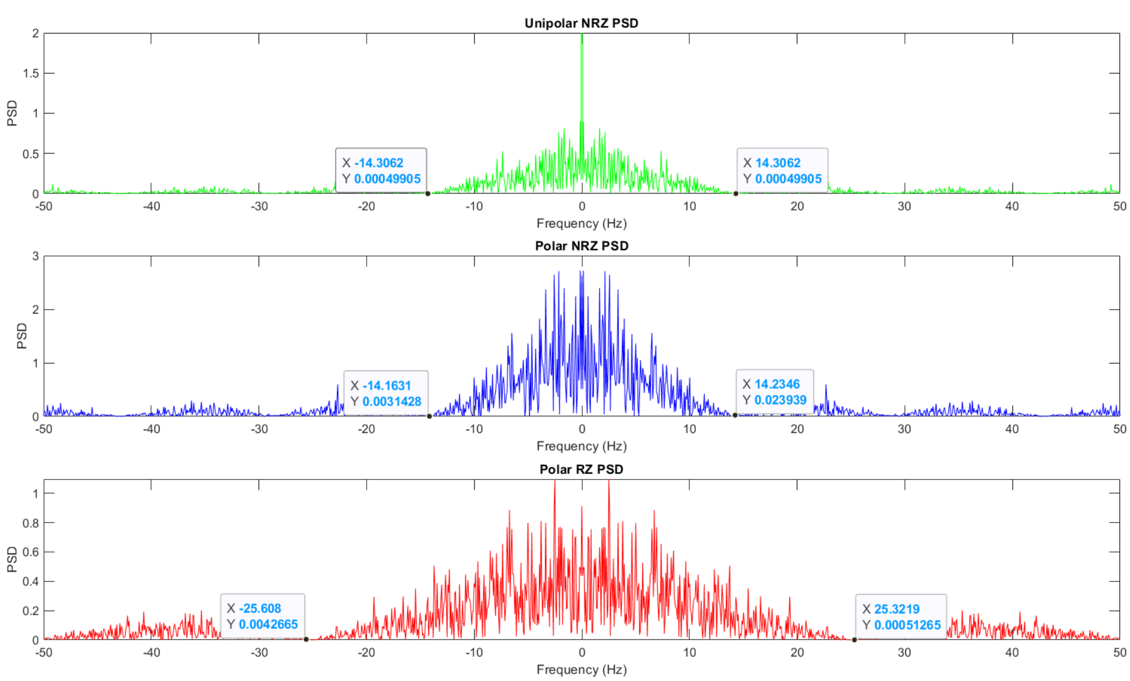


Figure 10 PSD plot of the Ensemble

Annotations

- in polar RZ & NRZ : we have sinc^2 function without delta at zero frequency (NO DC)
- in uni polar NRZ : we have sinc^2 function with delta at zero frequency (there is DC)
- BW of the unipolar NRZ & polar NRZ is the bitrate which approximately equal 14.23 hz
- BW of the polar RZ is the double of bitrate which approximately equal 25.32 hz

J. Appendix

```
clc;
clear;
close all;

% Parameters
A = 4;                      % Amplitude
N_realizations = 500;         % Number of waveforms (ensemble size)
num_bits = 100+1;             % Bits per waveform and one extra bit for shifting
bit_duration = 70e-3;     % 70 ms per bit
dac_interval = 10e-3;     % DAC updates every 10 ms
samples_per_bit = round(bit_duration / dac_interval);  % 7 samples per bit
total_time = num_bits * bit_duration;                  % Total waveform
duration
t = 0:dac_interval:(total_time - dac_interval);        % Time vector

% Preallocate matrices for efficiency
Unipolar_All = zeros(N_realizations, length(t), 'int8');
PolarNRZ_All = zeros(N_realizations, length(t), 'int8');
PolarRZ_All = zeros(N_realizations, length(t), 'int8');

% Generate and store 500 realizations
for i = 1:N_realizations
    Data = randi([0, 1], 1, num_bits, 'int8');  % Random bit sequence

    %encode the data
    [Unipolar, PolarNRZ, PolarRZ] = generate_linecodes(Data, A,
samples_per_bit);

    % Store in matrices
    Unipolar_All(i,:) = Unipolar;
    PolarNRZ_All(i,:) = PolarNRZ;
    PolarRZ_All(i,:) = PolarRZ;

    % Plot the first realization
    if i==1
        plot_linecodes(Data, Unipolar, PolarNRZ, PolarRZ, t, num_bits-
1,'Realization 1');
```

```
    end
end



% Plot the first 5 realizations as a sample
figure;
for i = 1:5
    subplot(5,1,i);
    stairs(t, Unipolar_All(i,:), 'b', 'LineWidth', 1.5);
    title(['Unipolar NRZ - Realization ' num2str(i)]);
    grid on;
end
xlabel('Time (s)');

% Apply random shift to all realizations
[Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted] =...
   apply_random_shift_fixed_size(Unipolar_All, PolarNRZ_All, PolarRZ_All,
samples_per_bit);

t_shifted = t(1:length(Unipolar_Shifted)); % Ensure the time vector matches

%plot after shift
plot_linecodes(Data, Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted,...
                t_shifted, num_bits-1, 'Realization 1 shifted');

% Convert to double for accuracy
Unipolar_Shifted = double(Unipolar_Shifted);
PolarNRZ_Shifted = double(PolarNRZ_Shifted);
PolarRZ_Shifted = double(PolarRZ_Shifted);

%calculate the mean acrros time (question 1)
Unipolar_Mean = calculate_mean(Unipolar_Shifted);
PolarNRZ_Mean = calculate_mean(PolarNRZ_Shifted);
PolarRZ_Mean = calculate_mean(PolarRZ_Shifted);

%plot the mean across time
plot_mean_waveforms(t_shifted, Unipolar_Mean, PolarNRZ_Mean, PolarRZ_Mean, A);

% Compute variance for each code line
Unipolar_Var = calculate_variance(Unipolar_Shifted);
PolarNRZ_Var = calculate_variance(PolarNRZ_Shifted);
PolarRZ_Var = calculate_variance(PolarRZ_Shifted);

%plot the vatiance across time
plot_variance(t_shifted, Unipolar_Var, PolarNRZ_Var, PolarRZ_Var);

% Determine max_lag dynamically
max_lag = size(Unipolar_Shifted, 2) - 1;

%calculate the autocorrelation (question 3)
[Unipolar_AutoCorr, PolarNRZ_AutoCorr, PolarRZ_AutoCorr] = ...
```

```
       compute_stat_autocorr(Unipolar_Shifted, PolarNRZ_Shifted,
PolarRZ_Shifted, max_lag);

%plot the autocorrelation
plot_autocorrelation(Unipolar_AutoCorr, PolarNRZ_AutoCorr, PolarRZ_AutoCorr,
max_lag)

% Compute time autocorrelation
[R_unipolar_nrz_t, R_polar_nrz_t, R_polar_rz_t, taw] =...
    compute_time_autocorr(Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted);

% Plot the time autocorrelation
plot_time_autocorrelation(R_unipolar_nrz_t, R_polar_nrz_t, R_polar_rz_t, taw,
max_lag);

% Compute time mean for each line code
Unipolar_TimeMean = compute_time_mean(Unipolar_Shifted);
PolarNRZ_TimeMean = compute_time_mean(PolarNRZ_Shifted);
PolarRZ_TimeMean = compute_time_mean(PolarRZ_Shifted);

% Plot the time mean
plot_realizations_with_mean(t_shifted, Unipolar_TimeMean, Unipolar_Shifted,
'Unipolar NRZ', 'g');
plot_realizations_with_mean(t_shifted, PolarNRZ_TimeMean, PolarNRZ_Shifted,
'Polar NRZ', 'b');
plot_realizations_with_mean(t_shifted, PolarRZ_TimeMean, PolarRZ_Shifted,
'Polar RZ', 'r');

%Plot time mean vs realization
plot_time_mean_vs_realization(Unipolar_TimeMean, PolarNRZ_TimeMean,
PolarRZ_TimeMean, A);

fs = 100; % Sampling frequency
[BW_unipolar, BW_polarNRZ, BW_polarRZ] = ...
    estimate_all_bandwidths(R_unipolar_nrz_t, R_polar_nrz_t, R_polar_rz_t, fs);


%-----------------------Functions----------------------------

function [Unipolar, PolarNRZ, PolarRZ] = generate_linecodes(Data, A,
samples_per_bit)
    % Ensure input Data is of type int8
    Data = int8(Data);

    % Convert samples_per_bit to double for safe calculations
    samples_per_bitd = double(samples_per_bit);

    % Unipolar NRZ: 0 → 0V, 1 → A
    Unipolar = int8(Data * A);
    Unipolar = repelem(Unipolar, samples_per_bit); % Repeat each bit for
duration

    % Polar NRZ: 0 → -A, 1 → +A
```

```
    PolarNRZ = int8((2 * Data - 1) * A);
    PolarNRZ = repelem(PolarNRZ, samples_per_bit);

    % Polar Return-to-Zero (RZ): Same as Polar NRZ but second half set to 0
    PolarRZ = PolarNRZ;

    % Apply RZ rule: second half of each bit period should be zero
    i = length(Data);  % Start from the last bit
    while i > 0
        end_idx = i * samples_per_bitd;  % Last sample of the bit
        start_idx = end_idx - floor(samples_per_bitd / 2) + 1;  % Start of the
second half
        PolarRZ(start_idx:end_idx) = 0;  % Set the second half of the bit
period to zero
        i = i - 1;  % Move to the previous bit
    end
end

function plot_linecodes(Data, Unipolar, PolarNRZ, PolarRZ, t, num_bits_to_show,
plot_title)
    % Ensure num_bits_to_show does not exceed the actual number of bits
    num_samples_per_bit = ceil(length(t) / length(Data));
    num_samples_to_show = num_bits_to_show * num_samples_per_bit;

    % Trim the signals to display only the required number of bits
    t_show = t(1:num_samples_to_show);
    Unipolar_show = Unipolar(1, 1:num_samples_to_show); % Select row 1
explicitly
    PolarNRZ_show = PolarNRZ(1, 1:num_samples_to_show);
    PolarRZ_show = PolarRZ(1, 1:num_samples_to_show);

    % Convert Data into a sample-wise representation for accurate plotting
    Data_show = repelem(Data(1:num_bits_to_show), num_samples_per_bit);
    Data_t = t(1:length(Data_show)); % Adjust time axis

    % Plot the signals
    figure;
    sgtitle(plot_title); % Set a title for the entire figure

    subplot(4,1,1);
    stairs(Data_t, Data_show, 'k', 'LineWidth', 2);
    title('Orignal Data');
    ylim([-0.5, 1.5]); % Keep the binary level range
    yticks([0 1]);
    yticklabels({'0', '1'});
    grid on;

    subplot(4,1,2);
    stairs(t_show, Unipolar_show, 'b', 'LineWidth', 2);
    title('Unipolar NRZ');
    grid on;

    subplot(4,1,3);
```

```
    stairs(t_show, PolarNRZ_show, 'r', 'LineWidth', 2);
    title('Polar NRZ');
    grid on;

    subplot(4,1,4);
    stairs(t_show, PolarRZ_show, 'm', 'LineWidth', 2);
    title('Polar RZ');
    grid on;

    xlabel('Time (s)');
end

function [Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted] =...
    apply_random_shift_fixed_size(Unipolar_All, PolarNRZ_All, PolarRZ_All,
samples_per_bit)
    % Define parameters
    N_realizations = size(Unipolar_All, 1);  % 500 realizations
    extended_samples = size(Unipolar_All, 2); % 707 samples
    total_samples = 700;  % Fixed output size

    % Initialize shifted matrices
    Unipolar_Shifted = zeros(N_realizations, total_samples, 'int8');
    PolarNRZ_Shifted = zeros(N_realizations, total_samples, 'int8');
    PolarRZ_Shifted = zeros(N_realizations, total_samples, 'int8');

    % Apply random shift to each realization
    for i = 1:N_realizations
        % Generate random shift in range [0, samples_per_bit-1] samples
        random_shift_bits = randi([0, samples_per_bit-1]);

        % Extract shifted region
        Unipolar_Shifted(i, :) = Unipolar_All(i, random_shift_bits+1 :
random_shift_bits+total_samples);
        PolarNRZ_Shifted(i, :) = PolarNRZ_All(i, random_shift_bits+1 :
random_shift_bits+total_samples);
        PolarRZ_Shifted(i, :) = PolarRZ_All(i, random_shift_bits+1 :
random_shift_bits+total_samples);
    end
end

function mean_waveform = calculate_mean(waveform_matrix)
    % Calculates the mean across all realizations without using the mean
function
    % waveform_matrix: Matrix where each row is a realization

    [num_realizations, num_samples] = size(waveform_matrix); % Get matrix
dimensions
    mean_waveform = sum(waveform_matrix, 1) / num_realizations; % Sum and
divide by count

end

function plot_mean_waveforms(t, Unipolar_Mean, PolarNRZ_Mean, PolarRZ_Mean, A)
```

```
    % Function to plot the mean waveforms of different line codes across time
    % Inputs:
    %   t - Time vector
    %   Unipolar_Mean - Mean waveform of Unipolar NRZ
    %   PolarNRZ_Mean - Mean waveform of Polar NRZ
    %   PolarRZ_Mean - Mean waveform of Polar RZ
    %   A - Amplitude limit

    figure;

    subplot(3,1,1);
    plot(t, Unipolar_Mean, 'r', 'LineWidth', 2);
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('Mean of Unipolar NRZ');
    grid on;
    ylim([-A, A]); % Set y-axis limits

    subplot(3,1,2);
    plot(t, PolarNRZ_Mean, 'g', 'LineWidth', 2);
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('Mean of Polar NRZ');
    grid on;
    ylim([-A, A]);

    subplot(3,1,3);
    plot(t, PolarRZ_Mean, 'b', 'LineWidth', 2);
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('Mean of Polar RZ');
    grid on;
    ylim([-A, A]);

    % Add a super title for the whole figure
    sgtitle('Mean of Different Line Codes');
end

function variance_waveform = calculate_variance(waveform_matrix)
     % Calculates the variance across all realizations (column-wise)
    % waveform_matrix: Matrix where each row is a realization (num_realizations
x num_samples)
    % Returns: variance_waveform (1 x num_samples), representing the variance
of each sample point

    % Compute the mean using the previously implemented function
    mean_waveform = calculate_mean(waveform_matrix);

    [num_realizations, num_samples] = size(waveform_matrix); % Get dimensions

    % Ensure mean_waveform is the same size for element-wise subtraction
    mean_waveform = repmat(mean_waveform, num_realizations, 1);
```

```
     % Compute variance manually using the variance formula
    variance_waveform = sum((waveform_matrix - mean_waveform).^2, 1) /
num_realizations; % Population variance

end

function plot_variance(t, Unipolar_Var, PolarNRZ_Var, PolarRZ_Var)
    % Plots the variance of different line codes over time
    % t: Time vector
    % Unipolar_Var, PolarNRZ_Var, PolarRZ_Var: Variance waveforms

    figure;

    subplot(3,1,1);
    plot(t, Unipolar_Var, 'r', 'LineWidth', 2);
    xlabel('Time (s)');
    ylabel('Variance');
    title('Variance of Unipolar NRZ');
    grid on;

    subplot(3,1,2);
    plot(t, PolarNRZ_Var, 'g', 'LineWidth', 2);
    xlabel('Time (s)');
    ylabel('Variance');
    title('Variance of Polar NRZ');
    grid on;

    subplot(3,1,3);
    plot(t, PolarRZ_Var, 'b', 'LineWidth', 2);
    xlabel('Time (s)');
    ylabel('Variance');
    title('Variance of Polar RZ');
    grid on;

    % Add a super title for clarity
    sgtitle('Variance of Different Line Codes');
end

function [Unipolar_AutoCorr, PolarNRZ_AutoCorr, PolarRZ_AutoCorr] =...
    compute_stat_autocorr(Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted,
max_lag)
    % Compute Statistical Autocorrelation for given signals using
calculate_mean
    % Inputs:
    %   Unipolar_Shifted - Shifted signal matrix for Unipolar NRZ
    %   PolarNRZ_Shifted - Shifted signal matrix for Polar NRZ
    %   PolarRZ_Shifted - Shifted signal matrix for Polar RZ
    % Outputs:
    %   Unipolar_AutoCorr - Computed autocorrelation for Unipolar NRZ
    %   PolarNRZ_AutoCorr - Computed autocorrelation for Polar NRZ
    %   PolarRZ_AutoCorr - Computed autocorrelation for Polar RZ
```

```
    % Set x-axis limits dynamically
    x_limit = max_lag / 10;

    % Initialize autocorrelation arrays
    Unipolar_AutoCorr = zeros(1, max_lag + 1);
    PolarNRZ_AutoCorr = zeros(1, max_lag + 1);
    PolarRZ_AutoCorr = zeros(1, max_lag + 1);

    % Compute mean autocorrelation using calculate_mean function
    for i = 0:max_lag
        Unipolar_AutoCorr(i+1) = calculate_mean(Unipolar_Shifted(:, 1) .*
Unipolar_Shifted(:, i+1));
        PolarNRZ_AutoCorr(i+1) = calculate_mean(PolarNRZ_Shifted(:, 1) .*
PolarNRZ_Shifted(:, i+1));
        PolarRZ_AutoCorr(i+1) = calculate_mean(PolarRZ_Shifted(:, 1) .*
PolarRZ_Shifted(:, i+1));
    end

    % Time axis for plotting
    t = -max_lag:max_lag;

    % Compute symmetric autocorrelation values
    Unipolar_AutoCorr = [fliplr(Unipolar_AutoCorr), Unipolar_AutoCorr(2:end)];
    PolarNRZ_AutoCorr = [fliplr(PolarNRZ_AutoCorr), PolarNRZ_AutoCorr(2:end)];
    PolarRZ_AutoCorr = [fliplr(PolarRZ_AutoCorr), PolarRZ_AutoCorr(2:end)];


end

function plot_autocorrelation(Unipolar_AutoCorr, PolarNRZ_AutoCorr,
PolarRZ_AutoCorr, max_lag)
    % Plots the statistical autocorrelation of Unipolar NRZ, Polar NRZ, and
Polar RZ
    % Inputs:
    %   Unipolar_AutoCorr - Autocorrelation for Unipolar NRZ
    %   PolarNRZ_AutoCorr - Autocorrelation for Polar NRZ
    %   PolarRZ_AutoCorr - Autocorrelation for Polar RZ
    %   max_lag - Maximum lag value used for the time axis

    % Time axis
    t = -max_lag:max_lag;

    % Compute x-axis limit
    x_limit = max_lag / 10;

    % Plot results
    figure("name", "Statistical Autocorrelation");

    subplot(3,1,1);
    plot(t, Unipolar_AutoCorr, 'g');
    xlim([-x_limit x_limit]);
    xlabel("Time axis");
    ylabel("Autocorr axis");
```

```
    title("Unipolar NRZ Statistical Autocorrelation");

    subplot(3,1,2);
    plot(t, PolarNRZ_AutoCorr, 'b');
    xlim([-x_limit x_limit]);
    xlabel("Time axis");
    ylabel("Autocorr axis");
    title("Polar NRZ Statistical Autocorrelation");

    subplot(3,1,3);
    plot(t, PolarRZ_AutoCorr, 'r');
    xlim([-x_limit x_limit]);
    xlabel("Time axis");
    ylabel("Autocorr axis");
    title("Polar RZ Statistical Autocorrelation");
end

function TimeMean = compute_time_mean(waveform_matrix)
    % Computes the time mean for each realization of a given waveform
    % Inputs:
    %   waveform_matrix - Matrix where each row represents a realization
    % Output:
    %   TimeMean - Column vector containing the time mean for each realization

    % Compute time mean for each realization (mean along rows)
    TimeMean = sum(waveform_matrix, 2) / size(waveform_matrix, 2);
end

function plot_realizations_with_mean(t_shifted, Signals_TimeMean,
signals_waveform, signal_name, color)
    % Plots the first 3 realizations of a signal in a 3x2 grid and displays
their time means as text.
    %
    % Inputs:
    %   t_shifted          - Time vector
    %   Signals_TimeMean   - Vector of time means (one per realization)
    %   signals_waveform   - Matrix where each row is a realization
    %   signal_name        - Name of the signal (string) for labeling
    %   color              - Plot color (e.g., 'g' for green)

    figure('Name', [signal_name, ' - Realizations and Time Mean']);


    for i = 1:3
        % First Column: Plot the waveform realization
        subplot(3,2,(i-1)*2+1);
        plot(t_shifted, signals_waveform(i,:), color, 'LineWidth', 1.5); % Plot
waveform
        hold on;
        yline(Signals_TimeMean(i), '--k', 'LineWidth', 1.5); % Add time mean
line
        hold off;
        xlabel('Time');
```

```
        ylabel('Amplitude');
        title([signal_name, ' - Realization ', num2str(i)]);
        grid on;

        % Second Column: Display time mean as a text box
        subplot(3,2,(i-1)*2+2);
        axis off; % Hide axes for a clean text display
        text(0.5, 0.5, sprintf('Time Mean:\n%.3f', Signals_TimeMean(i)), ...
            'FontSize', 12, 'FontWeight', 'bold', 'HorizontalAlignment',
'center', 'BackgroundColor', 'w');
    end
end

function plot_time_mean_vs_realization(unipolar_mean, polarNRZ_mean,
polarRZ_mean, A)
    % Function to plot the time mean vs realization index for different signals
separately
    %
    % Inputs:
    % - unipolar_mean: Time mean of Unipolar NRZ (1xN vector)
    % - polarNRZ_mean: Time mean of Polar NRZ (1xN vector)
    % - polarRZ_mean: Time mean of Polar RZ (1xN vector)
    % - A: Amplitude limit for y-axis

    % Define the x-axis (Index based on row size)
    num_realizations = length(unipolar_mean);
    realization_indices = 1:num_realizations;

    % Create a figure for subplots
    figure;

    % Subplot 1: Unipolar NRZ
    subplot(3,1,1);
    plot(realization_indices, unipolar_mean, 'r', 'LineWidth', 2);
    grid on;
    xlabel('Realization Index');
    ylabel('Time Mean');
    title('Unipolar NRZ Time Mean');
    ylim([-A, A]); % Set y-axis limits

    % Subplot 2: Polar NRZ
    subplot(3,1,2);
    plot(realization_indices, polarNRZ_mean, 'g', 'LineWidth', 2);
    grid on;
    xlabel('Realization Index');
    ylabel('Time Mean');
    title('Polar NRZ Time Mean');
    ylim([-A, A]);

    % Subplot 3: Polar RZ
    subplot(3,1,3);
    plot(realization_indices, polarRZ_mean, 'b', 'LineWidth', 2);
    grid on;
```

```
    xlabel('Realization Index');
    ylabel('Time Mean');
    title('Polar RZ Time Mean');
    ylim([-A, A]);

    % Add a super title for the whole figure
    sgtitle('Time Mean vs Realization Index');
end

function [R_unipolar_nrz_t, R_polar_nrz_t, R_polar_rz_t, taw2] = ...
    compute_time_autocorr(UnipolarNRZ, PolarNRZ, PolarRZ)
    % Computes time autocorrelation for each realization separately.
    %
    % Inputs:
    %   UnipolarNRZ - Matrix containing all realizations for Unipolar NRZ
    %   PolarNRZ - Matrix containing all realizations for Polar NRZ
    %   PolarRZ - Matrix containing all realizations for Polar RZ
    %
    % Outputs:
    %   R_unipolar_nrz_t - Time autocorrelation for Unipolar NRZ (each row
computed separately)
    %   R_polar_nrz_t - Time autocorrelation for Polar NRZ (each row computed
separately)
    %   R_polar_rz_t - Time autocorrelation for Polar RZ (each row computed
separately)

    % Get number of realizations and samples
    [num_realizations, num_samples] = size(UnipolarNRZ);

    % Define range of time lags
    max_lag = num_samples - 1; % Maximum lag value
    taw2 = -max_lag:max_lag; % Lag vector

    % Initialize autocorrelation matrices (each row for a realization)
    R_unipolar_nrz_t = zeros(num_realizations, length(taw2));
    R_polar_nrz_t = zeros(num_realizations, length(taw2));
    R_polar_rz_t = zeros(num_realizations, length(taw2));

    % Compute autocorrelation for each realization separately
    for r = 1:num_realizations
        for i = taw2
            M = i + max_lag + 1; % Shift index to fit within array bounds

            % Compute sum for each lag (dot product of signal with shifted
version)
            if i >= 0
                valid_samples = num_samples - i;
                R_unipolar_nrz_t(r, M) = sum(UnipolarNRZ(r, 1:valid_samples) .*
UnipolarNRZ(r, i+1:num_samples)) / valid_samples;
                R_polar_nrz_t(r, M) = sum(PolarNRZ(r, 1:valid_samples) .*
PolarNRZ(r, i+1:num_samples)) / valid_samples;
                R_polar_rz_t(r, M) = sum(PolarRZ(r, 1:valid_samples) .*
PolarRZ(r, i+1:num_samples)) / valid_samples;
```

```
            else
                valid_samples = num_samples + i;
                R_unipolar_nrz_t(r, M) = sum(UnipolarNRZ(r, -i+1:num_samples)
.* UnipolarNRZ(r, 1:valid_samples)) / valid_samples;
                R_polar_nrz_t(r, M) = sum(PolarNRZ(r, -i+1:num_samples) .*
PolarNRZ(r, 1:valid_samples)) / valid_samples;
                R_polar_rz_t(r, M) = sum(PolarRZ(r, -i+1:num_samples) .*
PolarRZ(r, 1:valid_samples)) / valid_samples;
            end
        end
    end
end

function plot_time_autocorrelation(R_unipolar, R_polarNRZ, R_polarRZ, taw,
max_lag)
    % Plots the time autocorrelation of the first realization for each waveform
type.
    %
    % Inputs:
    %   R_unipolar - Matrix containing time autocorrelation for Unipolar NRZ
(each row is a realization)
    %   R_polarNRZ - Matrix containing time autocorrelation for Polar NRZ (each
row is a realization)
    %   R_polarRZ - Matrix containing time autocorrelation for Polar RZ (each
row is a realization)
    %   taw - Vector of lag values
    %   max_lag - Maximum lag value to set axis limits dynamically

    % Set dynamic x-axis limits based on max_lag
    x_limit = max_lag / 10;

    figure('Name', 'Time Autocorrelation');

    % Plot Unipolar NRZ
    subplot(3,1,1);
    plot(taw, R_unipolar(1, :), 'b', 'LineWidth', 1.5);
    grid on;
    xlim([-x_limit x_limit]); % Adjust only the x-axis dynamically
    xlabel('Time Lag (samples)');
    ylabel('Magnitude');
    title('Time Autocorrelation of Unipolar NRZ');

    % Plot Polar NRZ
    subplot(3,1,2);
    plot(taw, R_polarNRZ(1, :), 'r', 'LineWidth', 1.5);
    grid on;
    xlim([-x_limit x_limit]); % Adjust only the x-axis dynamically
    xlabel('Time Lag (samples)');
    ylabel('Magnitude');
    title('Time Autocorrelation of Polar NRZ');

    % Plot Polar RZ
    subplot(3,1,3);
```

```
    plot(taw, R_polarRZ(1, :), 'g', 'LineWidth', 1.5);
    grid on;
    xlim([-x_limit x_limit]); % Adjust only the x-axis dynamically
    xlabel('Time Lag (samples)');
    ylabel('Magnitude');
    title('Time Autocorrelation of Polar RZ');
end

function [BW_unipolar, BW_polarNRZ, BW_polarRZ] = ...
    estimate_all_bandwidths(R_unipolar_nrz, R_polar_nrz, R_polar_rz, fs)
    % Function to estimate the bandwidth for Unipolar NRZ, Polar NRZ, and Polar
RZ
    % using FFT and power spectral density (PSD) method.

    % Extract the first realization
    R_unipolar_nrz = R_unipolar_nrz(:,1);
    R_polar_nrz = R_polar_nrz(:,1);
    R_polar_rz = R_polar_rz(:,1);

    % Number of samples (adjust dynamically)
    n = length(R_unipolar_nrz);

    % Compute FFT coefficients
    unipolar_nrz_coe = fft(R_unipolar_nrz) / n;
    polar_nrz_coe = fft(R_polar_nrz) / n;
    polar_rz_coe = fft(R_polar_rz) / n;

    % Compute magnitude of FFT (PSD estimation)
    amplitude_unipolar_nrz = abs(unipolar_nrz_coe);
    amplitude_polar_nrz = abs(polar_nrz_coe);
    amplitude_polar_rz = abs(polar_rz_coe);

    % Frequency axis mapping (centered around 0)
    freq_axis = (-n/2:n/2-1) * (fs / n);  % Corrected frequency scaling

    % Shift FFT for centered display
    amp_unipolar_nrz = fftshift(amplitude_unipolar_nrz);
    amp_polar_nrz = fftshift(amplitude_polar_nrz);
    amp_polar_rz = fftshift(amplitude_polar_rz);

    % Estimate bandwidths using -3dB method
    BW_unipolar = find_bandwidth(amp_unipolar_nrz, freq_axis);
    BW_polarNRZ = find_bandwidth(amp_polar_nrz, freq_axis);
    BW_polarRZ = find_bandwidth(amp_polar_rz, freq_axis);

    %{
    % Display Bandwidth Values
    disp(['BW Unipolar NRZ: ', num2str(BW_unipolar), ' Hz']);
    disp(['BW Polar NRZ: ', num2str(BW_polarNRZ), ' Hz']);
    disp(['BW Polar RZ: ', num2str(BW_polarRZ), ' Hz']);
    %}

    % Plot PSDs
```

```
    figure('Name', 'Power Spectral Density');

    subplot(3,1,1);
    plot(freq_axis, amp_unipolar_nrz, 'g');
    ylim([0 max(amp_unipolar_nrz)*1.1]); xlim([-fs/2 fs/2]);
    grid on;
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title('PSD of Unipolar NRZ');

    subplot(3,1,2);
    plot(freq_axis, amp_polar_nrz, 'b');
    ylim([0 max(amp_polar_nrz)*1.1]); xlim([-fs/2 fs/2]);
    grid on;
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title('PSD of Polar NRZ');

    subplot(3,1,3);
    plot(freq_axis, amp_polar_rz, 'r');
    ylim([0 max(amp_polar_rz)*1.1]); xlim([-fs/2 fs/2]);
    grid on;
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title('PSD of Polar RZ');
end

function BW = find_bandwidth(PSD, freq_axis)
    % Function to find the -3dB bandwidth from PSD
    PSD_max = max(PSD);        % Find max power
    threshold = PSD_max / 2;   % -3dB threshold (half power)

    % Find indices where PSD is above threshold
    valid_indices = find(PSD >= threshold);

    % Check if we found any valid points
    if isempty(valid_indices)
        BW = 0; % If no valid range is found, return 0
        return;
    end
    % Ensure indices are within valid bounds
    min_index = max(1, min(valid_indices));  % Ensure it's at least 1
    max_index = min(length(freq_axis), max(valid_indices));  % Ensure it's
within bounds

    % Compute bandwidth as the difference between first and last crossing
points
    BW = abs(freq_axis(max_index) - freq_axis(min_index));
end
```