Department of Electronics and
Electrical Communications Engineering
Faculty of Engineering - Cairo University

Digital Communications Project # 1

Team Members

Name	Code	Role
Passant Nabil Fawzy	136	Data generating & Calculating Mean
Gamal Allaa Eldin	144	Calculating PSD & BW
Roaa Atef Mohamed	211	Data generating & Calculating Mean
Shehab Eldin Tarek	224	Calculating Autocorrelation function
Omar Ahmed Ragab	244	Calculating Autocorrelation function

Table of Contents

Table of Figures

I. Problem Description

The objective of this project is to generate and analyze waveforms for transmitting binary data using three different line coding techniques (Unipolar NRZ, Polar NRZ, and Unipolar RZ) For each coding type, an ensemble of 500 waveforms, each containing 100 random bits, will be generated. These waveforms will be studied to complete the following tasks:

Calculate the statistical mean of the ensemble to find the average signal level.

Check if the random process is stationary by examining whether its mean and autocorrelation stay consistent over time.

Compute the ensemble autocorrelation function Rx(τ) to analyze the relationship between signal values at different time shifts.

Select one waveform and compute its **time mean and autocorrelation function** to explore individual behavior.

Determine if the process is ergodic by comparing the time averages of a single waveform with the ensemble averages.

Estimate the bandwidth of the transmitted signal by analyzing its frequency components.

These steps will help in comparing the line codes in terms of their statistical behavior and efficiency in signal transmission using software-defined methods.

II. Introduction

Software-Defined Radio (SDR) is a modern approach to communication systems where traditional hardware functions are replaced by software implementations. Instead of relying on analog circuitry, SDR allows waveforms to be generated and processed using programmable code. This flexibility makes it possible to easily change modulation methods, coding schemes, and transmission techniques without altering the hardware. SDR also supports data transmission over both wireless and wired channels, where antennas can be replaced by cables and digital line coding techniques can be applied.

In this project, SDR concepts are used to simulate the transmission of binary data using three different line coding techniques: Unipolar NRZ, Polar NRZ, and Unipolar RZ. Each coding scheme shapes the waveform in a specific way, affecting how the signal behaves over time and frequency. By generating random binary data and converting it into waveforms using MATLAB, the output can be treated as a random process. This allows for statistical and spectral analysis to better understand the behavior and performance of each line code in a software-defined environment.

III. Control Flags

Control Flags	Values
Number of bits	100 bits
Number of realizations	500 realizations
Number of samples	7 samples per bit
Time shift	random number from 1 to 7

IV. Generation of data

- Code snippet:

```
num_realization = 500;
num_bits = 100;
```

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

- Explain:

To generate **500 realizations** of a random signal, where each realization consists of **100 bits**, we use the **randi** function .it generates binary values (0 or 1) for each bit. The second argument represents the number of realizations (**500**), while the third argument specifies the number of bits per realization (**100**). This results in a **500 × 100** matrix, where each row recognizes the random binary signal.

V. Creation of Polar NRZ ensemble

- Code snippet:

```
%% 1. Polar NRZ (0 -> -A, 1 -> A)
%convert data to polar_nrz
Polar_NRZ = (2 * Data - 1) * A;
% Expand each bit by repeating its value 7 times(sampling)
Polar_NRZ_out = zeros(num_realization, total_samples);
for i = 1:num_realization
    Polar_NRZ_reshaped = repmat(Polar_NRZ(i, :), samples_per_bit, 1);
    Polar_NRZ_out(i, :) = reshape(Polar_NRZ_reshaped, 1, []);
end
```

- Explain:

We represent **Polar NRZ** using **A,-A**

1. Convert the random signals, which are 0 and 1 to **A,-A** to represent **Polar NRZ** using the formula **((2*Data-1)*A)**
2. Expand each bit by repeating its value 7 times (sampling) using **repmat** function to repeat each bit 7 times
3. Reshaping the repeated bits into a row vector using **reshape**

4. VI. Creation of unipolar ensemble

- Code snippet:

```
%% 2. Unipolar(0 -> 0, 1 -> A)
% Convert data to Uni-Polar
Uni_Polar = Data * A;
% Expand each bit by repeating its value 7 times
Uni_Polar_out = zeros(num_realization, total_samples);
for i = 1:num_realization
    Uni_Polar_reshaped = repmat(Uni_Polar(i, :), samples_per_bit, 1);
    Uni_Polar_out(i, :) = reshape(Uni_Polar_reshaped, 1, []);
end
```

- Explain:

We represent **Unipolar** using **A,0**

1. Convert the random signals, which are 0 and 1 to **A,0** to represent **Unipolar** using the formula (**Data*A)**

2. Expand each bit by repeating its value 7 times (sampling) using **repmat** function to repeat each bit 7 times
3. Reshaping the repeated bits into a row vector using **reshape**

VII. Creation of Polar RZ ensemble

- Code snippet:

```
%% 3. Polar RZ
%%%% Convert to Polar RZ (0 -> -A, 1 -> A for first half, then 0 for second half) %%%%
Polar_RZ = (2 * Data - 1) * A;
% Expand each bit by repeating its value over 7 times
Polar_RZ_out = zeros(num_realization, total_samples);
for i = 1:num_realization
    Polar_RZ_expanded = repmat(Polar_RZ(i, :), samples_per_bit, 1);

    % Set second half of each bit to zero
    for j = 1:num_bits
        Polar_RZ_expanded(floor(samples_per_bit / 2) + 1:end, j) = 0;
    end

    Polar_RZ_out(i, :) = reshape(Polar_RZ_expanded, 1, []);
end
```

- Explain:

We represent **Polar RZ** using **A,-A** for first half, and **0** for the second half

1. Convert the random signals, which are 0 and 1, to **A,-A** to represent **Polar RZ** using the formula **((2*Data-1)*A)**
2. Expand each bit by repeating its value 7 times (sampling) using the **repmat** function to repeat each bit 7 times
3. Set the second half of the bit to zero using the formula **Polar_RZ_expanded(floor(samples_per_bit / 2) + 1:end, j) = 0;**
4. Reshaping the repeated bits into a row vector using **reshape**

VIII. Applying random initial time shifts for each waveform

- Code snippet:

```
% Generate random delay in the range of the bit samples
delay_samples = randi([1, 7], num_realization, 1);
```

```
% Apply delay to Polar NRZ using circular shift
Polar_NRZ_delayed = zeros(size(Polar_NRZ_out));
for i = 1:num_realization
    Polar_NRZ_delayed(i, :) = circshift(Polar_NRZ_out(i, :), delay_samples(i));
end
```

- Explain:

1. To generate a random delay for **500 realizations**, we use the **randi** function The delay is randomly selected within a range of **1 to 7**, corresponding to the total number of samples per bit. This ensures that each realization experiences a unique delay within the specified range
2. Adding randomness to the signal processing using **circshift** function

○ Results

- Polar NRZ:

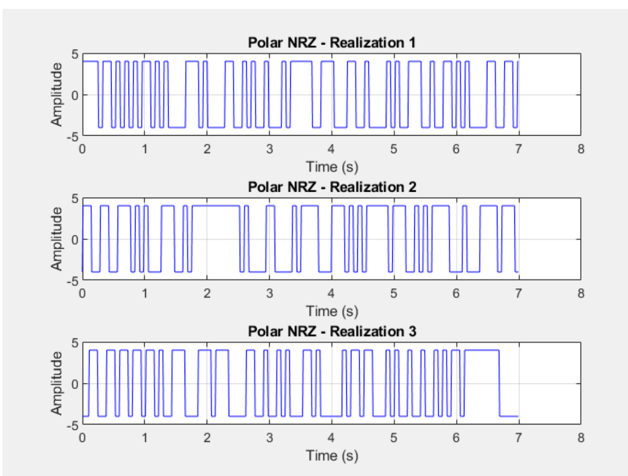

Figure 1:Polar NRZ realizations after delay

- Unipolar:

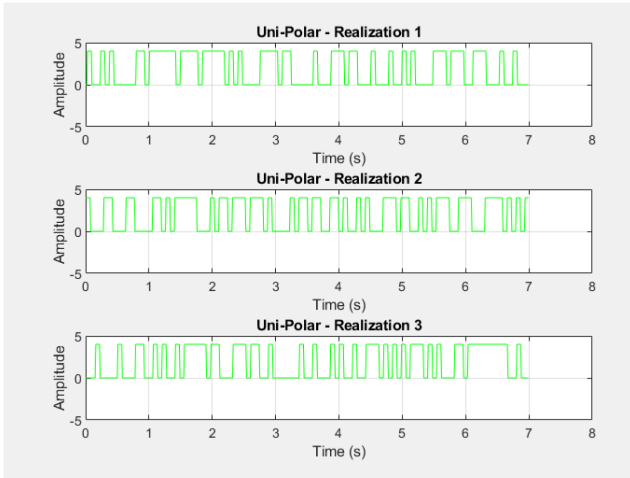

Figure 2:Unipolar realizations after delay

- Polar RZ:

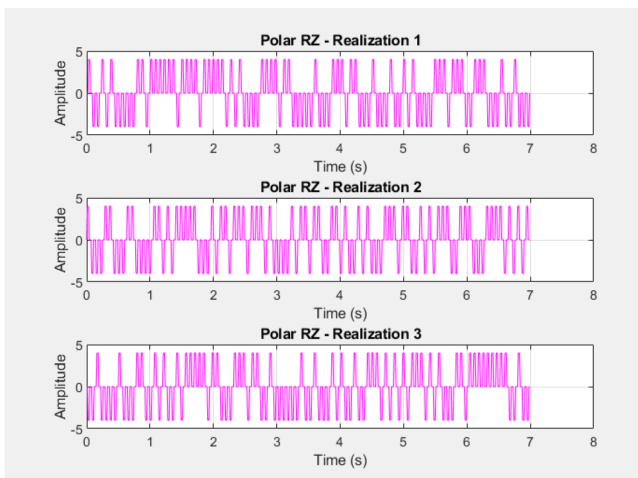

Figure 3:Polar RZ realizatins after delay

IX. Calculating the statistical mean

- Code snippet:

```
%% Function to Calculate Statistical Mean (Ensemble Mean)
function statistical_mean = calculate_statistical_mean(signal, num_realization, total_samples)
    Sum_signal = zeros(1, total_samples);
    for t = 1:total_samples
        for i = 1:num_realization
            Sum_signal(t) = Sum_signal(t) + signal(i, t);
        end
    end
    statistical_mean = Sum_signal / num_realization;
end
```

- Explain:

This function calculates the statistical mean (ensemble mean) of a signal over multiple realizations.

- The function iterates through each time sample and adds up the corresponding signal values from all realizations.
- After summing the values, each sample is divided by the total number of realizations to compute the statistical mean.

○ Results

- Polar NRZ:

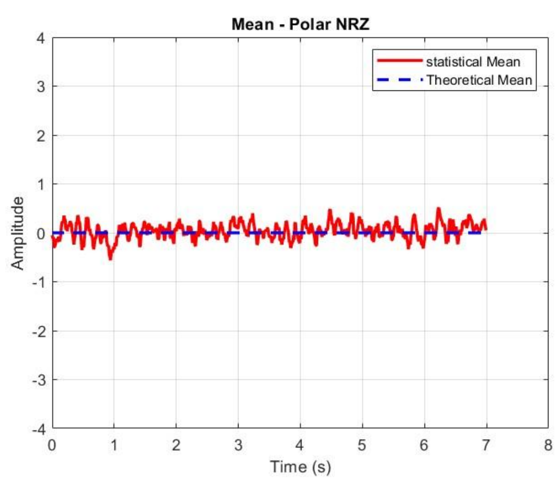

Figure 4:Polar NRZ statistical mean

- ○ Comment: The statistical mean is around zero, as expected for the Polar NRZ line code.

- Unipolar:

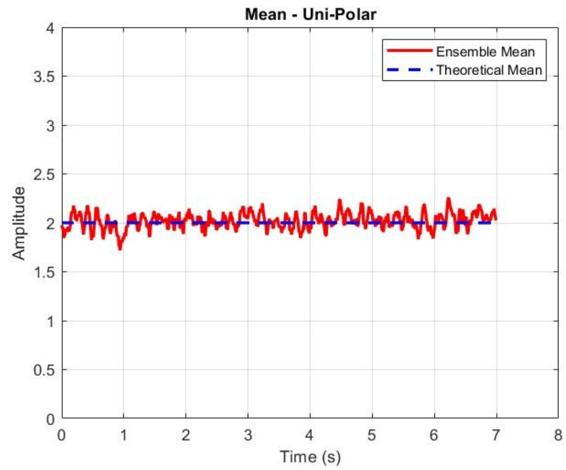

Figure 5:Unipolar statistical mean

  - Comment: The statistical mean is around two (A / 2), as expected for the Unipolar line code.

- Polar RZ:

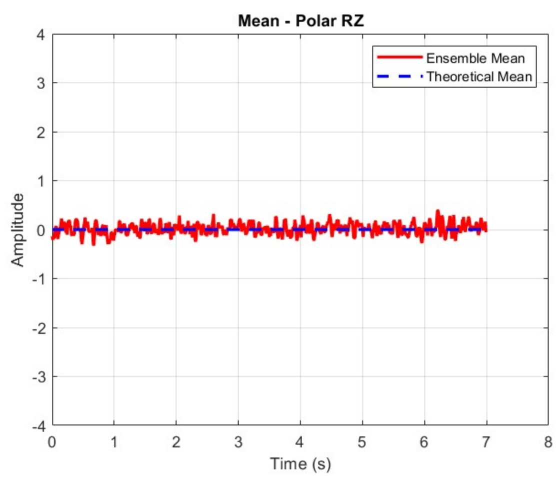

Figure 6:Polar RZ statistical mean

  - Comment: The statistical mean is around zero, as expected for the Polar RZ line code.

General comment on mean graphs: The small variations in the mean occur due to the finite number of samples, bits, and realizations. Increasing these parameters will result in a more stable mean that closely matches the theoretical value.

XII. Calculating the statistical autocorrelation

- **Code Snippet (theoretical)**

```
%%---------------Theoritical AutoCorrelation---------------------
%theoritical PolarNRZ AutoCorrelation is (A^2) @tau=0 only and zero
everywhere else
theoretical_auto_corr_PolarNRZ =@(tau) (A^2) * exp(-abs(tau) / 7);
%theoritical UniPolarNRZ AutoCorrelation is (A^2/2) @tau=0 only and
(A^2/4) everywhere else
theoretical_auto_corr_UnipolarNRZ = @(tau) (A^2 / 4) + ((A^2 / 4) * exp(-
abs(tau) / 7));

%theoritical PolarRZ AutoCorrelation is (A^2 * 4/7) @tau=0 only and zero
everywhere else
theoretical_auto_corr_PolarRZ = @(tau) (A^2 * (4/7)) * exp(-abs(tau) /
7);

tau_values = 1:700;

PolarNRZ_Theoritical = arrayfun(theoretical_auto_corr_PolarNRZ,
tau_values);
UnipolarNRZ_Theoritical = arrayfun(theoretical_auto_corr_UnipolarNRZ,
tau_values);
PolarRZ_Theoritical = arrayfun(theoretical_auto_corr_PolarRZ,
tau_values);

%%---------------------------------------------------------------
```

- We start by preparing the theoretical autocorrelation for all the 3 required line codes

- **Explanation :**

- We got the theoretical using hand analysis as follows:

For polar NRZ :

- Assume it's Bernoulli distribution
- we assumed that the theoretical autocorrelation function is an continuous function not discrete to see its ideal values

P(A) = $\frac{1}{2}$, P(-A) = $\frac{1}{2}$

R(0) = $E\left(X^2(t)\right) = \frac{1}{2}\,(-A)^2 + \frac{1}{2}\,(A)^2 = A^2$

X(t)	$X(t+\tau)$	X(t)$X(t+\tau)$	
A	A	A^2	Prob = 0.25
-A	A	$-A^2$	Prob = 0.25
A	-A	$-A^2$	Prob = 0.25
-A	-A	A^2	Prob = 0.25

$$R(\tau) = E\big(X(t)X(t+\tau)\big) = 2 \times \frac{1}{4} \times A^2 + 2 \times \frac{1}{4} \times -A^2 = Zero$$

$$R(\tau)\begin{cases} A^2 * e^{\frac{-|\tau|}{7}}, for\ \tau < 7 \\ 0 \quad , for\ \tau > 7 \end{cases}$$

➔ **The Exponential as a function has max at $\boldsymbol{\tau = 0}$ and decreases with slope (1/bit duration) till reach its dc value at infinity.**

For polar RZ:

-Assume it's Bernoulli distribution

P(A) = $\frac{1}{2}$, P(-A) = $\frac{1}{2}$

R(0) = $E\big(X^2(t)\big) = \left[\frac{1}{2}\ (-A)^2 + \frac{1}{2}\ (A)^2\right] * bit\ duration = A^2 * \frac{4}{7}$

Bit duration is 4/7 as we have 4 samples high out of 7 samples

X(t)	$X(t+\tau)$	X(t)$X(t+\tau)$	
A	-A	$-A^2$	Prob = 0.25
-A	A	$-A^2$	Prob = 0.25
A	A	A^2	Prob = 0.25
-A	-A	A^2	Prob = 0.25

$$R(\tau) = E\big(X(t)X(t+\tau)\big) = 2 \times \frac{1}{4} \times A^2 + 2 \times \frac{1}{4} \times -A^2 = Zero$$

$$R(\tau)\begin{cases} \frac{4}{7}\ A^2 * e^{\frac{-|\tau|}{7}}, for\ \tau < 7 \\ 0 \quad , for\ \tau > 7 \end{cases}$$

➔ **The Exponential as a function has max at $\boldsymbol{\tau = 0}$ and decreases with slope (1/bit duration) till reach its dc value at infinity.**

For Unipolar NRZ:

-Assume it's Bernoulli distribution

P(A) = $\frac{1}{2}$, P(0) = $\frac{1}{2}$

R(0) = $E\left(X^2(t)\right) = \left[\frac{1}{2}\ (0)^2 + \frac{1}{2}\ (A)^2\right] = \frac{A^2}{2}$

Bit duration is 4/7 as we have 4 samples high out of 7 samples

X(t)	$X(t+\tau)$	X(t)$X(t+\tau)$	
0	0	0	Prob = 0.25
0	A	0	Prob = 0.25
A	0	0	Prob = 0.25
A	A	A^2	Prob = 0.25

$$R(\tau) = E\left(X(t)X(t+\tau)\right) = \frac{1}{4} \times A^2 + 3 \times 0\ = \frac{A^2}{4}$$

$$R(\tau)\begin{cases} \frac{1}{2}A^2 * e^{\frac{-|\tau|}{7}}, for\ \tau < 7 \\ \frac{A^2}{4} \quad * e^{\frac{-|\tau|}{7}} \quad , for\ \tau > 7 \end{cases}$$

➔ **The Exponential as a function has max at $\boldsymbol{\tau = 0}$ and decreases with slope (1/bit duration) till reach its dc value at infinity.**

- **Code Snippet (Simulation of AutoCorrealtion) :**

```
%%--------------------Auto Correlation simulation @ multiple values of tau---
----------------------
%---------------------for polar NRZ
pnrz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Polar_NRZ_delayed,"Polar NRZ Auto Correlation @
tau = [0:687]");
plot_sim_and_theor(pnrz_auto_corr_multiple_tau,PolarNRZ_Theoritical,"Polar
NRZ AutoCorrelation","Theoretical Polar NRZ AutoCorrelation","PolarNRZ
AutoCorr. Simulated vs Theoritical");
%---------------------
%---------------------for polar RZ
prz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Polar_RZ_delayed,"Polar RZ Auto Correlation @ tau
= [0:687]");
plot_sim_and_theor(prz_auto_corr_multiple_tau,PolarRZ_Theoritical,"Polar RZ
AutoCorrelation","Theoretical Polar RZ AutoCorrelation","Polar RZ AutoCorr.
Simulated vs Theoritical");
%---------------------

%---------------------for Unipolar NRZ
upnrz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Uni_Polar_delayed,"UniPolar NRZ Auto Correlation
@ tau = [0:687]");
plot_sim_and_theor(upnrz_auto_corr_multiple_tau,UnipolarNRZ_Theoritical,"Uni
polar NRZ AutoCorr","Theoretical Unipolar NRZ AutoCorr","UnipolarNRZ
AutoCorr. Simulated vs Theoritical");
%%--------------------Auto Correlation simulation @ multiple values of tau---
----------------------
%---------------------for polar NRZ
pnrz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Polar_NRZ_delayed,"Polar NRZ Auto Correlation @
tau = [0:687]");
plot_sim_and_theor(pnrz_auto_corr_multiple_tau,PolarNRZ_Theoritical,"Polar
NRZ AutoCorrelation","Theoretical Polar NRZ AutoCorrelation","PolarNRZ
AutoCorr. Simulated vs Theoritical");
%---------------------
%---------------------for polar RZ
prz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Polar_RZ_delayed,"Polar RZ Auto Correlation @ tau
= [0:687]");
plot_sim_and_theor(prz_auto_corr_multiple_tau,PolarRZ_Theoritical,"Polar RZ
AutoCorrelation","Theoretical Polar RZ AutoCorrelation","Polar RZ AutoCorr.
Simulated vs Theoritical");
%---------------------

%---------------------for Unipolar NRZ
upnrz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Uni_Polar_delayed,"UniPolar NRZ Auto Correlation
@ tau = [0:687]");
plot_sim_and_theor(upnrz_auto_corr_multiple_tau,UnipolarNRZ_Theoritical,"Uni
polar NRZ AutoCorr","Theoretical Unipolar NRZ AutoCorr","UnipolarNRZ
AutoCorr. Simulated vs Theoritical");
```

- **Function of Autocorrelation with Multiple tau values [0:500]:**

```
function autocorr_from_t1_or_t2= autocorr_func_multiple_tau(matrix,
plot_title)
      autocorr_t1 = zeros(1, 700);
      autocorr_t2 = zeros(1, 700);
        % Loop over each RV
    for sample = 1:2
        autocorr_current = zeros(1, 700);
        for tau = 0:699
            sample_values1 = matrix(:, sample); % Extract the reference
column
            if (tau == 699 && sample == 2) % Special case for circular shift
                sample_values2 = matrix(:, sample-1);
            else
                sample_values2 = matrix(:, sample+tau);
            end
            squared_values = sample_values1 .* sample_values2;
            autocorr_current(tau+1) = sum(squared_values) / 500;
        end
        if sample == 1
            autocorr_t1 = autocorr_current;
        else
            autocorr_t2 = autocorr_current;
        end
    end
    % Flip and mirror for plotting
    autocorr_flipped(1, :) = [fliplr(autocorr_t1), autocorr_t1];
    autocorr_flipped(2, :) = [fliplr(autocorr_t2), autocorr_t2];
    % Plot the results
    % Combined plot for comparison
    figure;
    plot(-699:700, autocorr_flipped(1, :), 'b-', 'LineWidth', 1.5);
    hold on;
    plot(-699:700, autocorr_flipped(2, :), 'r-', 'LineWidth', 1.5);
    hold off;
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title([plot_title ' - Comparison']);
    legend('Autocorr @ t=t1', 'Autocorr @ t=t2');
    grid on;
    figure;
    plot(-699:700, autocorr_flipped(1, :), 'b-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title([plot_title ' - Starting from t1']);
    grid on;
    figure;
    plot(-699:700, autocorr_flipped(2, :), 'r-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title([plot_title ' - Starting from t2']);
    grid on;
    % Return the avg of the autocorrelation from t1 & t2 for PSD calcuations
    autocorr_from_t1_or_t2 = (autocorr_t1 + autocorr_t2) / 2;
end
```

- **Plot simulated & theoretical Function** :

```
function plot_sim_and_theor(matrix1, matrix2, title1, title2, title3)
    figure;
    hold on;

    x = -699:700;

    % Flip matrix and concatenate it with itself
    matrix1_flipped = flip(matrix1);
    matrix1_padded = [matrix1_flipped, matrix1];
    matrix2_flipped = flip(matrix2);
    matrix2_padded = [matrix2_flipped, matrix2];

    plot(x, matrix1_padded, 'b-', 'LineWidth', 1.5);
    plot(x, matrix2_padded, 'r--', 'LineWidth', 1.5);

    legend(title1, title2, 'Location', 'best');
    xlabel('tau');
    ylabel('AutoCorr Amplitude');
    title(title3);
    grid on;
    hold off;
end
```

- **Explanation**:

  Here we are taking each line code modulation and getting the autocorrelation for each one and returning the output to an array that will be used in the "plot_sim_and_theor" to be able to compare between the simulation and the theoretical.

  The "autocorr_func_multiple_tau" function makes two main tasks:

  -First get the autocorrelation by applying the equation

  $R(t) = E(X(t) * X(t+\tau))$

  Where X(t = t1) is the first column and tau represent the shift

  mainly we need at starting from t= t1 to compare with the theoretical and tau [0:699]

  after applying the column Dot product we divide by 500 to get the expectation of the dot product.

  We used tau [0:688] to avoid the peaks that appears at the end of the graphs it could be avoided by also adding more columns to the matrix.

  Then we plot the autocorrelation starting from t= t1 and use it in the "plot_sim_and_theor" to compare the results

- second the other main task of the "autocorr_func_multiple_tau" is that it gets the autocorrelation starting from t=t2 to compare it with the autocorrelation starting from t=t1 where if they are the same it means that only time shift affects the autocorrelation function which will help us to prove that the function is **Wide-Sense Stationary (WSS)**.

- **Results**:

Polar NRZ:

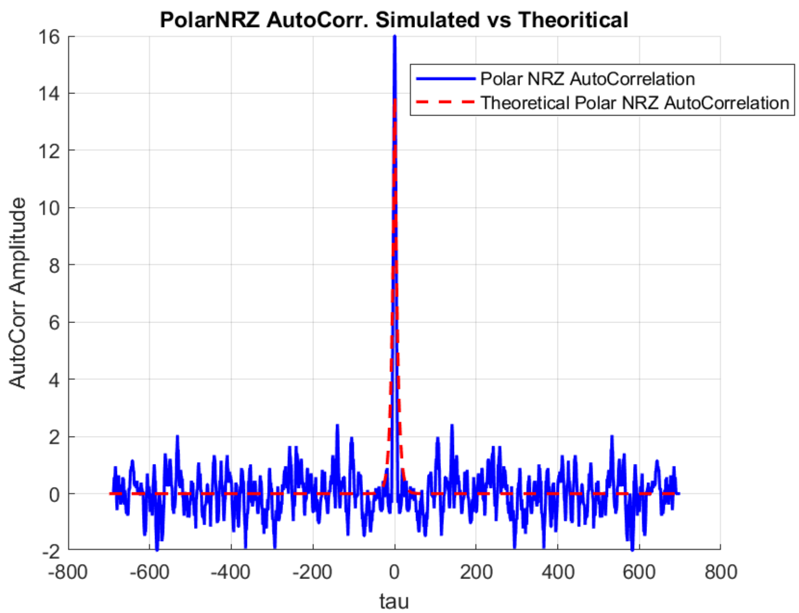

Figure 7: Polar NRZ AutoCorr. Sim Vs Theoretical

➔ **As we expected the simulation is close to the theoretical (dashed line)**

➔ **The Plot of the Autocorrelation is of tau [0:687] not [0:699] to avoid high peaks occurring in the autocorrelation between sample RV at t1 and the last columns**

➔ **The Average Power = R(0) ≈ 16**

➔ **The DC Power = R(∞) ≈ 0**

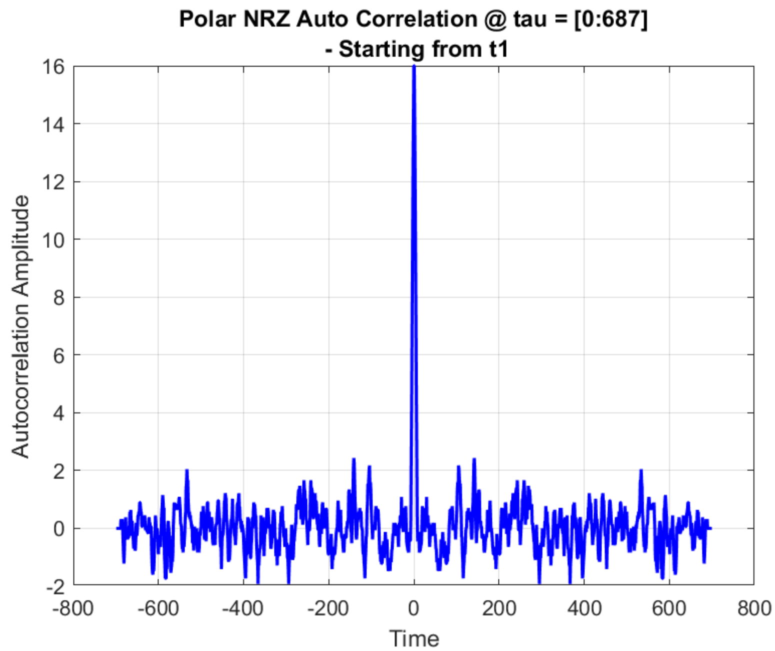

Figure 8: Polar NRZ AutoCorr. for t = t1 & tau [0:687]

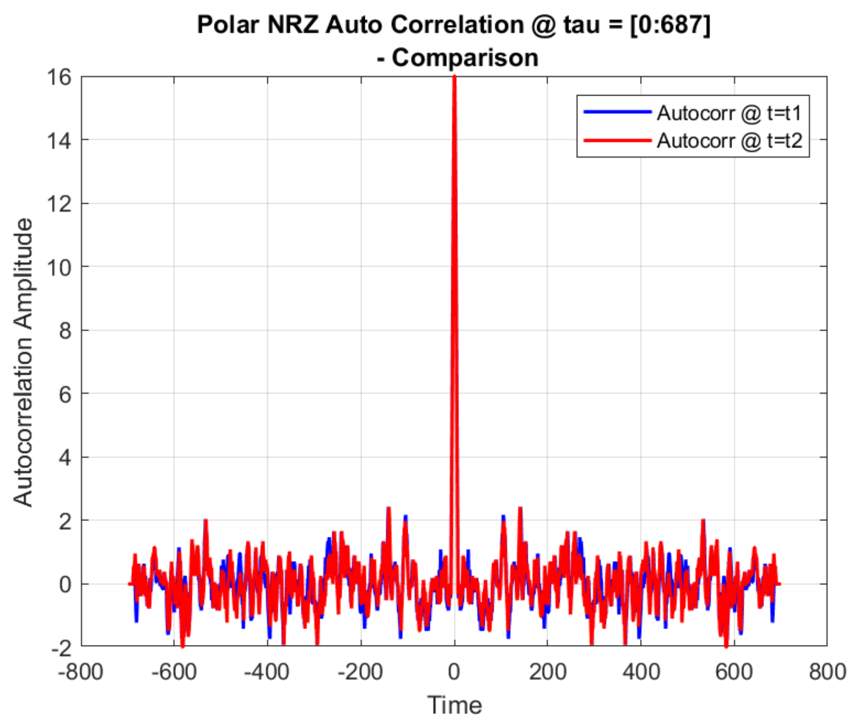

Figure 9: Polar NRZ AutoCorr. t=t1 vs t=t2

- ➔ **As expected the autocorrelation starting from t=t1 & t=t2 are almost the same**
- ➔ **which proves that <u>only time shift affects the autocorrelation function</u>**

Polar RZ:

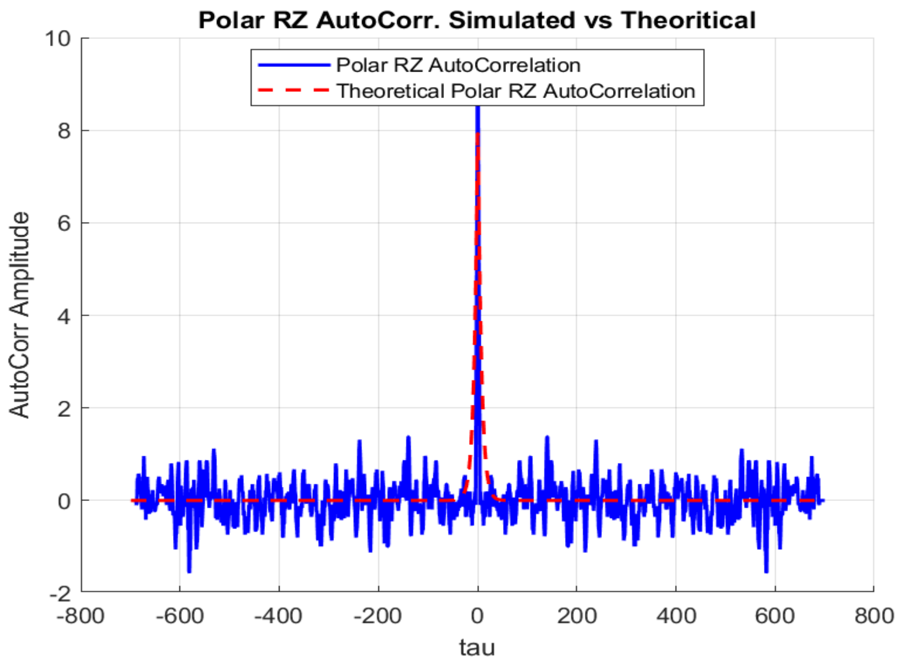

Figure 10:Polar RZ AutoCorr. Sim Vs Theoretical

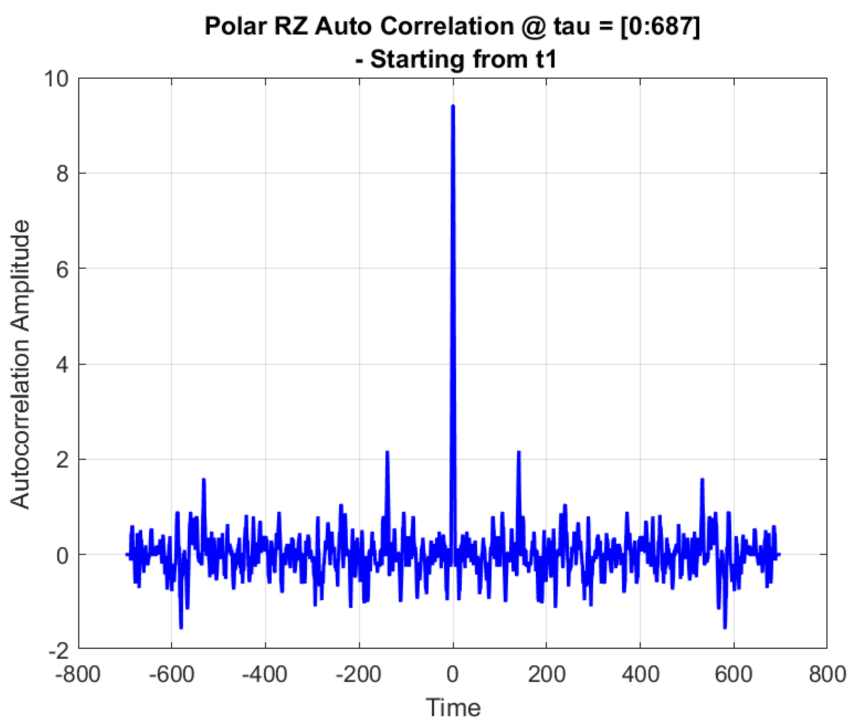

Figure 11:Polar RZ AutoCorr. for t =t1 & tau [0:687]

➔ **As we expected the simulation at fig.10 is close to the theoretical (dashed line)**

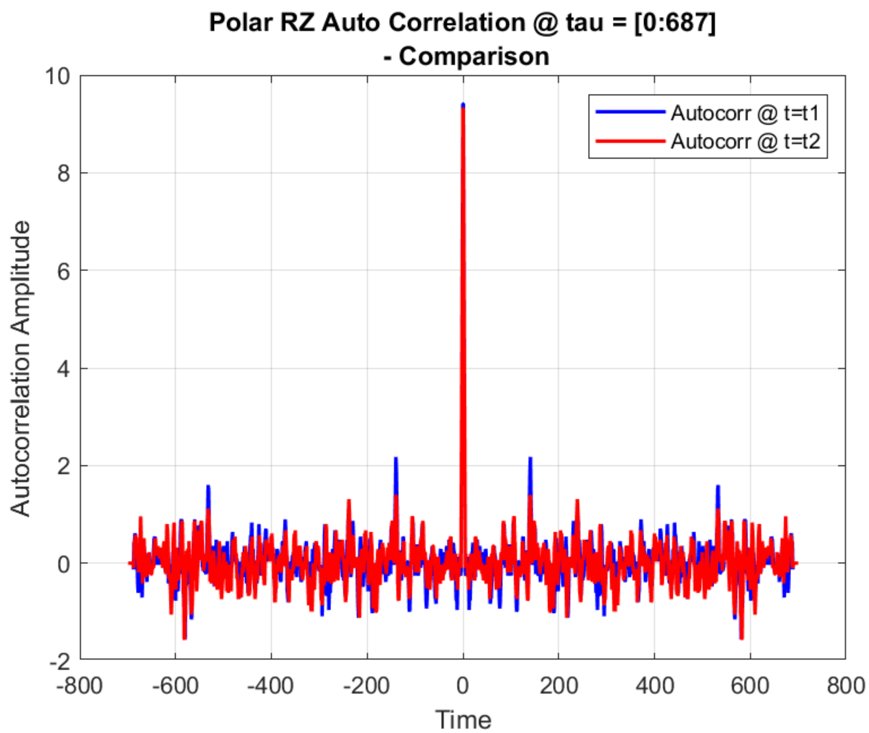

Figure 12:Polar RZ AutoCorr. t=t1 vs t=t2

- ➔ **As expected the autocorrelation starting from t=t1 & t=t2 are almost the same which proves <u>that only time shift affects the autocorrelation function</u>**
- ➔ **The Average Power = R(0) ≈ 9.2**
- ➔ **The DC Power = R(∞) ≈ 0**

Unipolar NRZ:

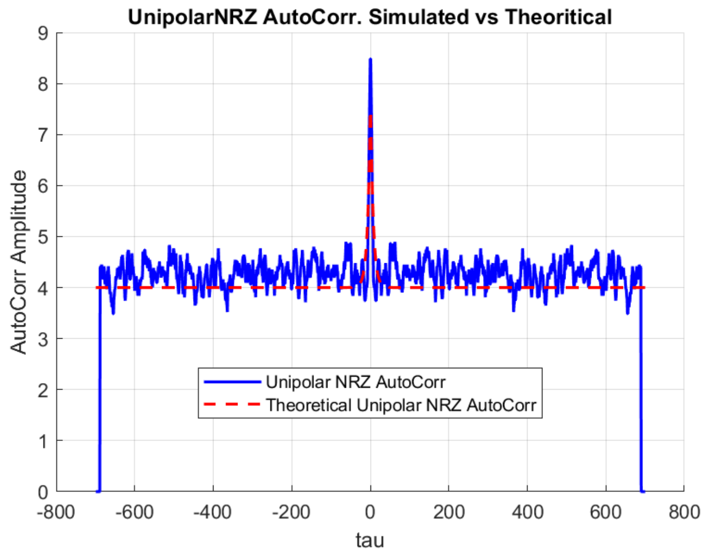

Figure 13:Unipolar NRZ AutoCorr. Sim Vs Theoretical

➔ **As we expected the simulation at fig.13 is so close to the theoretical (dashed line)**

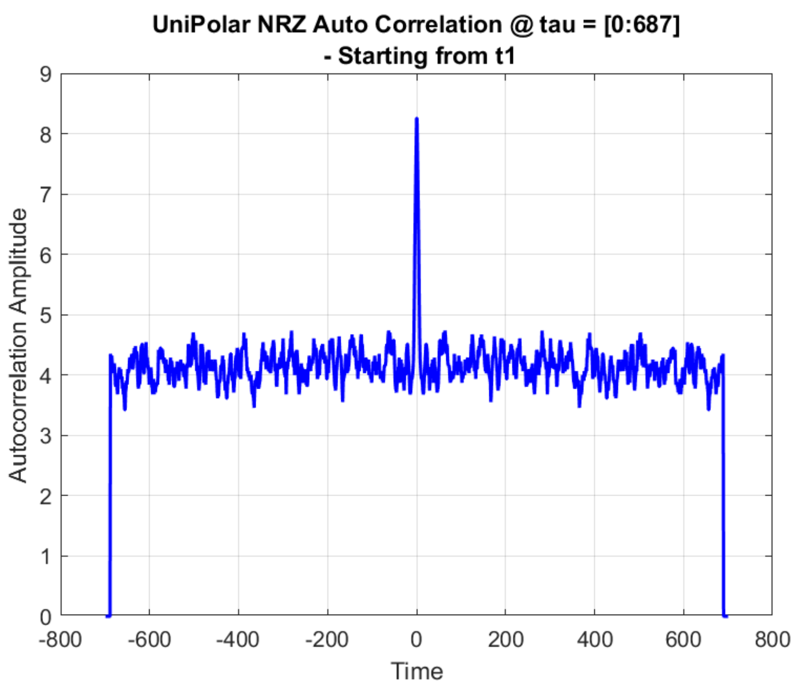

Figure 14:UniPolar NRZ AutoCorr. for t =t1 & tau [0:687]

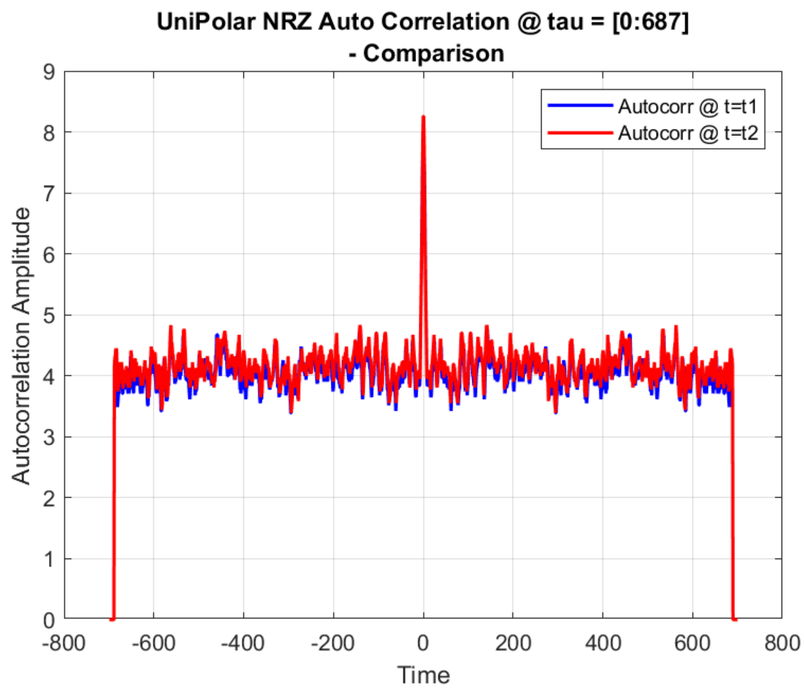

Figure 15:Unipolar NRZ AutoCorr. t=t1 vs t=t2

- ➔ **As expected the autocorrelation starting from t=t1 & t=t2 are almost the same which proves <u>that only time shift affects the autocorrelation function</u>**
- ➔ **The Average Power = R(0) ≈ 8**
- ➔ **The DC Power = R(∞) ≈ 4**

XIV. Is the process stationary?

XV. Computing the time mean and auto correlation

- Code snippet:

```
%% Function to Calculate Time Mean
function time_mean = Calculate_time_mean(signal, num_realization, total_samples)
    Sum_time_signal = zeros(num_realization, 1);
    for i = 1:num_realization
        for t = 1:total_samples
            Sum_time_signal(i) = Sum_time_signal(i) + signal(i, t);
        end
    end
    time_mean = Sum_time_signal / total_samples;
end
```

- **Explain:**

This function calculates the time mean of a signal over multiple realizations.

- The function iterates through each time sample within a single realization and adds up all the signal values for that realization.
- After summing the values, the total sum for each realization is divided by the total number of time samples to compute the time mean.

○ Results

- Polar NRZ:

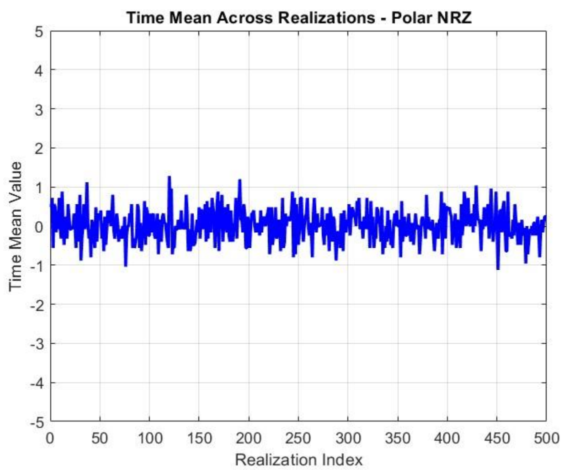

Figure 16:Polar NRZ time mean

➔ **Comment: The time mean is around zero, as expected for the Polar NRZ line code.**

- **Unipolar:**

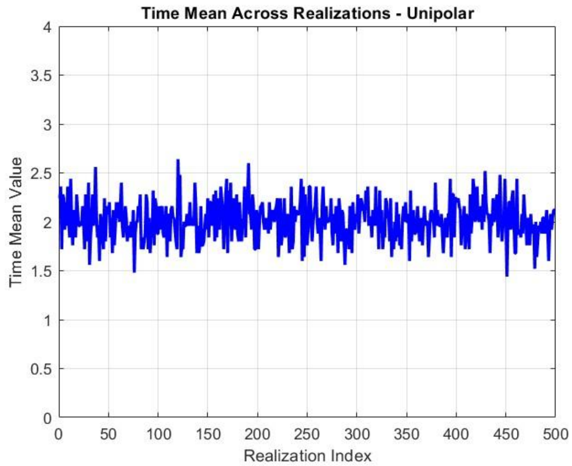

Figure 17:Unipolar time mean

➔ **Comment: The time mean is around to (A / 2), as expected for the Unipolar line code.**

- **Polar RZ:**

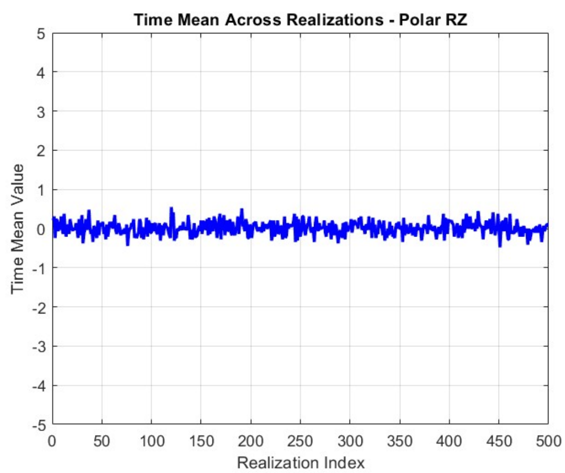

Figure 18:Polar RZ time mean

➔ **Comment: The time mean is around zero, as expected for the Polar RZ line code.**

- ➔ **Here we saw that the mean is nearly varying around the theoretical mean across the time at each line code & the autocorrelation function is nearly the same however starting from t1 or t2 as example so the autocorrelation function depends only on the time lag between the two random variables, So the Process is WSS**

- ➔ **Since the time mean is nearly constant across the 500 realizations as shown in fig.16,17,18 that will provide us with the ergodicity check**

XVI. IS the random process ergodic?

- **Code snippet**

```
%definging an 1-D array aa a test values of tau to test the Ergodicity
ergodic_tau_values = 0:num_realization;
%%-------------Ergotic polar NRZ------------------------------------------
%% 4.isergodic @tau=0 , Self AutoCorr for each Realization
pnrz_isergodic=isergotic(Polar_NRZ_delayed,"Avg. Self-Correlation of all
Polar NRZ Realizations @tau=0");
plot_autocorr(polar_nrz_corr_zero_shift,pnrz_isergodic,"Polar NRZ Auto
Correlation @ tau = 0","Avg. Self-Correlation of all Polar NRZ
Realizations");
%% 5.Ergodic AutoCorrelation @tau=0:500
ergodic_auto_corr(Polar_NRZ_delayed,ergodic_tau_values,"AutoCorrelation
across Time Vs AutoCorrelation across ensamble - Polar NRZ");

%%-------------Ergotic polar RZ-------------------------------------------
%% 4.isergodic @tau=0 , Self AutoCorr for each Realization
prz_isergodic=isergotic(Polar_RZ_delayed,"Avg. Self-Correlation of all Polar
RZ Realizations @tau=0");
plot_autocorr(polar_rz_corr_zero_shift,prz_isergodic,"Polar RZ Auto
Correlation @ tau = 0","Avg. Self-Correlation of all Polar RZ
Realizations");

%% 5.Ergodic AutoCorrelation @tau=0:500
ergodic_auto_corr(Polar_RZ_delayed,ergodic_tau_values,"AutoCorrelation
across Time Vs AutoCorrelation across ensamble - Polar RZ");

%%-------------Ergotic Unipolar NRZ---------------------------------------
%% 4.isergodic @tau=0 , Self AutoCorr for each Realization
upnrz_isergodic=isergotic(Uni_Polar_delayed,"Avg. Self-Correlation of all
UniPolar NRZ Realizations @tau=0");
plot_autocorr(unipolar_nrz_corr_zero_shift,upnrz_isergodic,"UniPolar NRZ
Auto Correlation @ tau = 0","Avg. Self-Correlation of all UniPolar NRZ
Realizations");
%%----------------------------------------------------------------------
%% 5.Ergodic AutoCorrelation @tau=0:500
ergodic_auto_corr(Uni_Polar_delayed,ergodic_tau_values,"AutoCorrelation
```

```
across Time Vs AutoCorrelation across ensamble - UniPolar NRZ");
%%--------------------------------------------------------------------------
```

- **Explanation :**

  - Here to test the Ergodicity we have to check first that the average self autocorr. Of all realizations is nearly constant and that is checked by **Isergotic Function** to see that the average of self autocorr of the three line codes
  - Then we will check that the autocorr. between the realizations with lag of values [0:500] and the autocorr. between the Sample RV's with lag of values [0:500] and then compare if they are nearly equal to check ergodicity or not

• Functions :-

- **Isergotic Function:**

```
function ergodic_value = isergotic(matrix, plot_title)
    [rows, cols] = size(matrix);
    row_auto_corr = zeros(1, rows); % Initialize correctly sized array

    for r = 1:rows
        wavefrom = matrix(r, :); % Extract row
        squaredSum = sum(wavefrom .^ 2);
        row_auto_corr(r) = squaredSum / 700; % Store result correctly
    end

    ergodic_value = sum(row_auto_corr) / rows; % Normalize by total rows

    % Plot the ergodic value
    figure;
    plot(1:cols, ones(1, cols) * ergodic_value, 'r-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    yMin = floor(0 * 10) / 10; % Round down to nearest 0.1
    yMax = ceil(20 * 10) / 10;  % Round up to nearest 0.1
    ylim([yMin, yMax]); % Set limits
    yticks(yMin:0.5:yMax); % Set ticks at intervals of 0.1
    title(plot_title);
    grid on;
end
```

- Here the **Isergotic Function** takes the data array and performs a dot product for each realization (each row) with itself (tau = 0) and plotting it against its index [0:500] as they are 500 realizations

- **plot_autocorr:**

```
function plot_autocorr(autocorr_RVs, ergodic_value, title_autocorr,
title_ergodic)
    cols = length(autocorr_RVs);
    time = 1:cols;
    figure;
    plot(time, autocorr_RVs, 'b-', 'LineWidth', 1.5);
    hold on;
    plot(time, ones(1, cols) * ergodic_value, 'r-', 'LineWidth', 1.5);

    xlabel('Time');
    ylabel('Autocorrelation Amplitude');
    title([title_autocorr, ' vs ', title_ergodic]);
    legend(title_autocorr, title_ergodic);
    grid on;
    hold off;
end
```

- **ergodic_auto_corr :**

```
function ergodic_auto_corr(matrix2D, tau_values, plot_title)
    [rows, cols] = size(matrix2D);
    num_lags = length(tau_values);
    col_corr = zeros(1, num_lags);
    row_corr = zeros(1, num_lags);
    %column dot product
    for i = 1:num_lags
        tau = tau_values(i);
        if tau < cols
            col1 = matrix2D(:, 1);
            col2 = matrix2D(:, tau + 1);
            col_corr(i) = dot(col1, col2) / rows;
        else
            col_corr(i) = NaN; % If lag exceeds dimensions
        end
    end
    % row dot product
    for i = 1:num_lags
        tau = tau_values(i);
        if tau < rows
            row1 = matrix2D(1, :);
            row2 = matrix2D(tau + 1, :);
            row_corr(i) = dot(row1, row2) / cols;
        else
            row_corr(i) = NaN;
        end
    end
    % Plot results
    figure;
    hold on;
    plot(tau_values, col_corr, 'r-', 'LineWidth', 2, 'DisplayName', 'Across
Ensamble Correlation');
    plot(tau_values, row_corr, 'b-', 'LineWidth', 2, 'DisplayName', 'Across
Time Correlation');
```

```
    legend;
    title(plot_title);
    xlabel('Lag (Tau)');
    ylabel('Normalized Dot Product');
    grid on;
    hold off;
end
```

- **Explanation :**

In our code to prove ergodicity we have two conditions

First:

- The average time mean across all the realizations is nearly the same shown in fig.16,17,18 and nearly equal to the ensemble mean which we have shown in fig.4,5,6

- That condition we have already checked, as we have proven that the process is WSS

Second :

- Average the autocorrelation function value for each waveform with the other waveforms is nearly equal to the autocorrelation function value between the random variable across all the realizations to the other random variables (depending only on the time lag)

- Analysis :

i. Ergodicity (Mean) :

- Assume data has Bernoulli distribution with probability (p)
- Mean across for each waveform :

- For Polar RZ = $[A * (p) + (-A) * (1-p)] * \frac{4}{7} = A(2p-1) * \frac{4}{7}$
- For polar NRZ = $A * (p) + (-A) * (1-p) \ = A(2p-1)$
- For Unipolar NRZ = $A(p) + (0)(1-p) = Ap$
- Assume p = ½ (Bernoulli distribution)

$$\mu_{PRZ} = 0 \mid \mu_{PNRZ} = 0 \mid \mu_{Unipolar} = \frac{A}{2}$$

- Mean across each random variable will have Bernoulli distribution too, which has p=0.5 and for each line code the statistical mean will be :

- For Polar RZ = $[A * (p) + (-A) * (1-p)] * \frac{4}{7} = A(2p-1) * \frac{4}{7}$
- For polar NRZ = $A * (p) + (-A) * (1-p) \ = A(2p-1)$
- For Unipolar NRZ = $A(p) + (0)(1-p) = Ap$
- Assume p = ½ (Bernoulli distribution)

$$m_{PRZ} = 0 \mid m_{PNRZ} = 0 \mid m_{Unipolar} = \frac{A}{2}$$

- **Therefore the Statistical mean is equal to the time mean**

ii. Ergodicity (Autocorrelation) :

- Each wave form has Bernoulli distribution
- The realization are identically independent distributed
- Therefore autocorrelation depends on mean only :

$$R(\tau) = \left(E\big(X(t)\big)\right)^2 = \mu^2 = [A * p + (-A) * (1-p)]^2$$
$$R(\tau) = A^2(2p-1)^2$$

So, for every line code (p = 0.5):

$$R_{PRZ}(\tau) = \mu_{PRZ}{}^2 = zero$$
$$R_{PNRZ}(\tau) = \mu_{PNRZ}{}^2 = zero$$
$$R_{Unipolar}(\tau) = \mu_{Unipolar}{}^2 = \frac{A^2}{4}$$

For Second condition:

- We want to prove that the statistical autocorrelation equals the time autocorrelation.

  And we will get it from comparing the graphs

- **Results** :

Polar NRZ:

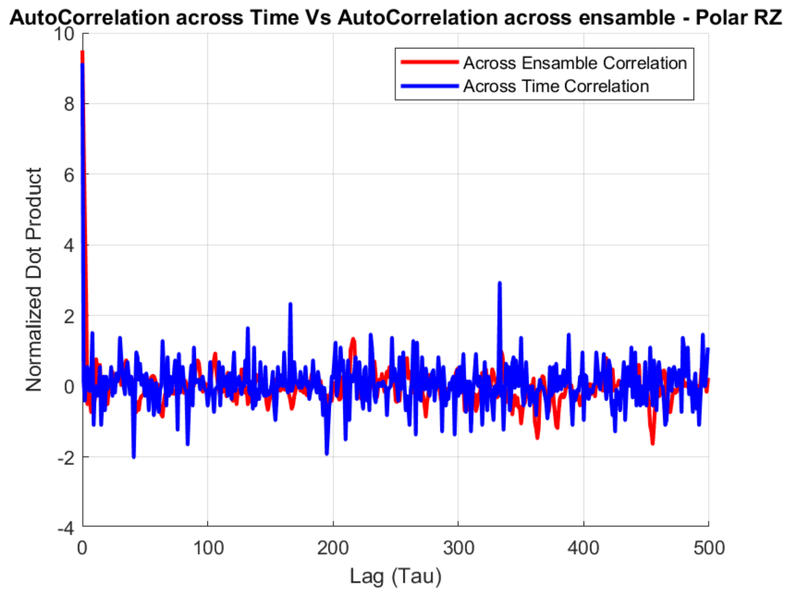

Figure 19 PolarNRZ Time Autocorrelation vs ensemble autocorrelation

➔ **As the time autocorrelation and ensemble autocorrelation are almost the same depending on the time lag only , <u>Therefore the process is Ergodic</u>**

Polar RZ:

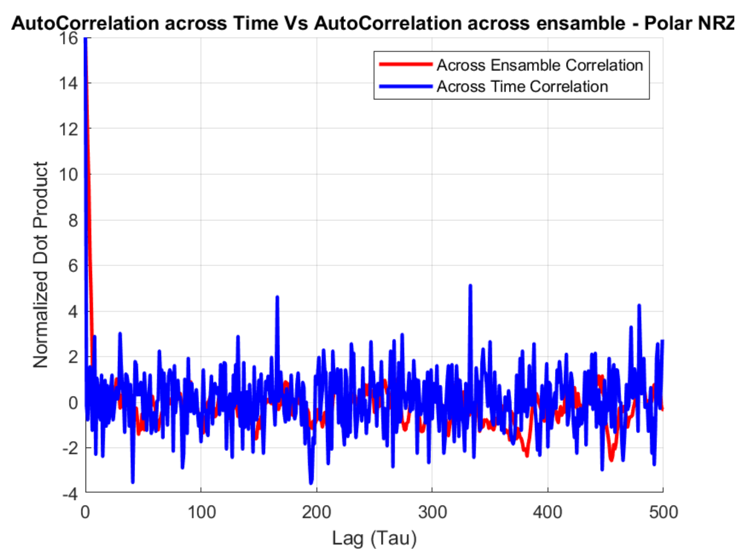

Figure 20 PolarRZ Time Autocorrelation vs ensemble autocorrelation

➔ **As the time autocorrelation and ensemble autocorrelation are almost the same depending on the time lag only , <u>Therefore the process is Ergodic</u>**

Unipolar NRZ:

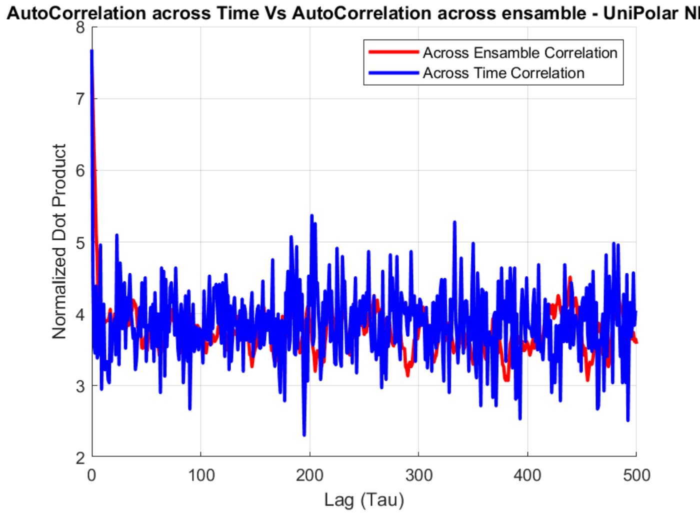

Figure 21 UniPolarNRZ Time Autocorrelation vs ensemble autocorrelation

➔ **As the time autocorrelation and ensemble autocorrelation are almost the same depending on the time lag only , Therefore the process is Ergodic**

XVII.Plotting the PSD of the ensemble

PSD Formula [1]

PSD calculated by applying Fourier Transform to the auto correlation function:

PSD(S_{xx}(w) = F.T{RX(τ)}

Line code Type	S_{xx}(w)
POLAR NRZ	Fs*(A^2 * bitDuration) * (sinc(freq_axis * bitDuration).^2)
POLAR RZ	Fs* (A^2 * bitDuration* 16 / 49) * (sinc(freq_axis * bitDuration*4 / 7).^2)
UNIPOLAR	Fs* (A^2 * bitDuration / 4) * (sinc(freq_axis * bitDuration).^2) + 10*Fs* (A^2 / 4) * δ(fs))

Code Snippet

```
% Ensure FFT matches the length of the autocorrelation function
N = length(pnrz_auto_corr_multiple_tau);

% Define the frequency axis for the Fourier Transform (centered at 0 Hz)
freq_axis = linspace(-Fs/2, Fs/2, N);

% PSD is the Fourier Transform (FFT) of the autocorrelation function

PSD_Polar_NRZ = abs(fftshift(fft(pnrz_auto_corr_multiple_tau, N))); % Polar NRZ
PSD_Uni_Polar = abs(fftshift(fft(upnrz_auto_corr_multiple_tau, N))); % Unipolar NRZ
PSD_Polar_RZ = abs(fftshift(fft(prz_auto_corr_multiple_tau, N))); % Polar RZ
```

This MATLAB script computes the Power Spectral Density (PSD) of Polar NRZ, Unipolar NRZ, and Polar RZ line coding schemes. The PSD is obtained by applying the Fast Fourier Transform (FFT) to their autocorrelation functions, helping analyze bandwidth and spectral characteristics.

The frequency axis is defined from -Fs/2 to Fs/2 with a step size of Fs/N, ensuring proper spectral representation. The fftshift() function centers the spectrum around 0 Hz, and the absolute value is taken to obtain the magnitude spectrum.

The value of N (equal to the length of the autocorrelation function) determines the number of FFT points. It directly affects the frequency resolution of the PSD (Δf = Fs/N): a larger N results in a finer frequency resolution and smoother spectrum, while a smaller N can lead to broader spectral features and visible ripples due to limited sample points.

Results

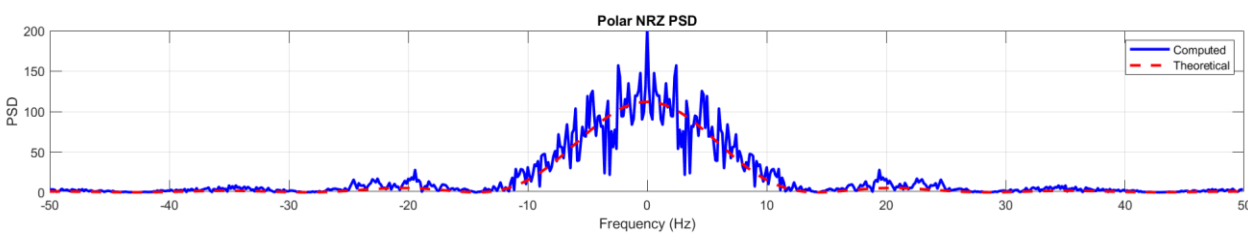

Figure 22: Fourier Transform of Polar NRZ auto correlation

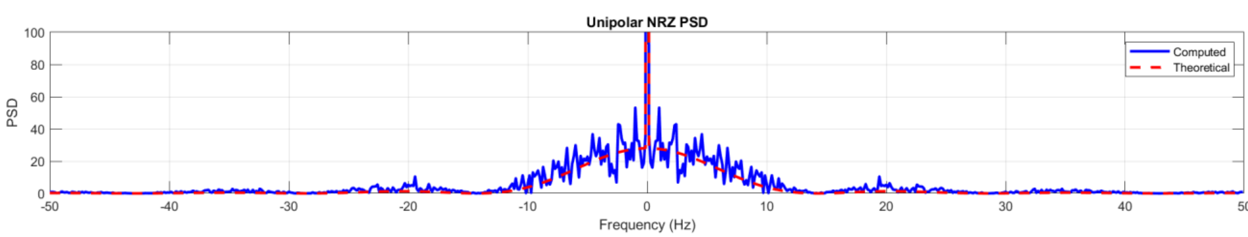

Figure 23: Fourier Transform of Unipolar NRZ auto correlation

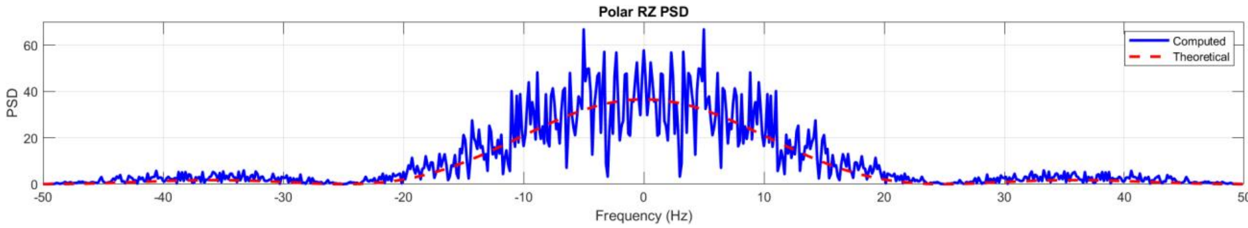

Figure 24: Fourier Transform of Polar RZ auto correlation

Comment

The ripples observed in the computed PSD curves are due to the finite number of samples and realizations. Despite these fluctuations, the overall shape matches the theoretical PSD.

XVIII.What is the Bandwidth of the transmitted signal?

For the simulation I calculated the BW when the values of the PSD functions are = 10^{-6} of the max values of polar NRZ, unipolar and Polar RZ, I also used the graph of the PSD .

Code Snippet

```
% Define a strict threshold close to zero for polar_NRZ (0.1m% of max PSD)
threshold_polar_NRZ = max(PSD_Polar_NRZ) * 1e-6;

% Find the first null (first zero-crossing or minimum point)
null_index_NRZ = find(PSD_Polar_NRZ(ceil(N/2):end) < threshold_polar_NRZ, 1,
'first') + ceil(N/2) - 1;
B_simulated_Polar_NRZ = abs(freq_axis(null_index_NRZ));

% Define a strict threshold close to zero for uni_polar_NRZ (0.1m% of max PSD)
threshold_uni_polar_NRZ = max(PSD_Uni_Polar) * 1e-6;

null_index_Uni = find(PSD_Uni_Polar(ceil(N/2):end) < threshold_uni_polar_NRZ, 1,
'first') + ceil(N/2) - 1;
B_simulated_Uni_Polar = abs(freq_axis(null_index_Uni));


% Define a strict threshold close to zero for polar_RZ (0.1m% of max PSD)
threshold_polar_RZ= max(PSD_Polar_RZ) * 1e-6;

null_index_prz = find(PSD_Polar_RZ(ceil(N/2):end) < threshold_polar_RZ, 1, 'first')
+ ceil(N/2) - 1;
B_simulated_polar_RZ = abs(freq_axis(null_index_prz));
```

Results

From Graph:

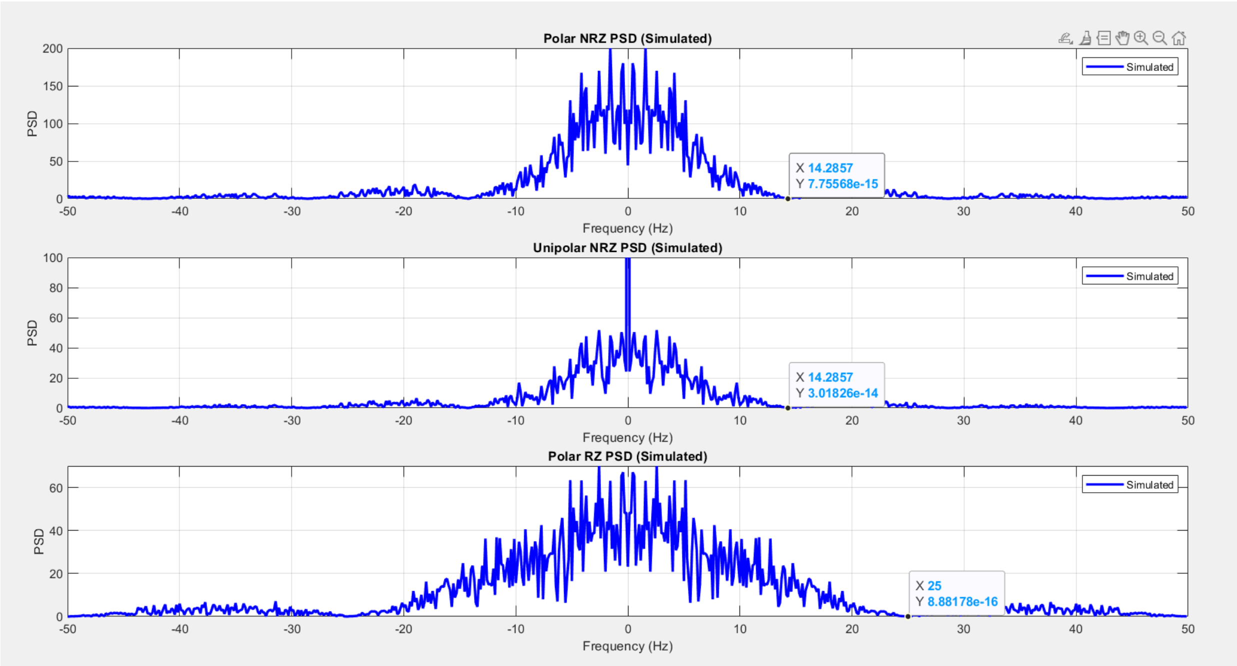

Figure 25: Signals BW from the PSD graph

Signal type	Theoretical BW	Simulated BW
POLAR NRZ	$\frac{1}{Tb}$ = 14.2857 Hz	14.2857 Hz
POLAR RZ	$\frac{7}{4}\, x \frac{1}{Tb} = 25\ Hz$	25 Hz
UNIPOLAR	$\frac{1}{Tb}$ = 14.2857 Hz	14.2857 Hz

XIX. Full MATLAB code :

```
clc; clear; close all;
%define basic parameters
A = 4;
num_realization = 500;
num_bits = 100;
samples_per_bit = 7;
bitDuration = 70e-3;
Fs = 1 / (bitDuration / samples_per_bit);
total_samples = num_bits * samples_per_bit;
% Generate random binary data
Data = randi([0, 1], num_realization, num_bits);
% Generate random delay in the range of the bit samples
delay_samples = randi([1, 7], num_realization, 1);
% Time axis for plotting
time = (0:total_samples-1) / Fs;

%% 1. Polar NRZ (0 -> -A, 1 -> A)
%convert data to polar_nrz
Polar_NRZ = (2 * Data - 1) * A;
% Expand each bit by repeating its value 7 times(sampling)
Polar_NRZ_out = zeros(num_realization, total_samples);
for i = 1:num_realization
    Polar_NRZ_reshaped = repmat(Polar_NRZ(i, :), samples_per_bit, 1);
    Polar_NRZ_out(i, :) = reshape(Polar_NRZ_reshaped, 1, []);
end
% Apply delay to Polar NRZ using circular shift
Polar_NRZ_delayed = zeros(size(Polar_NRZ_out));
for i = 1:num_realization
    Polar_NRZ_delayed(i, :) = circshift(Polar_NRZ_out(i, :),
delay_samples(i));
end
% Plotting random realizations  against time
figure;
for i = 1:3
    subplot(3,1,i);
    plot(time, Polar_NRZ_delayed(i, :), 'b');
    title(['Polar NRZ - Realization ', num2str(i)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
    grid on;
    xlim([0, 8]);
    ylim([-5, 5]);
end

% Compute means of PNRZ
stat_mean_NRZ = calculate_statistical_mean(Polar_NRZ_delayed,
num_realization, total_samples);
theoretical_NRZ = 0; % Theoretical mean
time_mean_NRZ = Calculate_time_mean(Polar_NRZ_delayed, num_realization,
total_samples);

% Plot ensamble mean for Polar NRZ
figure;
plot(time, stat_mean_NRZ, 'r', 'LineWidth', 2); hold on;
```

```
plot(time, theoretical_NRZ * ones(1, total_samples), 'b--', 'LineWidth', 2);
title('Mean - Polar NRZ');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Ensemble Mean', 'Theoretical Mean');
grid on;
xlim([0, 8]);
ylim([-2, 2]);
% Plot time mean across realizations for Polar NRZ
figure;
plot(1:num_realization, time_mean_NRZ, 'b', 'LineWidth', 2);
title('Time Mean Across Realizations - Polar NRZ');
xlabel('Realization Index');
ylabel('Time Mean Value');
grid on;
ylim([-5, 5]);

%% 2. Unipolar(0 -> 0, 1 -> A)
% Convert data to Uni-Polar
Uni_Polar = Data * A;
% Expand each bit by repeating its value 7 times
Uni_Polar_out = zeros(num_realization, total_samples);
for i = 1:num_realization
    Uni_Polar_reshaped = repmat(Uni_Polar(i, :), samples_per_bit, 1);
    Uni_Polar_out(i, :) = reshape(Uni_Polar_reshaped, 1, []);
end
% Apply delay to Uni_Polar using circular shift
Uni_Polar_delayed = zeros(size(Uni_Polar_out));
for i = 1:num_realization
    Uni_Polar_delayed(i, :) = circshift(Uni_Polar_out(i, :),
delay_samples(i));
end
% Plotting random realizations  against time
figure;
for i = 1:3
    subplot(3,1,i);
    plot(time, Uni_Polar_delayed(i, :), 'g');
    title(['Uni-Polar - Realization ', num2str(i)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
    grid on;
    xlim([0, 8]);
     ylim([-5, 5]);
end
% Compute means of UPNRZ
stat_mean_Unipolar = calculate_statistical_mean(Uni_Polar_delayed,
num_realization, total_samples);
theoretical_Unipolar = 0.5 * A; % Theoretical mean
time_mean_Unipolar = Calculate_time_mean(Uni_Polar_delayed, num_realization,
total_samples);

% Plot ensamble mean for Unipolar
figure;
plot(time, stat_mean_Unipolar, 'r', 'LineWidth', 2); hold on;
plot(time, theoretical_Unipolar * ones(1, total_samples), 'b--',
'LineWidth', 2);
title('Mean - Unipolar');
```

```
xlabel('Time (s)');
ylabel('Amplitude');
legend('Ensemble Mean', 'Theoretical Mean');
grid on;
xlim([0, 8]);
ylim([0, 4]);
% Plot time mean across realizations for Unipolar
figure;
plot(1:num_realization, time_mean_Unipolar, 'b', 'LineWidth', 2);
title('Time Mean Across Realizations - Unipolar');
xlabel('Realization Index');
ylabel('Time Mean Value');
grid on;
ylim([0, 4]);

%% 3. Polar RZ
% Convert to Polar RZ (0 -> -A, 1 -> A for 4 samples, then 0 for 3 samples)
Polar_RZ = (2 * Data - 1) * A;

Polar_RZ_out = zeros(num_realization, total_samples);
% Iterate over realizations
for i = 1:num_realization
    % Expand each bit to cover the required samples
    Polar_RZ_expanded = repmat(Polar_RZ(i, :), samples_per_bit, 1);

    % Set last 3 samples of each bit to zero (Polar RZ format)
    for j = 1:num_bits
        Polar_RZ_expanded(5:end, j) = 0;  % First 4 samples remain, last 3
go to zero
    end

    % Flatten to a single row
    Polar_RZ_out(i, :) = reshape(Polar_RZ_expanded, 1, []);
end


% Apply delay to Polar_RZ using circular shift
Polar_RZ_delayed = zeros(size(Polar_RZ_out));
for i = 1:num_realization
    Polar_RZ_delayed(i, :) = circshift(Polar_RZ_out(i, :),
delay_samples(i));
end
%plotting
figure;
for i = 1:3
    subplot(3,1,i);
    plot(time, Polar_RZ_delayed(i, :), 'm');
    title(['Polar RZ - Realization ', num2str(i)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
    grid on;
    xlim([0, 8]);
     ylim([-5, 5]);
end
% Compute means of PRZ
stat_mean_RZ = calculate_statistical_mean(Polar_RZ_delayed, num_realization,
total_samples);
```

```
theoretical_RZ = 0; % Theoretical mean
time_mean_RZ = Calculate_time_mean(Polar_RZ_delayed, num_realization,
total_samples);

% Plot ensamble mean for Polar RZ
figure;
plot(time, stat_mean_RZ, 'r', 'LineWidth', 2); hold on;
plot(time, theoretical_RZ * ones(1, total_samples), 'b--', 'LineWidth', 2);
title('Mean - Polar RZ');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Ensemble Mean', 'Theoretical Mean');
grid on;
xlim([0, 8]);
ylim([-2, 2]);
% Plot time mean across realizations for Polar RZ
figure;
plot(1:num_realization, time_mean_RZ, 'b', 'LineWidth', 2);
title('Time Mean Across Realizations - Polar RZ');
xlabel('Realization Index');
ylabel('Time Mean Value');
grid on;
ylim([-5, 5]);


%%---------------Theoritical AutoCorrelation---------------------
%theoritical PolarNRZ AutoCorrelation is (A^2) @tau=0 only and zero
everywhere with slope 1/bit duration
theoretical_auto_corr_PolarNRZ =@(tau) (A^2) * exp(-abs(tau) / 7);
%theoritical UniPolarNRZ AutoCorrelation is (A^2/2) @tau=0 only and (A^2/4)
everywhere else
theoretical_auto_corr_UnipolarNRZ = @(tau) (A^2 / 4) + ((A^2 / 4) * exp(-
abs(tau) / 7));

%theoritical PolarRZ AutoCorrelation is (A^2 * 4/7) @tau=0 only and zero
everywhere else
theoretical_auto_corr_PolarRZ = @(tau) (A^2 * (4/7)) * exp(-abs(tau) / 7);

tau_values = 1:700; %% As they are 700 bits

%%Applying the tau values if the theoritical formulas of the line codes
PolarNRZ_Theoritical = arrayfun(theoretical_auto_corr_PolarNRZ, tau_values);
UnipolarNRZ_Theoritical = arrayfun(theoretical_auto_corr_UnipolarNRZ,
tau_values);
PolarRZ_Theoritical = arrayfun(theoretical_auto_corr_PolarRZ, tau_values);

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



%%------------------Auto Correlation simulation @ multiple values of tau---
----------------------
%% 1.For Polar NRZ
pnrz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Polar_NRZ_delayed,"Polar NRZ Auto Correlation @
tau = [0:699]");
plot_sim_and_theor(pnrz_auto_corr_multiple_tau,PolarNRZ_Theoritical,"Polar
```

```
NRZ AutoCorrelation","Theoretical Polar NRZ AutoCorrelation","PolarNRZ
AutoCorr. Simulated vs Theoritical");
%---------------------

%% 2.For Polar RZ
prz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Polar_RZ_delayed,"Polar RZ Auto Correlation @ tau
= [0:699]");
plot_sim_and_theor(prz_auto_corr_multiple_tau,PolarRZ_Theoritical,"Polar RZ
AutoCorrelation","Theoretical Polar RZ AutoCorrelation","Polar RZ AutoCorr.
Simulated vs Theoritical");
%---------------------

%% 3.For UniPolar NRZ
upnrz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Uni_Polar_delayed,"UniPolar NRZ Auto Correlation
@ tau = [0:699]");
plot_sim_and_theor(upnrz_auto_corr_multiple_tau,UnipolarNRZ_Theoritical,"Uni
polar NRZ AutoCorr","Theoretical Unipolar NRZ AutoCorr","UnipolarNRZ
AutoCorr. Simulated vs Theoritical");

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



%%-------------------Auto Correlation of Polar NRZ-------------------------
 %% 1.Auto Correlation @ tau = 0 for each RV Sample
polar_nrz_corr_zero_shift=autocorr_func_zero_tau(Polar_NRZ_delayed,"Polar
NRZ Auto Correlation @ tau = 0");
%% 2.Auto Correlation @ tau = 1 between each two RV Samples
pnrz_auto_corr_unity_tau=auto_corr_unity_tau(Polar_NRZ_delayed,"Polar NRZ
Auto Correlation @ tau = 1");

%%-------------------Auto Correlation of Polar RZ-------------------------
 %% 1.Auto Correlation @ tau = 0 for each RV Sample
polar_rz_corr_zero_shift=autocorr_func_zero_tau(Polar_RZ_delayed,"Polar RZ
Auto Correlation @ tau = 0");
%% 2.Auto Correlation @ tau = 1 between each two RV Samples
prz_auto_corr_unity_tau=auto_corr_unity_tau(Polar_RZ_delayed,"Polar RZ Auto
Correlation @ tau = 1");

%%-------------------Auto Correlation of UniPolar NRZ-----------------------
--
 %% 1.Auto Correlation @ tau = 0 for each RV Sample
unipolar_nrz_corr_zero_shift=autocorr_func_zero_tau(Uni_Polar_delayed,"UniPo
lar NRZ Auto Correlation @ tau = 0");
%% 2.Auto Correlation @ tau = 1 between each two RV Samples
upnrz_auto_corr_unity_tau=auto_corr_unity_tau(Uni_Polar_delayed,"UniPolar
NRZ Auto Correlation @ tau = 1");

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




%defenging an 1-D array aa a test values of tau to test the Ergodicity
```

```
ergodic_tau_values = 0:num_realization; % 500 values as they are 500
realizations



%%-------------Ergoticity Polar NRZ------------------------------------------
--
%% 1.isergodic @tau=0 , Self AutoCorr for each Realization
pnrz_isergodic=isergotic(Polar_NRZ_delayed,"Avg. Self-Correlation of all
Polar NRZ Realizations @tau=0");
plot_autocorr(polar_nrz_corr_zero_shift,pnrz_isergodic,"Polar NRZ Auto
Correlation @ tau = 0","Avg. Self-Correlation of all Polar NRZ
Realizations");
%% 2.Ergodic AutoCorrelation @tau=0:500 , Auto Corr. between Realizations &
between Samples RV's
pnrz_ergodic_auto_corr_arr=ergodic_auto_corr(Polar_NRZ_delayed,ergodic_tau_v
alues,"AutoCorrelation across Time Vs AutoCorrelation across ensamble -
Polar NRZ");


%%-------------Ergoticity Polar RZ-------------------------------------------
--
%% 1.isergodic @tau=0 , Self AutoCorr for each Realization
prz_isergodic=isergotic(Polar_RZ_delayed,"Avg. Self-Correlation of all Polar
RZ Realizations @tau=0");
plot_autocorr(polar_rz_corr_zero_shift,prz_isergodic,"Polar RZ Auto
Correlation @ tau = 0","Avg. Self-Correlation of all Polar RZ
Realizations");

%% 2.Ergodic AutoCorrelation @tau=0:500 , Auto Corr. between Realizations &
between Samples RV's
prz_ergodic_auto_corr_arr=ergodic_auto_corr(Polar_RZ_delayed,ergodic_tau_val
ues,"AutoCorrelation across Time Vs AutoCorrelation across ensamble - Polar
RZ");



%%-------------Ergoticity UniPolar NRZ---------------------------------------
--
%% 1.isergodic @tau=0 , Self AutoCorr for each Realization
upnrz_isergodic=isergotic(Uni_Polar_delayed,"Avg. Self-Correlation of all
UniPolar NRZ Realizations @tau=0");
plot_autocorr(unipolar_nrz_corr_zero_shift,upnrz_isergodic,"UniPolar NRZ
Auto Correlation @ tau = 0","Avg. Self-Correlation of all UniPolar NRZ
Realizations");
%%--------------------------------------------------------------------------
%% 2.Ergodic AutoCorrelation @tau=0:500 , Auto Corr. between Realizations &
between Samples RV's
upnrz_ergodic_auto_corr_arr=ergodic_auto_corr(Uni_Polar_delayed,ergodic_tau_
values,"AutoCorrelation across Time Vs AutoCorrelation across ensamble -
UniPolar NRZ");
%%--------------------------------------------------------------------------



%% ------------------ Compute PSD Using Fourier Transform of Autocorrelation
------------------
```

```
% Ensure FFT matches the length of the autocorrelation function
N = length(prz_auto_corr_multiple_tau);

% Define frequency axis (centered at 0)
freq_axis = (-N/2:N/2-1) * (Fs / N);

% PSD is the Fourier Transform (FFT) of the autocorrelation function

PSD_Polar_NRZ = abs(fftshift(fft(pnrz_auto_corr_multiple_tau, N))); % Polar
NRZ
PSD_Uni_Polar = abs(fftshift(fft(upnrz_auto_corr_multiple_tau, N))); %
Unipolar NRZ
PSD_Polar_RZ = abs(fftshift(fft(prz_auto_corr_multiple_tau, N))); % Polar RZ

% Compute Theoretical PSD using the known mathematical formulas for each
signal type

% Theoretical PSD of Polar NRZ
theoretical_PSD_Polar_NRZ = Fs*(A^2 * bitDuration) * (sinc(freq_axis *
bitDuration).^2);

% Theoretical PSD of Unipolar NRZ
theoretical_PSD_Uni_Polar = Fs* (A^2 * bitDuration / 4) * (sinc(freq_axis *
bitDuration).^2);

% Approximate the delta function at f = 0
delta_approx = 10*Fs* (A^2 / 4) * (abs(freq_axis) < (Fs / N));
theoretical_PSD_Uni_Polar = theoretical_PSD_Uni_Polar + delta_approx;

% Theoretical PSD of Polar RZ
theoretical_PSD_Polar_RZ =Fs* (A^2 * bitDuration* 16 / 49) * (sinc(freq_axis
* bitDuration*4 / 7).^2);

% ------------------ Plot Simulated PSD Only ------------------

figure;

% Plot Polar NRZ PSD
subplot(3,1,1);
plot(freq_axis, PSD_Polar_NRZ, 'b', 'LineWidth', 2);
ylim([0 200]);
title('Polar NRZ PSD (Simulated)');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Simulated'); grid on;

% Plot Unipolar NRZ PSD
subplot(3,1,2);
plot(freq_axis, PSD_Uni_Polar, 'b', 'LineWidth', 2);
ylim([0 100]);
title('Unipolar NRZ PSD (Simulated)');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Simulated'); grid on;

% Plot Polar RZ PSD
subplot(3,1,3);
plot(freq_axis, PSD_Polar_RZ, 'b', 'LineWidth', 2);
```

```matlab
ylim([0 50]);
title('Polar RZ PSD (Simulated)');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Simulated'); grid on;

% ------------------ Plot Computed vs Theoretical PSD ------------------

figure;

% Plot Polar NRZ PSD
subplot(3,1,1);
plot(freq_axis, PSD_Polar_NRZ, 'b', 'LineWidth', 2); hold on;
plot(freq_axis, theoretical_PSD_Polar_NRZ, 'r--', 'LineWidth', 2);
ylim([0 200]);
title('Polar NRZ PSD');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Computed', 'Theoretical'); grid on;

% Plot Unipolar NRZ PSD
subplot(3,1,2);

plot(freq_axis, PSD_Uni_Polar, 'b', 'LineWidth', 2); hold on;
plot(freq_axis, theoretical_PSD_Uni_Polar, 'r--', 'LineWidth', 2);
ylim([0 100]);
title('Unipolar NRZ PSD');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Computed', 'Theoretical'); grid on;

% Plot Polar RZ PSD
subplot(3,1,3);
plot(freq_axis, PSD_Polar_RZ, 'b', 'LineWidth', 2); hold on;
plot(freq_axis, theoretical_PSD_Polar_RZ, 'r--', 'LineWidth', 2);
ylim([0 50]);
title('Polar RZ PSD');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Computed', 'Theoretical'); grid on;

%% ------------------ Compute Theoretical Bandwidth ------------------
B_theoretical_Polar_NRZ = 1 / bitDuration;  % First null for Polar NRZ
B_theoretical_Uni_Polar = 1 / bitDuration;  % First null for Unipolar NRZ
B_theoretical_Polar_RZ  = 7 /(4*  bitDuration);  % First null for Polar RZ

%% ------------------ Compute Simulated Bandwidth Till First Null ----------
--------

% Define a strict threshold close to zero for polar_NRZ (0.1m% of max PSD)
threshold_polar_NRZ = max(PSD_Polar_NRZ) * 1e-6;

% Find the first null (first zero-crossing or minimum point)
null_index_NRZ = find(PSD_Polar_NRZ(ceil(N/2):end) < threshold_polar_NRZ, 1,
'first') + ceil(N/2) - 1;
B_simulated_Polar_NRZ = abs(freq_axis(null_index_NRZ));

% Define a strict threshold close to zero for uni_polar_NRZ (0.1m% of max
PSD)
threshold_uni_polar_NRZ = max(PSD_Uni_Polar) * 1e-6;
```

```
null_index_Uni = find(PSD_Uni_Polar(ceil(N/2):end) <
threshold_uni_polar_NRZ, 1, 'first') + ceil(N/2) - 1;
B_simulated_Uni_Polar = abs(freq_axis(null_index_Uni));


% Define a strict threshold close to zero for polar_RZ (0.1m% of max PSD)
threshold_polar_RZ= max(PSD_Polar_RZ) * 1e-6;

null_index_prz = find(PSD_Polar_RZ(ceil(N/2):end) < threshold_polar_RZ, 1,
'first') + ceil(N/2) - 1;
B_simulated_polar_RZ = abs(freq_axis(null_index_prz));




%% ------------------ Display Bandwidth Results ------------------
disp('-----------------------------------------------------');
disp('Bandwidth Till First Null (Hz)');
disp(['Theoretical Polar NRZ: ', num2str(B_theoretical_Polar_NRZ), ' Hz']);
disp(['Simulated Polar NRZ: ', num2str(B_simulated_Polar_NRZ), ' Hz']);
disp('-----------------------------------------------------');
disp(['Theoretical Unipolar NRZ: ', num2str(B_theoretical_Uni_Polar), '
Hz']);
disp(['Simulated Unipolar NRZ: ', num2str(B_simulated_Uni_Polar), ' Hz']);
disp('-----------------------------------------------------');
disp(['Theoretical Polar RZ: ', num2str(B_theoretical_Polar_RZ), ' Hz']);
disp(['Simulated Polar NRZ: ', num2str(B_simulated_polar_RZ), ' Hz']);

disp('-----------------------------------------------------');
%%----------------------------------------------------------------------


%% Function Performing Self Auto Corr. for all Samples RV's (All Coloumns)
function autocorr_RVs = autocorr_func_zero_tau(matrix, plot_title)
    % Initialize output array
    autocorr_RVs = zeros(1, 700);

    % Loop over each RV
    for sample = 1:700
        sample_values = matrix(:, sample); % Extract the required column
        squared_values = sample_values .* sample_values; % Dot product
        autocorr_RVs(sample) = sum(squared_values) / 500;
    end

    % Plot the results
    figure;
    plot(1:700, autocorr_RVs, 'b-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude"); % Use the passed ylabel argument
    title(plot_title);
    grid on;
end

%% Function Performing Auto Corr. between each consecutive (tau = 1) Samples
RV's (All Coloumns)
function dot_products = auto_corr_unity_tau(X, plot_title)
    [~, cols] = size(X); %storing the coloumns in a 2d array
```

```
    dot_products = zeros(1, cols - 1); % Initialize output array

    % Compute dot product for each consecutive column pair
    for i = 1:cols-1
        dot_products(i) = dot(X(:, i), X(:, i+1)) ;
        %the auto-correlation value is normalized to be within [0:1]
    end
    dot_products = dot_products/500;
    % Plot the results
    figure;
    plot(1:length(dot_products), dot_products, 'b-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title(plot_title);
    grid on;
end

%% Function Performing Auto Corr. between Samples RV's (All Coloumns)
starting from t1 or t2
function autocorr_from_t1_or_t2 = autocorr_func_multiple_tau(matrix,
plot_title)

    autocorr_t1 = zeros(1, 700);
    autocorr_t2 = zeros(1, 700);


    for sample = 1:2
        autocorr_current = zeros(1, 700);
        for tau = 0:699
            sample_values1 = matrix(:, sample); % Extract the reference
column
            if (tau == 699 && sample == 2) % Special case for circular shift
                sample_values2 = matrix(:, sample-1);
            else
                sample_values2 = matrix(:, sample+tau);
            end
            squared_values = sample_values1 .* sample_values2;
            autocorr_current(tau+1) = sum(squared_values) / 500;
        end

        if sample == 1
            autocorr_t1 = autocorr_current;
        else
            autocorr_t2 = autocorr_current;
        end
    end

    % Flip and mirror for plotting
    autocorr_flipped(1, :) = [fliplr(autocorr_t1), autocorr_t1];
    autocorr_flipped(2, :) = [fliplr(autocorr_t2), autocorr_t2];


    figure;
    plot(-699:700, autocorr_flipped(1, :), 'b-', 'LineWidth', 1.5);
    hold on;
```

```
    plot(-699:700, autocorr_flipped(2, :), 'r-', 'LineWidth', 1.5);
    hold off;
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title([plot_title ' - Comparison']);
    legend('Autocorr @ t=t1', 'Autocorr @ t=t2');
    grid on;


    figure;
    plot(-699:700, autocorr_flipped(1, :), 'b-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title([plot_title ' - Starting from t1']);
    grid on;


    figure;
    plot(-699:700, autocorr_flipped(2, :), 'r-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title([plot_title ' - Starting from t2']);
    grid on;

    % Return the avg of the autocorrelation from t1 & t2 for PSD calcuations
    autocorr_from_t1_or_t2 = (autocorr_t1 + autocorr_t2) / 2;
end

%% Function Performing Self Auto Corr. (tau = 0 ) between each Realization
(All Rows)
function ergodic_value = isergotic(matrix, plot_title)
    [rows, cols] = size(matrix);
    row_auto_corr = zeros(1, rows);

    for r = 1:rows
        wavefrom = matrix(r, :); % Extract row
        squaredSum = sum(wavefrom .^ 2); % Squaring each element and get sum
of them
        row_auto_corr(r) = squaredSum / 700; % Averaging the sum of the
squared elements
    end

    ergodic_value = sum(row_auto_corr) / rows; % Normalize by total rows

    % Plot the ergodic value
    figure;
    plot(1:cols, ones(1, cols) * ergodic_value, 'r-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    yMin = floor(0 * 10) / 10; % Round down to nearest 0.1
    yMax = ceil(20 * 10) / 10;  % Round up to nearest 0.1
    ylim([yMin, yMax]);
    yticks(yMin:0.5:yMax); % Set ticks at intervals of 0.1
    title(plot_title);
    grid on;
end
```

```matlab
%% General Function Plotting Auto Corr. values with the values of lags (tau)
function plot_autocorr(autocorr_RVs, ergodic_value, title_autocorr,
title_ergodic)
    cols = length(autocorr_RVs);
    time = 1:cols;

    figure;
    plot(time, autocorr_RVs, 'b-', 'LineWidth', 1.5);
    hold on;

    plot(time, ones(1, cols) * ergodic_value, 'r-', 'LineWidth', 1.5);

    xlabel('Time');
    ylabel('Autocorrelation Amplitude');
    title([title_autocorr, ' vs ', title_ergodic]);
    legend(title_autocorr, title_ergodic);
    grid on;
    hold off;
end



%% Function to Calculate Statistical Mean (Ensemble Mean)
function statistical_mean = calculate_statistical_mean(signal,
num_realization, total_samples)
    Sum_signal = zeros(1, total_samples);
    for t = 1:total_samples
        for i = 1:num_realization
            Sum_signal(t) = Sum_signal(t) + signal(i, t);
        end
    end
    statistical_mean = Sum_signal /  num_realization;
end

%% Function to Calculate Time Mean
function time_mean = Calculate_time_mean(signal, num_realization,
total_samples)
    Sum_time_signal = zeros(num_realization, 1);
    for i = 1:num_realization
        for t = 1:total_samples
            Sum_time_signal(i) = Sum_time_signal(i) + signal(i, t);
        end
    end
    time_mean = Sum_time_signal / total_samples;
end

%% General Function Plotting Any Two Matrices
function plot_sim_and_theor(matrix1, matrix2, title1, title2, title3)
    figure;
    hold on;

    x = -699:700;

    % Flip matrix and concatenate it with itself
    matrix1_flipped = flip(matrix1);
    matrix1_padded = [matrix1_flipped, matrix1];
    matrix2_flipped = flip(matrix2);
```

```
    matrix2_padded = [matrix2_flipped, matrix2];

    plot(x, matrix1_padded, 'b-', 'LineWidth', 1.5);
    plot(x, matrix2_padded, 'r--', 'LineWidth', 1.5);

    legend(title1, title2, 'Location', 'best');
    xlabel('tau');
    ylabel('AutoCorr Amplitude');
    title(title3);
    grid on;
    hold off;
end

%% Function Performing  Auto Corr. between all Samples RV's & All
Realizations by the values of lag [0:500] to test Ergodicity
function ergodic_auto_corr_arr = ergodic_auto_corr(matrix2D, tau_values,
plot_title)
    [rows, cols] = size(matrix2D);
    num_lags = length(tau_values);
    col_corr = zeros(1, num_lags);
    row_corr = zeros(1, num_lags);

    % Column dot product (Samples RV's Auto Corr.)
    for i = 1:num_lags
        tau = tau_values(i);
        if tau < cols
            col1 = matrix2D(:, 1);
            col2 = matrix2D(:, tau + 1);
            col_corr(i) = dot(col1, col2) / rows;
        else
            col_corr(i) = NaN; % If lag exceeds dimensions (500)
        end
    end

    % Row dot product (Realizations Auto Corr.)
    for i = 1:num_lags
        tau = tau_values(i);
        if tau < rows
            row1 = matrix2D(1, :);
            row2 = matrix2D(tau + 1, :);
            row_corr(i) = dot(row1, row2) / cols;
        else
            row_corr(i) = NaN; % If lag exceeds dimensions
        end
    end

    % Plot results
    figure;
    hold on;
    plot(tau_values, col_corr, 'r-', 'LineWidth', 2, 'DisplayName', 'Across
Ensemble Correlation');
    plot(tau_values, row_corr, 'b-', 'LineWidth', 2, 'DisplayName', 'Across
Time Correlation');
    legend;
    title(plot_title);
    xlabel('Lag (Tau)');
    ylabel('Normalized Dot Product');
```

```
    grid on;
    hold off;

    % Return the correlation results
    ergodic_auto_corr_arr = [col_corr; row_corr];
endclc; clear; close all;
clc; clear; close all;
%define basic parameters
A = 4;
num_realization = 500;
num_bits = 100;
samples_per_bit = 7;
bitDuration = 70e-3;
Fs = 1 / (bitDuration / samples_per_bit);
total_samples = num_bits * samples_per_bit;
% Generate random binary data
Data = randi([0, 1], num_realization, num_bits);
% Generate random delay in the range of the bit samples
delay_samples = randi([1, 7], num_realization, 1);
% Time axis for plotting
time = (0:total_samples-1) / Fs;

%% 1. Polar NRZ (0 -> -A, 1 -> A)
%convert data to polar_nrz
Polar_NRZ = (2 * Data - 1) * A;
% Expand each bit by repeating its value 7 times(sampling)
Polar_NRZ_out = zeros(num_realization, total_samples);
for i = 1:num_realization
    Polar_NRZ_reshaped = repmat(Polar_NRZ(i, :), samples_per_bit, 1);
    Polar_NRZ_out(i, :) = reshape(Polar_NRZ_reshaped, 1, []);
end
% Apply delay to Polar NRZ using circular shift
Polar_NRZ_delayed = zeros(size(Polar_NRZ_out));
for i = 1:num_realization
    Polar_NRZ_delayed(i, :) = circshift(Polar_NRZ_out(i, :),
delay_samples(i));
end
% Plotting random realizations  against time
figure;
for i = 1:3
    subplot(3,1,i);
    plot(time, Polar_NRZ_delayed(i, :), 'b');
    title(['Polar NRZ - Realization ', num2str(i)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
    grid on;
    xlim([0, 8]);
    ylim([-5, 5]);
end

% Compute means of PNRZ
stat_mean_NRZ = calculate_statistical_mean(Polar_NRZ_delayed,
num_realization, total_samples);
theoretical_NRZ = 0; % Theoretical mean
time_mean_NRZ = Calculate_time_mean(Polar_NRZ_delayed, num_realization,
total_samples);
```

```
% Plot ensamble mean for Polar NRZ
figure;
plot(time, stat_mean_NRZ, 'r', 'LineWidth', 2); hold on;
plot(time, theoretical_NRZ * ones(1, total_samples), 'b--', 'LineWidth', 2);
title('Mean - Polar NRZ');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Ensemble Mean', 'Theoretical Mean');
grid on;
xlim([0, 8]);
ylim([-2, 2]);
% Plot time mean across realizations for Polar NRZ
figure;
plot(1:num_realization, time_mean_NRZ, 'b', 'LineWidth', 2);
title('Time Mean Across Realizations - Polar NRZ');
xlabel('Realization Index');
ylabel('Time Mean Value');
grid on;
ylim([-5, 5]);

%% 2. Unipolar(0 -> 0, 1 -> A)
% Convert data to Uni-Polar
Uni_Polar = Data * A;
% Expand each bit by repeating its value 7 times
Uni_Polar_out = zeros(num_realization, total_samples);
for i = 1:num_realization
    Uni_Polar_reshaped = repmat(Uni_Polar(i, :), samples_per_bit, 1);
    Uni_Polar_out(i, :) = reshape(Uni_Polar_reshaped, 1, []);
end
% Apply delay to Uni_Polar using circular shift
Uni_Polar_delayed = zeros(size(Uni_Polar_out));
for i = 1:num_realization
    Uni_Polar_delayed(i, :) = circshift(Uni_Polar_out(i, :),
delay_samples(i));
end
% Plotting random realizations  against time
figure;
for i = 1:3
    subplot(3,1,i);
    plot(time, Uni_Polar_delayed(i, :), 'g');
    title(['Uni-Polar - Realization ', num2str(i)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
    grid on;
    xlim([0, 8]);
     ylim([-5, 5]);
end
% Compute means of UPNRZ
stat_mean_Unipolar = calculate_statistical_mean(Uni_Polar_delayed,
num_realization, total_samples);
theoretical_Unipolar = 0.5 * A; % Theoretical mean
time_mean_Unipolar = Calculate_time_mean(Uni_Polar_delayed, num_realization,
total_samples);

% Plot ensamble mean for Unipolar
figure;
plot(time, stat_mean_Unipolar, 'r', 'LineWidth', 2); hold on;
```

```
plot(time, theoretical_Unipolar * ones(1, total_samples), 'b--',
'LineWidth', 2);
title('Mean - Unipolar');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Ensemble Mean', 'Theoretical Mean');
grid on;
xlim([0, 8]);
ylim([0, 4]);
% Plot time mean across realizations for Unipolar
figure;
plot(1:num_realization, time_mean_Unipolar, 'b', 'LineWidth', 2);
title('Time Mean Across Realizations - Unipolar');
xlabel('Realization Index');
ylabel('Time Mean Value');
grid on;
ylim([0, 4]);

%% 3. Polar RZ
% Convert to Polar RZ (0 -> -A, 1 -> A for 4 samples, then 0 for 3 samples)
Polar_RZ = (2 * Data - 1) * A;

Polar_RZ_out = zeros(num_realization, total_samples);
% Iterate over realizations
for i = 1:num_realization
    % Expand each bit to cover the required samples
    Polar_RZ_expanded = repmat(Polar_RZ(i, :), samples_per_bit, 1);

    % Set last 3 samples of each bit to zero (Polar RZ format)
    for j = 1:num_bits
        Polar_RZ_expanded(5:end, j) = 0;  % First 4 samples remain, last 3
go to zero
    end

    % Flatten to a single row
    Polar_RZ_out(i, :) = reshape(Polar_RZ_expanded, 1, []);
end


% Apply delay to Polar_RZ using circular shift
Polar_RZ_delayed = zeros(size(Polar_RZ_out));
for i = 1:num_realization
    Polar_RZ_delayed(i, :) = circshift(Polar_RZ_out(i, :),
delay_samples(i));
end
%plotting
figure;
for i = 1:3
    subplot(3,1,i);
    plot(time, Polar_RZ_delayed(i, :), 'm');
    title(['Polar RZ - Realization ', num2str(i)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
    grid on;
    xlim([0, 8]);
     ylim([-5, 5]);
end
```

```
% Compute means of PRZ
stat_mean_RZ = calculate_statistical_mean(Polar_RZ_delayed, num_realization,
total_samples);
theoretical_RZ = 0; % Theoretical mean
time_mean_RZ = Calculate_time_mean(Polar_RZ_delayed, num_realization,
total_samples);

% Plot ensamble mean for Polar RZ
figure;
plot(time, stat_mean_RZ, 'r', 'LineWidth', 2); hold on;
plot(time, theoretical_RZ * ones(1, total_samples), 'b--', 'LineWidth', 2);
title('Mean - Polar RZ');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Ensemble Mean', 'Theoretical Mean');
grid on;
xlim([0, 8]);
ylim([-2, 2]);
% Plot time mean across realizations for Polar RZ
figure;
plot(1:num_realization, time_mean_RZ, 'b', 'LineWidth', 2);
title('Time Mean Across Realizations - Polar RZ');
xlabel('Realization Index');
ylabel('Time Mean Value');
grid on;
ylim([-5, 5]);


%%---------------Theoritical AutoCorrelation---------------------
%theoritical PolarNRZ AutoCorrelation is (A^2) @tau=0 only and zero
everywhere with slope 1/bit duration
theoretical_auto_corr_PolarNRZ =@(tau) (A^2) * exp(-abs(tau) / 7);
%theoritical UniPolarNRZ AutoCorrelation is (A^2/2) @tau=0 only and (A^2/4)
everywhere else
theoretical_auto_corr_UnipolarNRZ = @(tau) (A^2 / 4) + ((A^2 / 4) * exp(-
abs(tau) / 7));

%theoritical PolarRZ AutoCorrelation is (A^2 * 4/7) @tau=0 only and zero
everywhere else
theoretical_auto_corr_PolarRZ = @(tau) (A^2 * (4/7)) * exp(-abs(tau) / 7);

tau_values = 1:700; %% As they are 700 bits

%%Applying the tau values if the theoritical formulas of the line codes
PolarNRZ_Theoritical = arrayfun(theoretical_auto_corr_PolarNRZ, tau_values);
UnipolarNRZ_Theoritical = arrayfun(theoretical_auto_corr_UnipolarNRZ,
tau_values);
PolarRZ_Theoritical = arrayfun(theoretical_auto_corr_PolarRZ, tau_values);

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



%%------------------Auto Correlation simulation @ multiple values of tau---
----------------------
%% 1.For Polar NRZ
pnrz_auto_corr_multiple_tau =
```

```
autocorr_func_multiple_tau(Polar_NRZ_delayed,"Polar NRZ Auto Correlation @
tau = [0:699]");
plot_sim_and_theor(pnrz_auto_corr_multiple_tau,PolarNRZ_Theoritical,"Polar
NRZ AutoCorrelation","Theoretical Polar NRZ AutoCorrelation","PolarNRZ
AutoCorr. Simulated vs Theoritical");
%---------------------

%% 2.For Polar RZ
prz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Polar_RZ_delayed,"Polar RZ Auto Correlation @ tau
= [0:699]");
plot_sim_and_theor(prz_auto_corr_multiple_tau,PolarRZ_Theoritical,"Polar RZ
AutoCorrelation","Theoretical Polar RZ AutoCorrelation","Polar RZ AutoCorr.
Simulated vs Theoritical");
%---------------------

%% 3.For UniPolar NRZ
upnrz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Uni_Polar_delayed,"UniPolar NRZ Auto Correlation
@ tau = [0:699]");
plot_sim_and_theor(upnrz_auto_corr_multiple_tau,UnipolarNRZ_Theoritical,"Uni
polar NRZ AutoCorr","Theoretical Unipolar NRZ AutoCorr","UnipolarNRZ
AutoCorr. Simulated vs Theoritical");

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


%%-------------------Auto Correlation of Polar NRZ-------------------------
 %% 1.Auto Correlation @ tau = 0 for each RV Sample
polar_nrz_corr_zero_shift=autocorr_func_zero_tau(Polar_NRZ_delayed,"Polar
NRZ Auto Correlation @ tau = 0");
%% 2.Auto Correlation @ tau = 1 between each two RV Samples
pnrz_auto_corr_unity_tau=auto_corr_unity_tau(Polar_NRZ_delayed,"Polar NRZ
Auto Correlation @ tau = 1");

%%-------------------Auto Correlation of Polar RZ-------------------------
 %% 1.Auto Correlation @ tau = 0 for each RV Sample
polar_rz_corr_zero_shift=autocorr_func_zero_tau(Polar_RZ_delayed,"Polar RZ
Auto Correlation @ tau = 0");
%% 2.Auto Correlation @ tau = 1 between each two RV Samples
prz_auto_corr_unity_tau=auto_corr_unity_tau(Polar_RZ_delayed,"Polar RZ Auto
Correlation @ tau = 1");

%%-------------------Auto Correlation of UniPolar NRZ-----------------------
--
 %% 1.Auto Correlation @ tau = 0 for each RV Sample
unipolar_nrz_corr_zero_shift=autocorr_func_zero_tau(Uni_Polar_delayed,"UniPo
lar NRZ Auto Correlation @ tau = 0");
%% 2.Auto Correlation @ tau = 1 between each two RV Samples
upnrz_auto_corr_unity_tau=auto_corr_unity_tau(Uni_Polar_delayed,"UniPolar
NRZ Auto Correlation @ tau = 1");

%%---------------------------------------------------------------------------
--------------------------------------------
```

```
%definging an 1-D array aa a test values of tau to test the Ergodicity
ergodic_tau_values = 0:num_realization; % 500 values as they are 500
realizations



%%-------------Ergoticity Polar NRZ-----------------------------------------
--
%% 1.isergodic @tau=0 , Self AutoCorr for each Realization
pnrz_isergodic=isergotic(Polar_NRZ_delayed,"Avg. Self-Correlation of all
Polar NRZ Realizations @tau=0");
plot_autocorr(polar_nrz_corr_zero_shift,pnrz_isergodic,"Polar NRZ Auto
Correlation @ tau = 0","Avg. Self-Correlation of all Polar NRZ
Realizations");
%% 2.Ergodic AutoCorrelation @tau=0:500 , Auto Corr. between Realizations &
between Samples RV's
pnrz_ergodic_auto_corr_arr=ergodic_auto_corr(Polar_NRZ_delayed,ergodic_tau_v
alues,"AutoCorrelation across Time Vs AutoCorrelation across ensamble -
Polar NRZ");


%%-------------Ergoticity Polar RZ------------------------------------------
--
%% 1.isergodic @tau=0 , Self AutoCorr for each Realization
prz_isergodic=isergotic(Polar_RZ_delayed,"Avg. Self-Correlation of all Polar
RZ Realizations @tau=0");
plot_autocorr(polar_rz_corr_zero_shift,prz_isergodic,"Polar RZ Auto
Correlation @ tau = 0","Avg. Self-Correlation of all Polar RZ
Realizations");

%% 2.Ergodic AutoCorrelation @tau=0:500 , Auto Corr. between Realizations &
between Samples RV's
prz_ergodic_auto_corr_arr=ergodic_auto_corr(Polar_RZ_delayed,ergodic_tau_val
ues,"AutoCorrelation across Time Vs AutoCorrelation across ensamble - Polar
RZ");



%%-------------Ergoticity UniPolar NRZ--------------------------------------
--
%% 1.isergodic @tau=0 , Self AutoCorr for each Realization
upnrz_isergodic=isergotic(Uni_Polar_delayed,"Avg. Self-Correlation of all
UniPolar NRZ Realizations @tau=0");
plot_autocorr(unipolar_nrz_corr_zero_shift,upnrz_isergodic,"UniPolar NRZ
Auto Correlation @ tau = 0","Avg. Self-Correlation of all UniPolar NRZ
Realizations");
%%---------------------------------------------------------------------------
%% 2.Ergodic AutoCorrelation @tau=0:500 , Auto Corr. between Realizations &
between Samples RV's
upnrz_ergodic_auto_corr_arr=ergodic_auto_corr(Uni_Polar_delayed,ergodic_tau_
values,"AutoCorrelation across Time Vs AutoCorrelation across ensamble -
UniPolar NRZ");
%%---------------------------------------------------------------------------
```

```matlab
%% ------------------ Compute PSD Using Fourier Transform of Autocorrelation
------------------

% Ensure FFT matches the length of the autocorrelation function
N = length(prz_auto_corr_multiple_tau);

% Define frequency axis (centered at 0)
freq_axis = (-N/2:N/2-1) * (Fs / N);

% PSD is the Fourier Transform (FFT) of the autocorrelation function

PSD_Polar_NRZ = abs(fftshift(fft(pnrz_auto_corr_multiple_tau, N))); % Polar
NRZ
PSD_Uni_Polar = abs(fftshift(fft(upnrz_auto_corr_multiple_tau, N))); %
Unipolar NRZ
PSD_Polar_RZ = abs(fftshift(fft(prz_auto_corr_multiple_tau, N))); % Polar RZ

% Compute Theoretical PSD using the known mathematical formulas for each
signal type

% Theoretical PSD of Polar NRZ
theoretical_PSD_Polar_NRZ = Fs*(A^2 * bitDuration) * (sinc(freq_axis *
bitDuration).^2);

% Theoretical PSD of Unipolar NRZ
theoretical_PSD_Uni_Polar = Fs* (A^2 * bitDuration / 4) * (sinc(freq_axis *
bitDuration).^2);

% Approximate the delta function at f = 0
delta_approx = 10*Fs* (A^2 / 4) * (abs(freq_axis) < (Fs / N));
theoretical_PSD_Uni_Polar = theoretical_PSD_Uni_Polar + delta_approx;

% Theoretical PSD of Polar RZ
theoretical_PSD_Polar_RZ =Fs* (A^2 * bitDuration* 16 / 49) * (sinc(freq_axis
* bitDuration*4 / 7).^2);

% ------------------ Plot Simulated PSD Only ------------------

figure;

% Plot Polar NRZ PSD
subplot(3,1,1);
plot(freq_axis, PSD_Polar_NRZ, 'b', 'LineWidth', 2);
ylim([0 200]);
title('Polar NRZ PSD (Simulated)');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Simulated'); grid on;

% Plot Unipolar NRZ PSD
subplot(3,1,2);
plot(freq_axis, PSD_Uni_Polar, 'b', 'LineWidth', 2);
ylim([0 100]);
title('Unipolar NRZ PSD (Simulated)');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Simulated'); grid on;
```

```matlab
% Plot Polar RZ PSD
subplot(3,1,3);
plot(freq_axis, PSD_Polar_RZ, 'b', 'LineWidth', 2);
ylim([0 50]);
title('Polar RZ PSD (Simulated)');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Simulated'); grid on;

% ------------------ Plot Computed vs Theoretical PSD ------------------

figure;

% Plot Polar NRZ PSD
subplot(3,1,1);
plot(freq_axis, PSD_Polar_NRZ, 'b', 'LineWidth', 2); hold on;
plot(freq_axis, theoretical_PSD_Polar_NRZ, 'r--', 'LineWidth', 2);
ylim([0 200]);
title('Polar NRZ PSD');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Computed', 'Theoretical'); grid on;

% Plot Unipolar NRZ PSD
subplot(3,1,2);

plot(freq_axis, PSD_Uni_Polar, 'b', 'LineWidth', 2); hold on;
plot(freq_axis, theoretical_PSD_Uni_Polar, 'r--', 'LineWidth', 2);
ylim([0 100]);
title('Unipolar NRZ PSD');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Computed', 'Theoretical'); grid on;

% Plot Polar RZ PSD
subplot(3,1,3);
plot(freq_axis, PSD_Polar_RZ, 'b', 'LineWidth', 2); hold on;
plot(freq_axis, theoretical_PSD_Polar_RZ, 'r--', 'LineWidth', 2);
ylim([0 50]);
title('Polar RZ PSD');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Computed', 'Theoretical'); grid on;

%% ------------------ Compute Theoretical Bandwidth ------------------
B_theoretical_Polar_NRZ = 1 / bitDuration;  % First null for Polar NRZ
B_theoretical_Uni_Polar = 1 / bitDuration;  % First null for Unipolar NRZ
B_theoretical_Polar_RZ  = 7 /(4*  bitDuration);  % First null for Polar RZ

%% ------------------ Compute Simulated Bandwidth Till First Null ----------
--------

% Define a strict threshold close to zero for polar_NRZ (0.1m% of max PSD)
threshold_polar_NRZ = max(PSD_Polar_NRZ) * 1e-6;

% Find the first null (first zero-crossing or minimum point)
null_index_NRZ = find(PSD_Polar_NRZ(ceil(N/2):end) < threshold_polar_NRZ, 1,
'first') + ceil(N/2) - 1;
B_simulated_Polar_NRZ = abs(freq_axis(null_index_NRZ));

% Define a strict threshold close to zero for uni_polar_NRZ (0.1m% of max
```

```
PSD)
threshold_uni_polar_NRZ = max(PSD_Uni_Polar) * 1e-6;

null_index_Uni = find(PSD_Uni_Polar(ceil(N/2):end) <
threshold_uni_polar_NRZ, 1, 'first') + ceil(N/2) - 1;
B_simulated_Uni_Polar = abs(freq_axis(null_index_Uni));


% Define a strict threshold close to zero for polar_RZ (0.1m% of max PSD)
threshold_polar_RZ= max(PSD_Polar_RZ) * 1e-6;

null_index_prz = find(PSD_Polar_RZ(ceil(N/2):end) < threshold_polar_RZ, 1,
'first') + ceil(N/2) - 1;
B_simulated_polar_RZ = abs(freq_axis(null_index_prz));




%% ------------------ Display Bandwidth Results ------------------
disp('----------------------------------------------------');
disp('Bandwidth Till First Null (Hz)');
disp(['Theoretical Polar NRZ: ', num2str(B_theoretical_Polar_NRZ), ' Hz']);
disp(['Simulated Polar NRZ: ', num2str(B_simulated_Polar_NRZ), ' Hz']);
disp('----------------------------------------------------');
disp(['Theoretical Unipolar NRZ: ', num2str(B_theoretical_Uni_Polar), '
Hz']);
disp(['Simulated Unipolar NRZ: ', num2str(B_simulated_Uni_Polar), ' Hz']);
disp('----------------------------------------------------');
disp(['Theoretical Polar RZ: ', num2str(B_theoretical_Polar_RZ), ' Hz']);
disp(['Simulated Polar NRZ: ', num2str(B_simulated_polar_RZ), ' Hz']);

disp('----------------------------------------------------');
%%--------------------------------------------------------------------


%% Function Performing Self Auto Corr. for all Samples RV's (All Coloumns)
function autocorr_RVs = autocorr_func_zero_tau(matrix, plot_title)
    % Initialize output array
    autocorr_RVs = zeros(1, 700);

    % Loop over each RV
    for sample = 1:700
        sample_values = matrix(:, sample); % Extract the required column
        squared_values = sample_values .* sample_values; % Dot product
        autocorr_RVs(sample) = sum(squared_values) / 500;
    end

    % Plot the results
    figure;
    plot(1:700, autocorr_RVs, 'b-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude"); % Use the passed ylabel argument
    title(plot_title);
    grid on;
end

%% Function Performing Auto Corr. between each consecutive (tau = 1) Samples
```

```
RV's (All Coloumns)
function dot_products = auto_corr_unity_tau(X, plot_title)
    [~, cols] = size(X); %storing the coloumns in a 2d array


    dot_products = zeros(1, cols - 1); % Initialize output array

    % Compute dot product for each consecutive column pair
    for i = 1:cols-1
        dot_products(i) = dot(X(:, i), X(:, i+1)) ;
        %the auto-correlation value is normalized to be within [0:1]
    end
    dot_products = dot_products/500;
    % Plot the results
    figure;
    plot(1:length(dot_products), dot_products, 'b-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title(plot_title);
    grid on;
end

%% Function Performing Auto Corr. between Samples RV's (All Coloumns)
starting from t1 or t2
function autocorr_from_t1_or_t2 = autocorr_func_multiple_tau(matrix,
plot_title)

    autocorr_t1 = zeros(1, 700);
    autocorr_t2 = zeros(1, 700);


    for sample = 1:2
        autocorr_current = zeros(1, 700);
        for tau = 0:699
            sample_values1 = matrix(:, sample); % Extract the reference
column
            if (tau == 699 && sample == 2) % Special case for circular shift
                sample_values2 = matrix(:, sample-1);
            else
                sample_values2 = matrix(:, sample+tau);
            end
            squared_values = sample_values1 .* sample_values2;
            autocorr_current(tau+1) = sum(squared_values) / 500;
        end

        if sample == 1
            autocorr_t1 = autocorr_current;
        else
            autocorr_t2 = autocorr_current;
        end
    end

    % Flip and mirror for plotting
    autocorr_flipped(1, :) = [fliplr(autocorr_t1), autocorr_t1];
    autocorr_flipped(2, :) = [fliplr(autocorr_t2), autocorr_t2];
```

```
    figure;
    plot(-699:700, autocorr_flipped(1, :), 'b-', 'LineWidth', 1.5);
    hold on;
    plot(-699:700, autocorr_flipped(2, :), 'r-', 'LineWidth', 1.5);
    hold off;
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title([plot_title ' - Comparison']);
    legend('Autocorr @ t=t1', 'Autocorr @ t=t2');
    grid on;


    figure;
    plot(-699:700, autocorr_flipped(1, :), 'b-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title([plot_title ' - Starting from t1']);
    grid on;


    figure;
    plot(-699:700, autocorr_flipped(2, :), 'r-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title([plot_title ' - Starting from t2']);
    grid on;

    % Return the avg of the autocorrelation from t1 & t2 for PSD calcuations
    autocorr_from_t1_or_t2 = (autocorr_t1 + autocorr_t2) / 2;
end

%% Function Performing Self Auto Corr. (tau = 0 ) between each Realization
(All Rows)
function ergodic_value = isergotic(matrix, plot_title)
    [rows, cols] = size(matrix);
    row_auto_corr = zeros(1, rows);

    for r = 1:rows
        wavefrom = matrix(r, :); % Extract row
        squaredSum = sum(wavefrom .^ 2); % Squaring each element and get sum
of them
        row_auto_corr(r) = squaredSum / 700; % Averaging the sum of the
squared elements
    end

    ergodic_value = sum(row_auto_corr) / rows; % Normalize by total rows

    % Plot the ergodic value
    figure;
    plot(1:cols, ones(1, cols) * ergodic_value, 'r-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    yMin = floor(0 * 10) / 10; % Round down to nearest 0.1
    yMax = ceil(20 * 10) / 10;  % Round up to nearest 0.1
    ylim([yMin, yMax]);
    yticks(yMin:0.5:yMax); % Set ticks at intervals of 0.1
    title(plot_title);
```

```
    grid on;
end

%% General Function Plotting Auto Corr. values with the values of lags (tau)
function plot_autocorr(autocorr_RVs, ergodic_value, title_autocorr,
title_ergodic)
    cols = length(autocorr_RVs);
    time = 1:cols;

    figure;
    plot(time, autocorr_RVs, 'b-', 'LineWidth', 1.5);
    hold on;

    plot(time, ones(1, cols) * ergodic_value, 'r-', 'LineWidth', 1.5);

    xlabel('Time');
    ylabel('Autocorrelation Amplitude');
    title([title_autocorr, ' vs ', title_ergodic]);
    legend(title_autocorr, title_ergodic);
    grid on;
    hold off;
end



%% Function to Calculate Statistical Mean (Ensemble Mean)
function statistical_mean = calculate_statistical_mean(signal,
num_realization, total_samples)
    Sum_signal = zeros(1, total_samples);
    for t = 1:total_samples
        for i = 1:num_realization
            Sum_signal(t) = Sum_signal(t) + signal(i, t);
        end
    end
    statistical_mean = Sum_signal /  num_realization;
end

%% Function to Calculate Time Mean
function time_mean = Calculate_time_mean(signal, num_realization,
total_samples)
    Sum_time_signal = zeros(num_realization, 1);
    for i = 1:num_realization
        for t = 1:total_samples
            Sum_time_signal(i) = Sum_time_signal(i) + signal(i, t);
        end
    end
    time_mean = Sum_time_signal / total_samples;
end

%% General Function Plotting Any Two Matrices
function plot_sim_and_theor(matrix1, matrix2, title1, title2, title3)
    figure;
    hold on;

    x = -699:700;

    % Flip matrix and concatenate it with itself
```

```
    matrix1_flipped = flip(matrix1);
    matrix1_padded = [matrix1_flipped, matrix1];
    matrix2_flipped = flip(matrix2);
    matrix2_padded = [matrix2_flipped, matrix2];

    plot(x, matrix1_padded, 'b-', 'LineWidth', 1.5);
    plot(x, matrix2_padded, 'r--', 'LineWidth', 1.5);

    legend(title1, title2, 'Location', 'best');
    xlabel('tau');
    ylabel('AutoCorr Amplitude');
    title(title3);
    grid on;
    hold off;
end

%% Function Performing  Auto Corr. between all Samples RV's & All
Realizations by the values of lag [0:500] to test Ergodicity
function ergodic_auto_corr_arr = ergodic_auto_corr(matrix2D, tau_values,
plot_title)
    [rows, cols] = size(matrix2D);
    num_lags = length(tau_values);
    col_corr = zeros(1, num_lags);
    row_corr = zeros(1, num_lags);

    % Column dot product (Samples RV's Auto Corr.)
    for i = 1:num_lags
        tau = tau_values(i);
        if tau < cols
            col1 = matrix2D(:, 1);
            col2 = matrix2D(:, tau + 1);
            col_corr(i) = dot(col1, col2) / rows;
        else
            col_corr(i) = NaN; % If lag exceeds dimensions (500)
        end
    end

    % Row dot product (Realizations Auto Corr.)
    for i = 1:num_lags
        tau = tau_values(i);
        if tau < rows
            row1 = matrix2D(1, :);
            row2 = matrix2D(tau + 1, :);
            row_corr(i) = dot(row1, row2) / cols;
        else
            row_corr(i) = NaN; % If lag exceeds dimensions
        end
    end

    % Plot results
    figure;
    hold on;
    plot(tau_values, col_corr, 'r-', 'LineWidth', 2, 'DisplayName', 'Across
Ensemble Correlation');
    plot(tau_values, row_corr, 'b-', 'LineWidth', 2, 'DisplayName', 'Across
Time Correlation');
    legend;
```

```
    title(plot_title);
    xlabel('Lag (Tau)');
    ylabel('Normalized Dot Product');
    grid on;
    hold off;

    % Return the correlation results
    ergodic_auto_corr_arr = [col_corr; row_corr];
endclc; clear; close all;
%define basic parameters
A = 4;
num_realization = 500;
num_bits = 100;
samples_per_bit = 7;
bitDuration = 70e-3;
Fs = 1 / (bitDuration / samples_per_bit);
total_samples = num_bits * samples_per_bit;
% Generate random binary data
Data = randi([0, 1], num_realization, num_bits);
% Generate random delay in the range of the bit samples
delay_samples = randi([1, 7], num_realization, 1);
% Time axis for plotting
time = (0:total_samples-1) / Fs;

%% 1. Polar NRZ (0 -> -A, 1 -> A)
%convert data to polar_nrz
Polar_NRZ = (2 * Data - 1) * A;
% Expand each bit by repeating its value 7 times(sampling)
Polar_NRZ_out = zeros(num_realization, total_samples);
for i = 1:num_realization
    Polar_NRZ_reshaped = repmat(Polar_NRZ(i, :), samples_per_bit, 1);
    Polar_NRZ_out(i, :) = reshape(Polar_NRZ_reshaped, 1, []);
end
% Apply delay to Polar NRZ using circular shift
Polar_NRZ_delayed = zeros(size(Polar_NRZ_out));
for i = 1:num_realization
    Polar_NRZ_delayed(i, :) = circshift(Polar_NRZ_out(i, :),
delay_samples(i));
end
% Plotting random realizations  against time
figure;
for i = 1:3
    subplot(3,1,i);
    plot(time, Polar_NRZ_delayed(i, :), 'b');
    title(['Polar NRZ - Realization ', num2str(i)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
    grid on;
    xlim([0, 8]);
    ylim([-5, 5]);
end

% Compute means of PNRZ
stat_mean_NRZ = calculate_statistical_mean(Polar_NRZ_delayed,
num_realization, total_samples);
theoretical_NRZ = 0; % Theoretical mean
time_mean_NRZ = Calculate_time_mean(Polar_NRZ_delayed, num_realization,
```

```
total_samples);

% Plot ensamble mean for Polar NRZ
figure;
plot(time, stat_mean_NRZ, 'r', 'LineWidth', 2); hold on;
plot(time, theoretical_NRZ * ones(1, total_samples), 'b--', 'LineWidth', 2);
title('Mean - Polar NRZ');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Ensemble Mean', 'Theoretical Mean');
grid on;
xlim([0, 8]);
ylim([-2, 2]);
% Plot time mean across realizations for Polar NRZ
figure;
plot(1:num_realization, time_mean_NRZ, 'b', 'LineWidth', 2);
title('Time Mean Across Realizations - Polar NRZ');
xlabel('Realization Index');
ylabel('Time Mean Value');
grid on;
ylim([-5, 5]);

%% 2. Unipolar(0 -> 0, 1 -> A)
% Convert data to Uni-Polar
Uni_Polar = Data * A;
% Expand each bit by repeating its value 7 times
Uni_Polar_out = zeros(num_realization, total_samples);
for i = 1:num_realization
    Uni_Polar_reshaped = repmat(Uni_Polar(i, :), samples_per_bit, 1);
    Uni_Polar_out(i, :) = reshape(Uni_Polar_reshaped, 1, []);
end
% Apply delay to Uni_Polar using circular shift
Uni_Polar_delayed = zeros(size(Uni_Polar_out));
for i = 1:num_realization
    Uni_Polar_delayed(i, :) = circshift(Uni_Polar_out(i, :),
delay_samples(i));
end
% Plotting random realizations  against time
figure;
for i = 1:3
    subplot(3,1,i);
    plot(time, Uni_Polar_delayed(i, :), 'g');
    title(['Uni-Polar - Realization ', num2str(i)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
    grid on;
    xlim([0, 8]);
     ylim([-5, 5]);
end
% Compute means of UPNRZ
stat_mean_Unipolar = calculate_statistical_mean(Uni_Polar_delayed,
num_realization, total_samples);
theoretical_Unipolar = 0.5 * A; % Theoretical mean
time_mean_Unipolar = Calculate_time_mean(Uni_Polar_delayed, num_realization,
total_samples);

% Plot ensamble mean for Unipolar
```

```
figure;
plot(time, stat_mean_Unipolar, 'r', 'LineWidth', 2); hold on;
plot(time, theoretical_Unipolar * ones(1, total_samples), 'b--',
'LineWidth', 2);
title('Mean - Unipolar');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Ensemble Mean', 'Theoretical Mean');
grid on;
xlim([0, 8]);
ylim([0, 4]);
% Plot time mean across realizations for Unipolar
figure;
plot(1:num_realization, time_mean_Unipolar, 'b', 'LineWidth', 2);
title('Time Mean Across Realizations - Unipolar');
xlabel('Realization Index');
ylabel('Time Mean Value');
grid on;
ylim([0, 4]);

%% 3. Polar RZ
% Convert to Polar RZ (0 -> -A, 1 -> A for 4 samples, then 0 for 3 samples)
Polar_RZ = (2 * Data - 1) * A;

Polar_RZ_out = zeros(num_realization, total_samples);
% Iterate over realizations
for i = 1:num_realization
    % Expand each bit to cover the required samples
    Polar_RZ_expanded = repmat(Polar_RZ(i, :), samples_per_bit, 1);

    % Set last 3 samples of each bit to zero (Polar RZ format)
    for j = 1:num_bits
        Polar_RZ_expanded(5:end, j) = 0;  % First 4 samples remain, last 3
go to zero
    end

    % Flatten to a single row
    Polar_RZ_out(i, :) = reshape(Polar_RZ_expanded, 1, []);
end


% Apply delay to Polar_RZ using circular shift
Polar_RZ_delayed = zeros(size(Polar_RZ_out));
for i = 1:num_realization
    Polar_RZ_delayed(i, :) = circshift(Polar_RZ_out(i, :),
delay_samples(i));
end
%plotting
figure;
for i = 1:3
    subplot(3,1,i);
    plot(time, Polar_RZ_delayed(i, :), 'm');
    title(['Polar RZ - Realization ', num2str(i)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
    grid on;
    xlim([0, 8]);
```

```
    ylim([-5, 5]);
end
% Compute means of PRZ
stat_mean_RZ = calculate_statistical_mean(Polar_RZ_delayed, num_realization,
total_samples);
theoretical_RZ = 0; % Theoretical mean
time_mean_RZ = Calculate_time_mean(Polar_RZ_delayed, num_realization,
total_samples);

% Plot ensamble mean for Polar RZ
figure;
plot(time, stat_mean_RZ, 'r', 'LineWidth', 2); hold on;
plot(time, theoretical_RZ * ones(1, total_samples), 'b--', 'LineWidth', 2);
title('Mean - Polar RZ');
xlabel('Time (s)');
ylabel('Amplitude');
legend('Ensemble Mean', 'Theoretical Mean');
grid on;
xlim([0, 8]);
ylim([-2, 2]);
% Plot time mean across realizations for Polar RZ
figure;
plot(1:num_realization, time_mean_RZ, 'b', 'LineWidth', 2);
title('Time Mean Across Realizations - Polar RZ');
xlabel('Realization Index');
ylabel('Time Mean Value');
grid on;
ylim([-5, 5]);


%%---------------Theoritical AutoCorrelation---------------------
%theoritical PolarNRZ AutoCorrelation is (A^2) @tau=0 only and zero
everywhere with slope 1/bit duration
theoretical_auto_corr_PolarNRZ =@(tau) (A^2) * exp(-abs(tau) / 7);
%theoritical UniPolarNRZ AutoCorrelation is (A^2/2) @tau=0 only and (A^2/4)
everywhere else
theoretical_auto_corr_UnipolarNRZ = @(tau) (A^2 / 4) + ((A^2 / 4) * exp(-
abs(tau) / 7));

%theoritical PolarRZ AutoCorrelation is (A^2 * 4/7) @tau=0 only and zero
everywhere else
theoretical_auto_corr_PolarRZ = @(tau) (A^2 * (4/7)) * exp(-abs(tau) / 7);

tau_values = 1:700; %% As they are 700 bits

%%Applying the tau values if the theoritical formulas of the line codes
PolarNRZ_Theoritical = arrayfun(theoretical_auto_corr_PolarNRZ, tau_values);
UnipolarNRZ_Theoritical = arrayfun(theoretical_auto_corr_UnipolarNRZ,
tau_values);
PolarRZ_Theoritical = arrayfun(theoretical_auto_corr_PolarRZ, tau_values);

%%-------------------------------------------------------------
```

```
%%-------------------Auto Correlation simulation @ multiple values of tau---
----------------------
%% 1.For Polar NRZ
pnrz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Polar_NRZ_delayed,"Polar NRZ Auto Correlation @
tau = [0:699]");
plot_sim_and_theor(pnrz_auto_corr_multiple_tau,PolarNRZ_Theoritical,"Polar
NRZ AutoCorrelation","Theoretical Polar NRZ AutoCorrelation","PolarNRZ
AutoCorr. Simulated vs Theoritical");
%---------------------

%% 2.For Polar RZ
prz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Polar_RZ_delayed,"Polar RZ Auto Correlation @ tau
= [0:699]");
plot_sim_and_theor(prz_auto_corr_multiple_tau,PolarRZ_Theoritical,"Polar RZ
AutoCorrelation","Theoretical Polar RZ AutoCorrelation","Polar RZ AutoCorr.
Simulated vs Theoritical");
%---------------------

%% 3.For UniPolar NRZ
upnrz_auto_corr_multiple_tau =
autocorr_func_multiple_tau(Uni_Polar_delayed,"UniPolar NRZ Auto Correlation
@ tau = [0:699]");
plot_sim_and_theor(upnrz_auto_corr_multiple_tau,UnipolarNRZ_Theoritical,"Uni
polar NRZ AutoCorr","Theoretical Unipolar NRZ AutoCorr","UnipolarNRZ
AutoCorr. Simulated vs Theoritical");

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


%%-------------------Auto Correlation of Polar NRZ-------------------------
 %% 1.Auto Correlation @ tau = 0 for each RV Sample
polar_nrz_corr_zero_shift=autocorr_func_zero_tau(Polar_NRZ_delayed,"Polar
NRZ Auto Correlation @ tau = 0");
%% 2.Auto Correlation @ tau = 1 between each two RV Samples
pnrz_auto_corr_unity_tau=auto_corr_unity_tau(Polar_NRZ_delayed,"Polar NRZ
Auto Correlation @ tau = 1");

%%-------------------Auto Correlation of Polar RZ-------------------------
 %% 1.Auto Correlation @ tau = 0 for each RV Sample
polar_rz_corr_zero_shift=autocorr_func_zero_tau(Polar_RZ_delayed,"Polar RZ
Auto Correlation @ tau = 0");
%% 2.Auto Correlation @ tau = 1 between each two RV Samples
prz_auto_corr_unity_tau=auto_corr_unity_tau(Polar_RZ_delayed,"Polar RZ Auto
Correlation @ tau = 1");
```

```
%%-------------------Auto Correlation of UniPolar NRZ-----------------------
--
 %% 1.Auto Correlation @ tau = 0 for each RV Sample
unipolar_nrz_corr_zero_shift=autocorr_func_zero_tau(Uni_Polar_delayed,"UniPo
lar NRZ Auto Correlation @ tau = 0");
%% 2.Auto Correlation @ tau = 1 between each two RV Samples
upnrz_auto_corr_unity_tau=auto_corr_unity_tau(Uni_Polar_delayed,"UniPolar
NRZ Auto Correlation @ tau = 1");

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



%definging an 1-D array aa a test values of tau to test the Ergodicity
ergodic_tau_values = 0:num_realization; % 500 values as they are 500
realizations


%%-------------Ergoticity Polar NRZ----------------------------------------
--
%% 1.isergodic @tau=0 , Self AutoCorr for each Realization
pnrz_isergodic=isergotic(Polar_NRZ_delayed,"Avg. Self-Correlation of all
Polar NRZ Realizations @tau=0");
plot_autocorr(polar_nrz_corr_zero_shift,pnrz_isergodic,"Polar NRZ Auto
Correlation @ tau = 0","Avg. Self-Correlation of all Polar NRZ
Realizations");
%% 2.Ergodic AutoCorrelation @tau=0:500 , Auto Corr. between Realizations &
between Samples RV's
pnrz_ergodic_auto_corr_arr=ergodic_auto_corr(Polar_NRZ_delayed,ergodic_tau_v
alues,"AutoCorrelation across Time Vs AutoCorrelation across ensamble -
Polar NRZ");


%%-------------Ergoticity Polar RZ-----------------------------------------
--
%% 1.isergodic @tau=0 , Self AutoCorr for each Realization
prz_isergodic=isergotic(Polar_RZ_delayed,"Avg. Self-Correlation of all Polar
RZ Realizations @tau=0");
plot_autocorr(polar_rz_corr_zero_shift,prz_isergodic,"Polar RZ Auto
Correlation @ tau = 0","Avg. Self-Correlation of all Polar RZ
Realizations");

%% 2.Ergodic AutoCorrelation @tau=0:500 , Auto Corr. between Realizations &
between Samples RV's
prz_ergodic_auto_corr_arr=ergodic_auto_corr(Polar_RZ_delayed,ergodic_tau_val
ues,"AutoCorrelation across Time Vs AutoCorrelation across ensamble - Polar
RZ");
```

```
%%-------------Ergoticity UniPolar NRZ------------------------------------------
--
%% 1.isergodic @tau=0 , Self AutoCorr for each Realization
upnrz_isergodic=isergotic(Uni_Polar_delayed,"Avg. Self-Correlation of all
UniPolar NRZ Realizations @tau=0");
plot_autocorr(unipolar_nrz_corr_zero_shift,upnrz_isergodic,"UniPolar NRZ
Auto Correlation @ tau = 0","Avg. Self-Correlation of all UniPolar NRZ
Realizations");
%%----------------------------------------------------------------------------
%% 2.Ergodic AutoCorrelation @tau=0:500 , Auto Corr. between Realizations &
between Samples RV's
upnrz_ergodic_auto_corr_arr=ergodic_auto_corr(Uni_Polar_delayed,ergodic_tau_
values,"AutoCorrelation across Time Vs AutoCorrelation across ensamble -
UniPolar NRZ");
%%----------------------------------------------------------------------------



%% ------------------ Compute PSD Using Fourier Transform of Autocorrelation
------------------

% Ensure FFT matches the length of the autocorrelation function
N = length(prz_auto_corr_multiple_tau);

% Define frequency axis (centered at 0)
freq_axis = (-N/2:N/2-1) * (Fs / N);

% PSD is the Fourier Transform (FFT) of the autocorrelation function

PSD_Polar_NRZ = abs(fftshift(fft(pnrz_auto_corr_multiple_tau, N))); % Polar
NRZ
PSD_Uni_Polar = abs(fftshift(fft(upnrz_auto_corr_multiple_tau, N))); %
Unipolar NRZ
PSD_Polar_RZ = abs(fftshift(fft(prz_auto_corr_multiple_tau, N))); % Polar RZ

% Compute Theoretical PSD using the known mathematical formulas for each
signal type

% Theoretical PSD of Polar NRZ
theoretical_PSD_Polar_NRZ = Fs*(A^2 * bitDuration) * (sinc(freq_axis *
bitDuration).^2);

% Theoretical PSD of Unipolar NRZ
theoretical_PSD_Uni_Polar = Fs* (A^2 * bitDuration / 4) * (sinc(freq_axis *
bitDuration).^2);

% Approximate the delta function at f = 0
delta_approx = 10*Fs* (A^2 / 4) * (abs(freq_axis) < (Fs / N));
theoretical_PSD_Uni_Polar = theoretical_PSD_Uni_Polar + delta_approx;

% Theoretical PSD of Polar RZ
theoretical_PSD_Polar_RZ =Fs* (A^2 * bitDuration* 16 / 49) * (sinc(freq_axis
* bitDuration*4 / 7).^2);
```

```
% ------------------ Plot Simulated PSD Only ------------------

figure;

% Plot Polar NRZ PSD
subplot(3,1,1);
plot(freq_axis, PSD_Polar_NRZ, 'b', 'LineWidth', 2);
ylim([0 200]);
title('Polar NRZ PSD (Simulated)');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Simulated'); grid on;

% Plot Unipolar NRZ PSD
subplot(3,1,2);
plot(freq_axis, PSD_Uni_Polar, 'b', 'LineWidth', 2);
ylim([0 100]);
title('Unipolar NRZ PSD (Simulated)');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Simulated'); grid on;

% Plot Polar RZ PSD
subplot(3,1,3);
plot(freq_axis, PSD_Polar_RZ, 'b', 'LineWidth', 2);
ylim([0 50]);
title('Polar RZ PSD (Simulated)');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Simulated'); grid on;

% ------------------ Plot Computed vs Theoretical PSD ------------------

figure;

% Plot Polar NRZ PSD
subplot(3,1,1);
plot(freq_axis, PSD_Polar_NRZ, 'b', 'LineWidth', 2); hold on;
plot(freq_axis, theoretical_PSD_Polar_NRZ, 'r--', 'LineWidth', 2);
ylim([0 200]);
title('Polar NRZ PSD');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Computed', 'Theoretical'); grid on;

% Plot Unipolar NRZ PSD
subplot(3,1,2);

plot(freq_axis, PSD_Uni_Polar, 'b', 'LineWidth', 2); hold on;
plot(freq_axis, theoretical_PSD_Uni_Polar, 'r--', 'LineWidth', 2);
ylim([0 100]);
title('Unipolar NRZ PSD');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Computed', 'Theoretical'); grid on;

% Plot Polar RZ PSD
subplot(3,1,3);
plot(freq_axis, PSD_Polar_RZ, 'b', 'LineWidth', 2); hold on;
plot(freq_axis, theoretical_PSD_Polar_RZ, 'r--', 'LineWidth', 2);
ylim([0 50]);
```

```matlab
title('Polar RZ PSD');
xlabel('Frequency (Hz)'); ylabel('PSD');
legend('Computed', 'Theoretical'); grid on;

%% ------------------ Compute Theoretical Bandwidth ------------------
B_theoretical_Polar_NRZ = 1 / bitDuration;  % First null for Polar NRZ
B_theoretical_Uni_Polar = 1 / bitDuration;  % First null for Unipolar NRZ
B_theoretical_Polar_RZ  = 7 /(4*  bitDuration);  % First null for Polar RZ

%% ------------------ Compute Simulated Bandwidth Till First Null ----------
--------

% Define a strict threshold close to zero for polar_NRZ (0.1m% of max PSD)
threshold_polar_NRZ = max(PSD_Polar_NRZ) * 1e-6;

% Find the first null (first zero-crossing or minimum point)
null_index_NRZ = find(PSD_Polar_NRZ(ceil(N/2):end) < threshold_polar_NRZ, 1,
'first') + ceil(N/2) - 1;
B_simulated_Polar_NRZ = abs(freq_axis(null_index_NRZ));

% Define a strict threshold close to zero for uni_polar_NRZ (0.1m% of max
PSD)
threshold_uni_polar_NRZ = max(PSD_Uni_Polar) * 1e-6;

null_index_Uni = find(PSD_Uni_Polar(ceil(N/2):end) <
threshold_uni_polar_NRZ, 1, 'first') + ceil(N/2) - 1;
B_simulated_Uni_Polar = abs(freq_axis(null_index_Uni));


% Define a strict threshold close to zero for polar_RZ (0.1m% of max PSD)
threshold_polar_RZ= max(PSD_Polar_RZ) * 1e-6;

null_index_prz = find(PSD_Polar_RZ(ceil(N/2):end) < threshold_polar_RZ, 1,
'first') + ceil(N/2) - 1;
B_simulated_polar_RZ = abs(freq_axis(null_index_prz));




%% ------------------ Display Bandwidth Results ------------------
disp('----------------------------------------------------');
disp('Bandwidth Till First Null (Hz)');
disp(['Theoretical Polar NRZ: ', num2str(B_theoretical_Polar_NRZ), ' Hz']);
disp(['Simulated Polar NRZ: ', num2str(B_simulated_Polar_NRZ), ' Hz']);
disp('----------------------------------------------------');
disp(['Theoretical Unipolar NRZ: ', num2str(B_theoretical_Uni_Polar), '
Hz']);
disp(['Simulated Unipolar NRZ: ', num2str(B_simulated_Uni_Polar), ' Hz']);
disp('----------------------------------------------------');
disp(['Theoretical Polar RZ: ', num2str(B_theoretical_Polar_RZ), ' Hz']);
disp(['Simulated Polar NRZ: ', num2str(B_simulated_polar_RZ), ' Hz']);

disp('----------------------------------------------------');
%%-----------------------------------------------------------------
```

```
%% Function Performing Self Auto Corr. for all Samples RV's (All Coloumns)
function autocorr_RVs = autocorr_func_zero_tau(matrix, plot_title)
    % Initialize output array
    autocorr_RVs = zeros(1, 700);

    % Loop over each RV
    for sample = 1:700
        sample_values = matrix(:, sample); % Extract the required column
        squared_values = sample_values .* sample_values; % Dot product
        autocorr_RVs(sample) = sum(squared_values) / 500;
    end

    % Plot the results
    figure;
    plot(1:700, autocorr_RVs, 'b-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude"); % Use the passed ylabel argument
    title(plot_title);
    grid on;
end

%% Function Performing Auto Corr. between each consecutive (tau = 1) Samples
RV's (All Coloumns)
function dot_products = auto_corr_unity_tau(X, plot_title)
    [~, cols] = size(X); %storing the coloumns in a 2d array


    dot_products = zeros(1, cols - 1); % Initialize output array

    % Compute dot product for each consecutive column pair
    for i = 1:cols-1
        dot_products(i) = dot(X(:, i), X(:, i+1)) ;
        %the auto-correlation value is normalized to be within [0:1]
    end
    dot_products = dot_products/500;
    % Plot the results
    figure;
    plot(1:length(dot_products), dot_products, 'b-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title(plot_title);
    grid on;
end

%% Function Performing Auto Corr. between Samples RV's (All Coloumns)
starting from t1 or t2
function autocorr_from_t1_or_t2 = autocorr_func_multiple_tau(matrix,
plot_title)

    autocorr_t1 = zeros(1, 700);
    autocorr_t2 = zeros(1, 700);


    for sample = 1:2
        autocorr_current = zeros(1, 700);
        for tau = 0:699
```

```
            sample_values1 = matrix(:, sample); % Extract the reference
column
            if (tau == 699 && sample == 2) % Special case for circular shift
                sample_values2 = matrix(:, sample-1);
            else
                sample_values2 = matrix(:, sample+tau);
            end
            squared_values = sample_values1 .* sample_values2;
            autocorr_current(tau+1) = sum(squared_values) / 500;
        end

        if sample == 1
            autocorr_t1 = autocorr_current;
        else
            autocorr_t2 = autocorr_current;
        end
    end

    % Flip and mirror for plotting
    autocorr_flipped(1, :) = [fliplr(autocorr_t1), autocorr_t1];
    autocorr_flipped(2, :) = [fliplr(autocorr_t2), autocorr_t2];


    figure;
    plot(-699:700, autocorr_flipped(1, :), 'b-', 'LineWidth', 1.5);
    hold on;
    plot(-699:700, autocorr_flipped(2, :), 'r-', 'LineWidth', 1.5);
    hold off;
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title([plot_title ' - Comparison']);
    legend('Autocorr @ t=t1', 'Autocorr @ t=t2');
    grid on;


    figure;
    plot(-699:700, autocorr_flipped(1, :), 'b-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title([plot_title ' - Starting from t1']);
    grid on;


    figure;
    plot(-699:700, autocorr_flipped(2, :), 'r-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    title([plot_title ' - Starting from t2']);
    grid on;

    % Return the avg of the autocorrelation from t1 & t2 for PSD calcuations
    autocorr_from_t1_or_t2 = (autocorr_t1 + autocorr_t2) / 2;
end
```

```
%% Function Performing Self Auto Corr. (tau = 0 ) between each Realization
(All Rows)
function ergodic_value = isergotic(matrix, plot_title)
    [rows, cols] = size(matrix);
    row_auto_corr = zeros(1, rows);

    for r = 1:rows
        wavefrom = matrix(r, :); % Extract row
        squaredSum = sum(wavefrom .^ 2); % Squaring each element and get sum
of them
        row_auto_corr(r) = squaredSum / 700; % Averaging the sum of the
squared elements
    end

    ergodic_value = sum(row_auto_corr) / rows; % Normalize by total rows

    % Plot the ergodic value
    figure;
    plot(1:cols, ones(1, cols) * ergodic_value, 'r-', 'LineWidth', 1.5);
    xlabel('Time');
    ylabel("Autocorrelation Amplitude");
    yMin = floor(0 * 10) / 10; % Round down to nearest 0.1
    yMax = ceil(20 * 10) / 10;  % Round up to nearest 0.1
    ylim([yMin, yMax]);
    yticks(yMin:0.5:yMax); % Set ticks at intervals of 0.1
    title(plot_title);
    grid on;
end

%% General Function Plotting Auto Corr. values with the values of lags (tau)
function plot_autocorr(autocorr_RVs, ergodic_value, title_autocorr,
title_ergodic)
    cols = length(autocorr_RVs);
    time = 1:cols;

    figure;
    plot(time, autocorr_RVs, 'b-', 'LineWidth', 1.5);
    hold on;

    plot(time, ones(1, cols) * ergodic_value, 'r-', 'LineWidth', 1.5);

    xlabel('Time');
    ylabel('Autocorrelation Amplitude');
    title([title_autocorr, ' vs ', title_ergodic]);
    legend(title_autocorr, title_ergodic);
    grid on;
    hold off;
end
```

```matlab
%% Function to Calculate Statistical Mean (Ensemble Mean)
function statistical_mean = calculate_statistical_mean(signal,
num_realization, total_samples)
    Sum_signal = zeros(1, total_samples);
    for t = 1:total_samples
        for i = 1:num_realization
            Sum_signal(t) = Sum_signal(t) + signal(i, t);
        end
    end
    statistical_mean = Sum_signal /  num_realization;
end

%% Function to Calculate Time Mean
function time_mean = Calculate_time_mean(signal, num_realization,
total_samples)
    Sum_time_signal = zeros(num_realization, 1);
    for i = 1:num_realization
        for t = 1:total_samples
            Sum_time_signal(i) = Sum_time_signal(i) + signal(i, t);
        end
    end
    time_mean = Sum_time_signal / total_samples;
end

%% General Function Plotting Any Two Matrices
function plot_sim_and_theor(matrix1, matrix2, title1, title2, title3)
    figure;
    hold on;

    x = -699:700;

    % Flip matrix and concatenate it with itself
    matrix1_flipped = flip(matrix1);
    matrix1_padded = [matrix1_flipped, matrix1];
    matrix2_flipped = flip(matrix2);
    matrix2_padded = [matrix2_flipped, matrix2];

    plot(x, matrix1_padded, 'b-', 'LineWidth', 1.5);
    plot(x, matrix2_padded, 'r--', 'LineWidth', 1.5);

    legend(title1, title2, 'Location', 'best');
    xlabel('tau');
    ylabel('AutoCorr Amplitude');
    title(title3);
    grid on;
    hold off;
end

%% Function Performing  Auto Corr. between all Samples RV's & All
Realizations by the values of lag [0:500] to test Ergodicity
function ergodic_auto_corr_arr = ergodic_auto_corr(matrix2D, tau_values,
plot_title)
    [rows, cols] = size(matrix2D);
    num_lags = length(tau_values);
    col_corr = zeros(1, num_lags);
```

```
    row_corr = zeros(1, num_lags);

    % Column dot product (Samples RV's Auto Corr.)
    for i = 1:num_lags
        tau = tau_values(i);
        if tau < cols
            col1 = matrix2D(:, 1);
            col2 = matrix2D(:, tau + 1);
            col_corr(i) = dot(col1, col2) / rows;
        else
            col_corr(i) = NaN; % If lag exceeds dimensions (500)
        end
    end

    % Row dot product (Realizations Auto Corr.)
    for i = 1:num_lags
        tau = tau_values(i);
        if tau < rows
            row1 = matrix2D(1, :);
            row2 = matrix2D(tau + 1, :);
            row_corr(i) = dot(row1, row2) / cols;
        else
            row_corr(i) = NaN; % If lag exceeds dimensions
        end
    end

    % Plot results
    figure;
    hold on;
    plot(tau_values, col_corr, 'r-', 'LineWidth', 2, 'DisplayName', 'Across
Ensemble Correlation');
    plot(tau_values, row_corr, 'b-', 'LineWidth', 2, 'DisplayName', 'Across
Time Correlation');
    legend;
    title(plot_title);
    xlabel('Lag (Tau)');
    ylabel('Normalized Dot Product');
    grid on;
    hold off;

    % Return the correlation results
    ergodic_auto_corr_arr = [col_corr; row_corr];
end
```

Reference :

[1] Principles of Baseband Digital Data Transmission - Line Codes and their Power Spectra