Communication Project
Project 1
3rd Year Comm. | Spring 2025
Team 25

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

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

A. Contents

A. **Contents** 2
B. Table of Figures 3
C. Role of Each Member 4
D. Project Description 5
E. Introduction 5
F. Control Flags 5
G. Generation of Data 6
H. polar NRZ ensemble creation 6
I. Uni polar NRZ ensemble creation 7
J. polarRZ ensemble creation 8
K. Random initial time shift 9
L. Getting cell arrays ready to calculate the statistical mean and autocorrelation: 10
M. Questions 12
1. Statistical Mean 12
1.1. Hand Analysis 12
1.2. Code Snippet 12
1.3. Plotting the Statistical Mean: 13
2. Statistical Autocorrelation 14
2.1. Hand Analysis 14
2.2. Code Snippet 15
2.3. Plotting the statistical autocorrelation 16
3. Is the Process Stationary 18
4. The time mean and autocorrelation function for one waveform 19
4.1. Time Mean 19
4.3. Time Auto Correlation 22
4.4. Time Auto Correlation for one wave form: 23
5. Is The Random Process Ergodic? 24
6. the PSD & Bandwidth of the Ensemble 26
6.1. PSD using fft: 26
6.2. Theoritical PSD: 28
N. References: 29
O. Appendix 29

B. Table of Figures

Figure 1 Rx and Tx path 5
Figure 2 ADC Binary Output 6
Figure 3 PolarNRZ Realizations 7
Figure 4 Uni Polar Realizations 8
Figure 5 PolarRZ Realization 9
Figure 6 Realization Shifted 10
Figure 7 Plot of Statistical Mean 13
Figure 8 Statistical Auto Correction plot 16
Figure 9 Statistical Auto Correction plot zoomed 16
Figure 10 autocorrelation at two different times 18
Figure 11 Time Mean for Uni Polar 19
Figure 12 Time Mean for Polar NRZ 20
Figure 13 Time Mean for Polar RZ 20
Figure 14 Time Mean Vs Realization 21
Figure 15 Time Auto Correction plot zoomed 23
Figure 16 Time Auto Correction plot 23
Figure 17 Time Mean vs Statistical 24
Figure 18 Time Auto Correlation Vs Statistical 24
Figure 19 PSD plot of the Ensemble 27

C. Role of Each Member

ROLE	NAME
code the generated waves	Youssef Khaled
compute the realization calculation	Ahmed Mohamed
compute the time calculation	Shahd Hamed
Report and Hand Analysis	Mohamed Ahmed
Report and Hand Analysis	Omar Ahmed

D. Project Description

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

E. Introduction

Software radio is a revolutionary approach that brings the programming code directly to the antenna, minimizing reliance on traditional radio hardware as shown in figure 1.
By doing so, it transforms challenges associated with radio hardware into software- related issues. Unlike conventional radios, where signal processing primarily relies on analog circuitry or a combination of analog and digital chips, software radio operates by having software dictate both the transmitted and received waveforms.
This paradigm shift allows for greater flexibility and adaptability in radio systems, as they can be easily reconfigured and optimized through software updates, rather than hardware modifications.

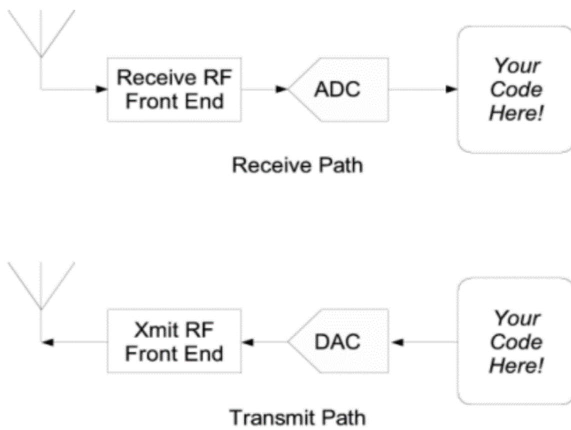

Figure 1 Rx and Tx path

F. Control Flags

Flag	Value	Description
A	4	Amplitude of line code
N_realizations	500	Number of waveforms (ensemble size)
num_bits	101	Bits per waveform and one extra bit for shifting
bit_duration	70e-3	Duration of each bit
dac_interval	10e-3	DAC update interval

G. Generation of Data

```
Data = randi([0, 1], 1, num_bits, 'int8');  % Random bit sequence
```

Using the function: "**Randi**" to generate random binary data of size 500x101[3]
(500 waveforms each with 101 bits). This data represents the binary bits that need to be encoded.

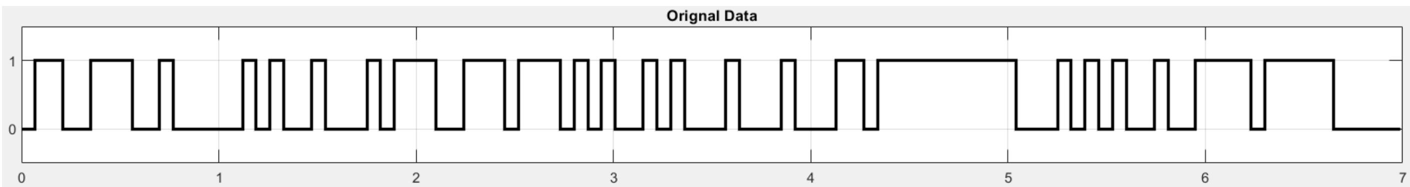

Figure 2 ADC Binary Output

For the line codes we will use this function:

```
function [Unipolar, PolarNRZ, PolarRZ] = generate_linecodes(Data, A, samples_per_bit)...
```

H. polar NRZ ensemble creation

```
% Polar NRZ: 0 → -A, 1 → +A
PolarNRZ = int8((2 * Data - 1) * A);
PolarNRZ = repelem(PolarNRZ, samples_per_bit);
```

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

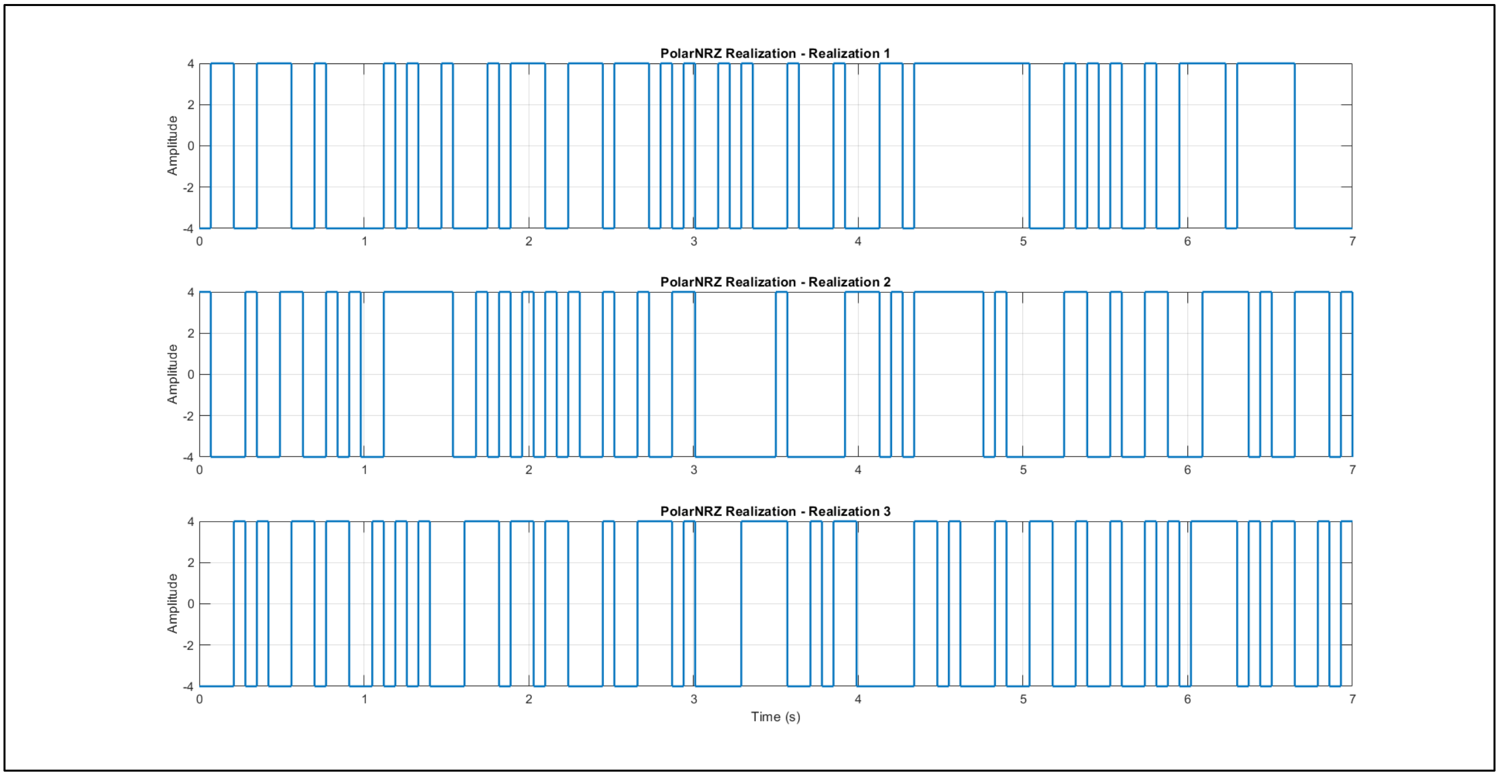

Figure 3 PolarNRZ Realizations

I. Uni polar NRZ ensemble creation

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

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

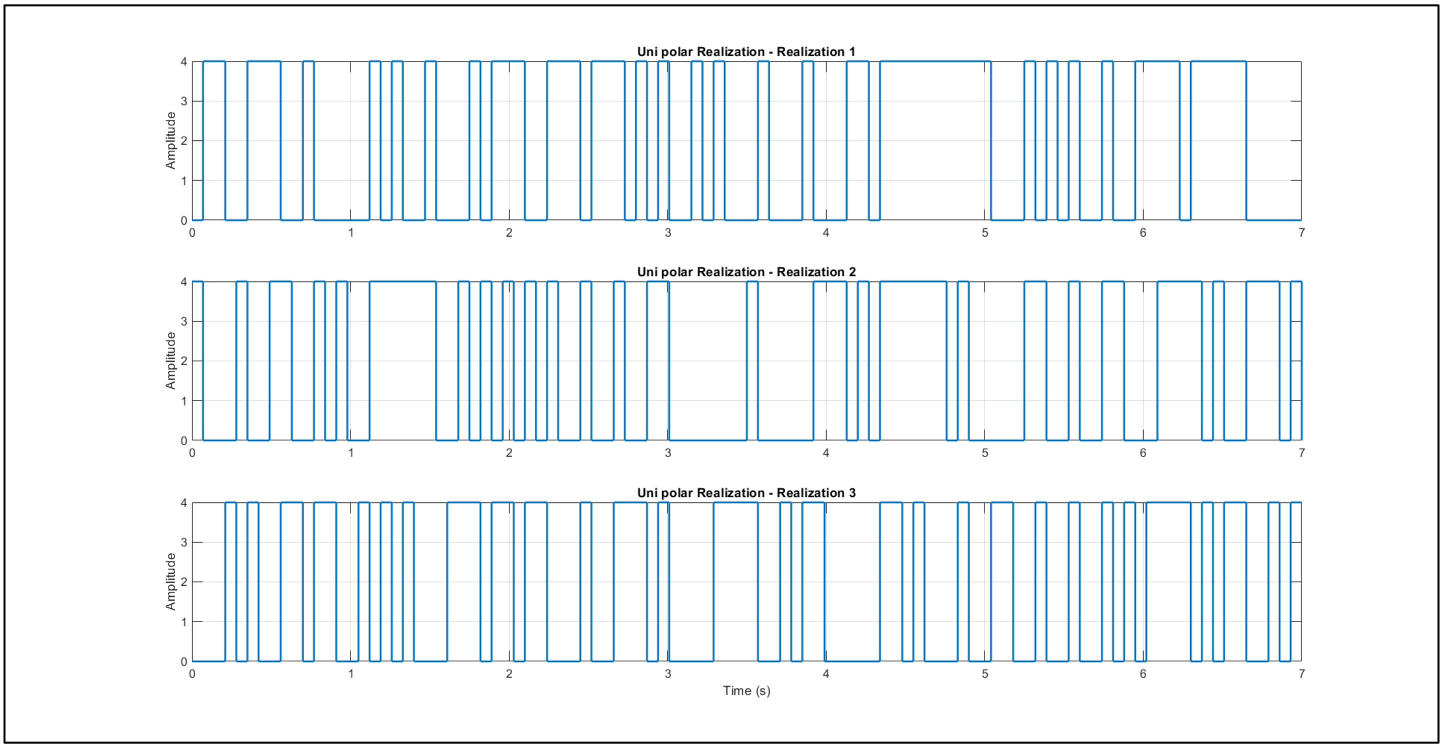

Figure 4 Uni Polar Realizations

J. polarRZ ensemble creation

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

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

- The data consists of 0s and 1s. We first convert these values to amplitudes: `0` → **`-A`, `1`** → **`+A`** (this is the standard **Polar NRZ** encoding).
- Then, we utilized the `repelem` function to repeat each amplitude value `samples_per_bit` times. This creates a constant level for each bit across its time duration.
- To convert **Polar NRZ** to **Polar Return-to-Zero (RZ)**, we start with the Polar NRZ waveform.
- We apply the RZ rule by modifying the **second half of each bit period**:
  For every bit, we calculate the index range that corresponds to the second half of its duration and set those values to zero.

- This creates a waveform where the signal returns to zero in the second half of each bit period, while the first half retains the Polar NRZ value (`+A` or `-A`).
- The result is a **Polar RZ** line code that has a non-zero level only during the first half of each bit, making it more suitable for synchronization at the receiver.

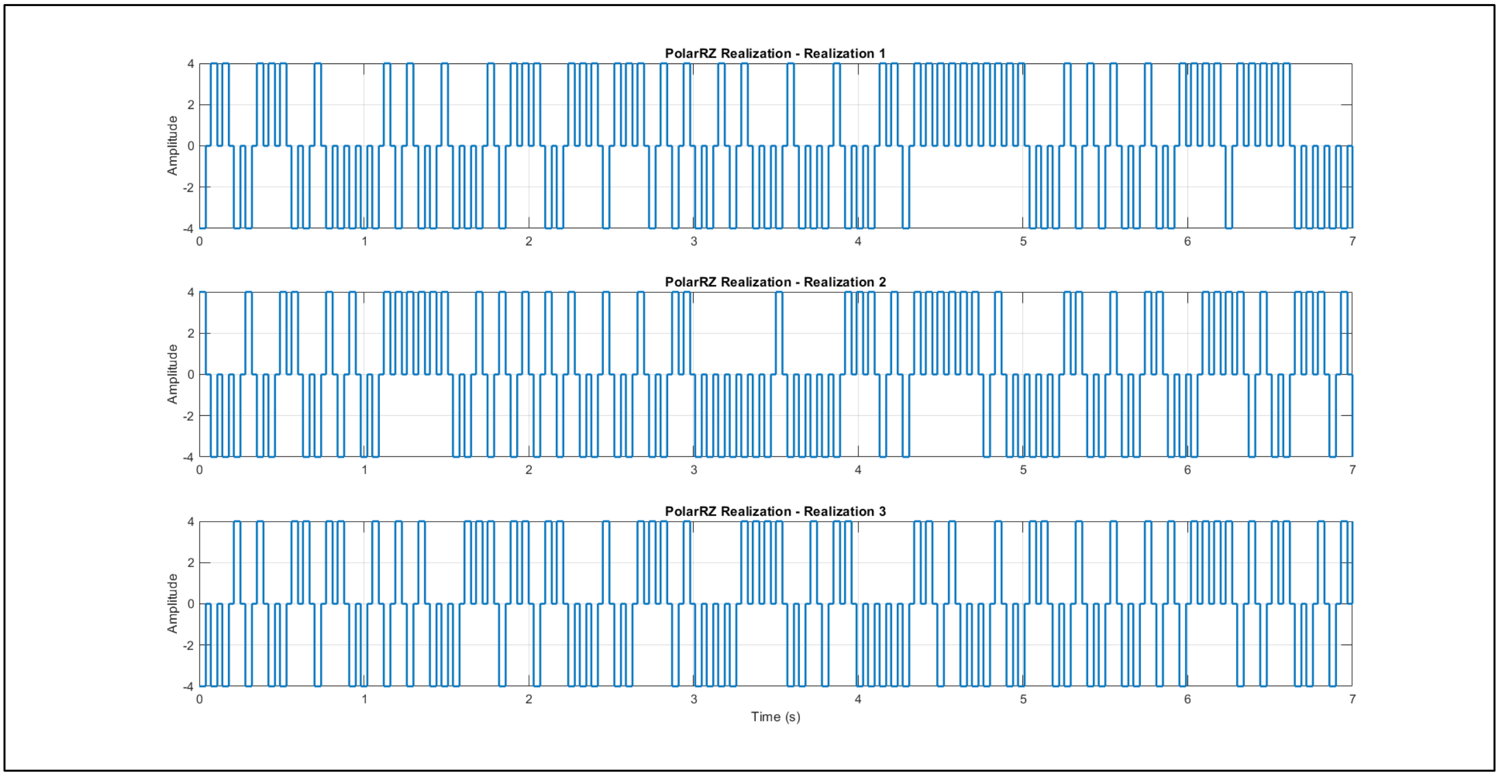

Figure 5 PolarRZ Realization

K. Random initial time shift

For the random shift we made this function:

```
function [Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted] =...
    apply_random_shift_fixed_size(Unipolar_All, PolarNRZ_All, PolarRZ_All, samples_per_bit)...
```

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

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

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

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

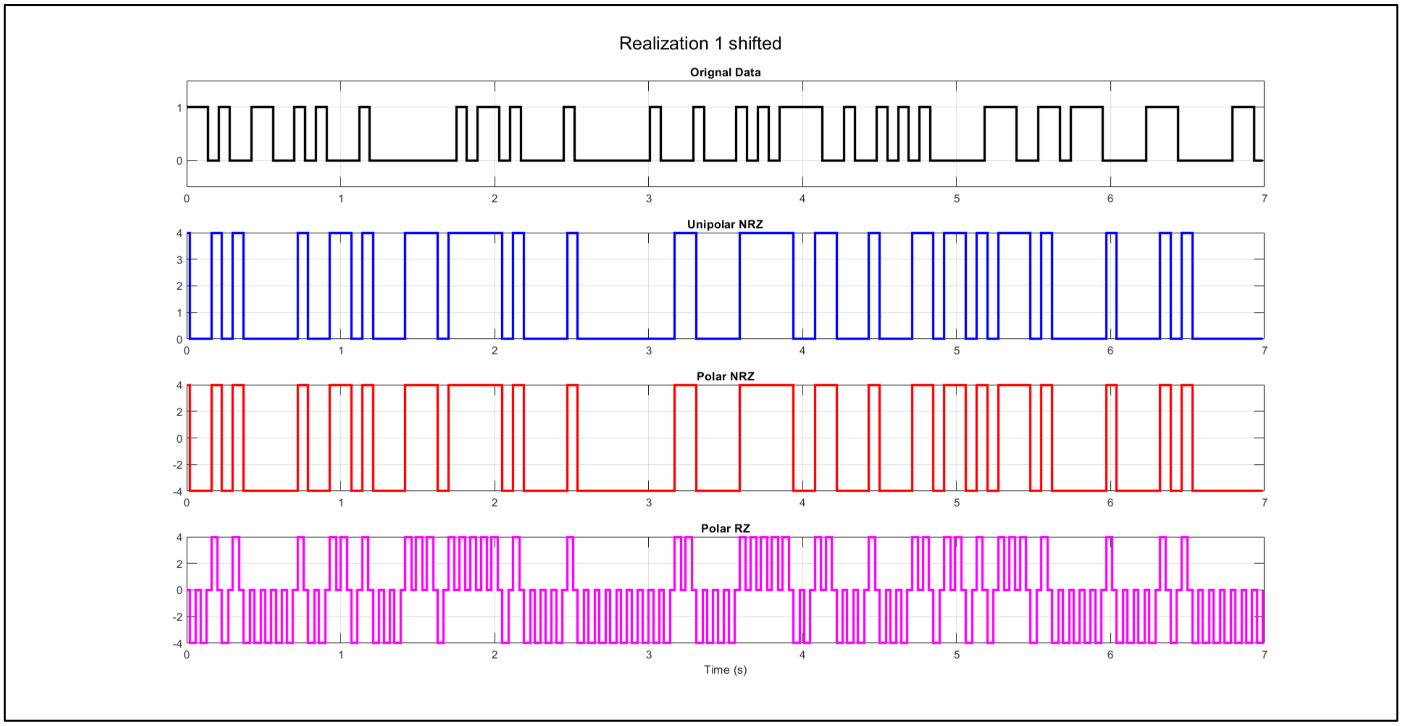

Figure 6 Realization Shifted

L. Getting cell arrays ready to calculate the statistical mean and autocorrelation:

For the mean the cells are ready, as for the autocorrelation we're going to use this function:

```
function [R_unipolar_nrz_t, R_polar_nrz_t, R_polar_rz_t, taw2] = ...
    compute_time_autocorr(UnipolarNRZ, PolarNRZ, PolarRZ)...
```

In which we're making the array ready by shifting it with tau.

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

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

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

So the array will have the length of max tau which is 700.

M. Questions

1. Statistical Mean

1.1. Hand Analysis

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

1.2. Code Snippet

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

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

end
```

- The mean is calculated as μ=ΣX/N (the sum divided by the number of the elements).

1.3.Plotting the Statistical Mean:

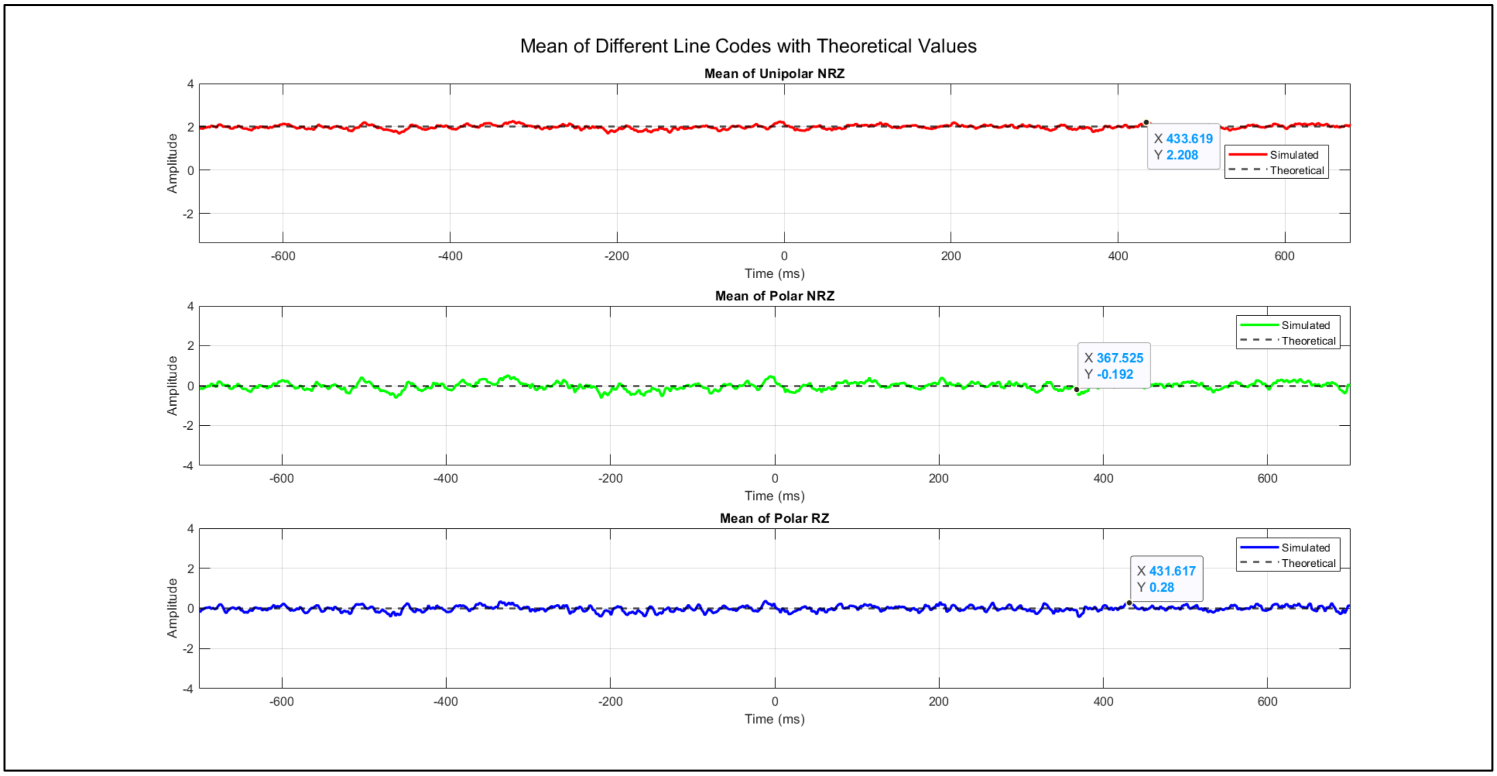

Figure 7 Plot of Statistical Mean

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

2. Statistical Autocorrelation

2.1. Hand Analysis

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

- **For Unipolar NRZ:**
  We have 2 cases **(Considering T to be 70ms or 7 samples),**

  1. $|\tau| < T$

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

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

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

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

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

  2. $|\tau| > T$

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

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

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

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

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

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

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

And as we know:

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

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

	Unipolar NRZ	Polar NRZ	Polar RZ
Total Power	**8**	**16**	**9.147**
DC Power	**4**	**0**	**0**
AC Power	**4**	**16**	**9.147**

2.2.Code Snippet

```
function [Unipolar_AutoCorr, PolarNRZ_AutoCorr, PolarRZ_AutoCorr] =...
    compute_stat_autocorr(Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted, max_lag)
    %{...%}

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

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

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

Annotations

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

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

2.3.Plotting the statistical autocorrelation

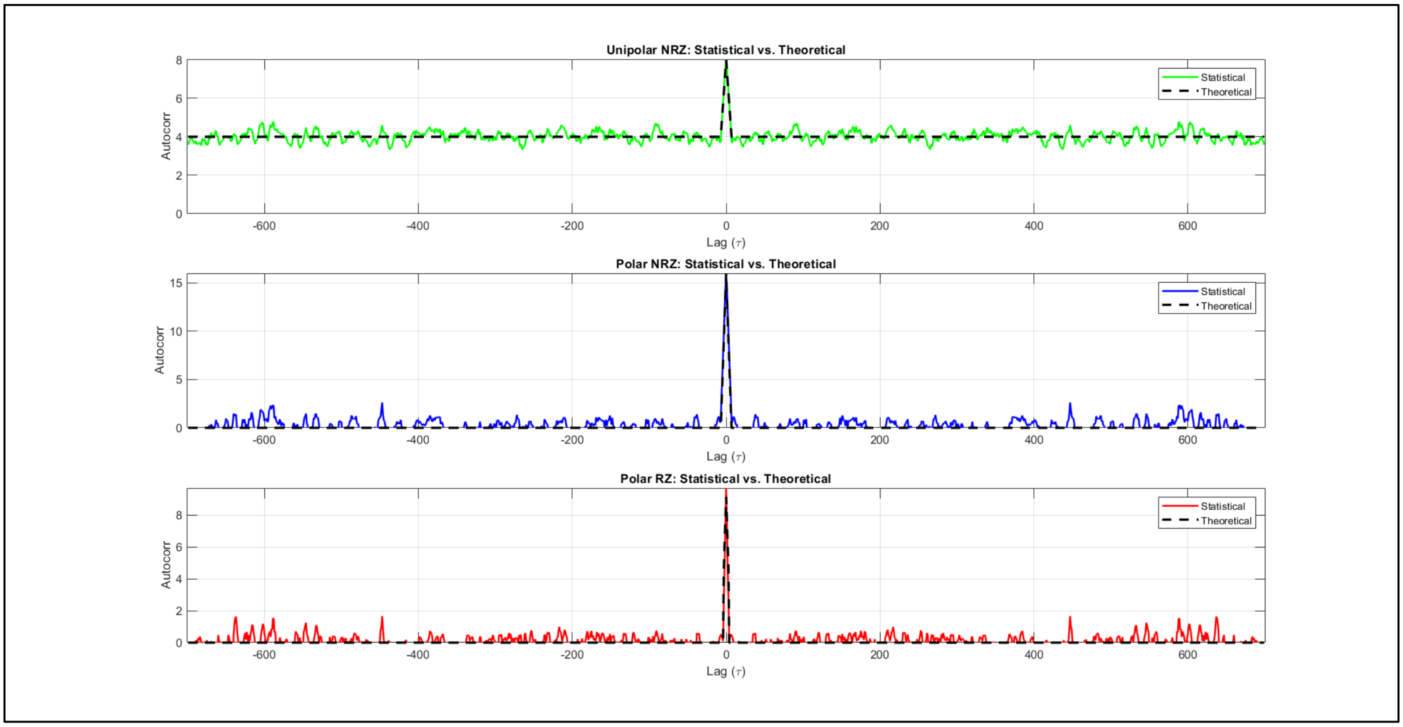

Figure 8 Statistical Auto Correction plot

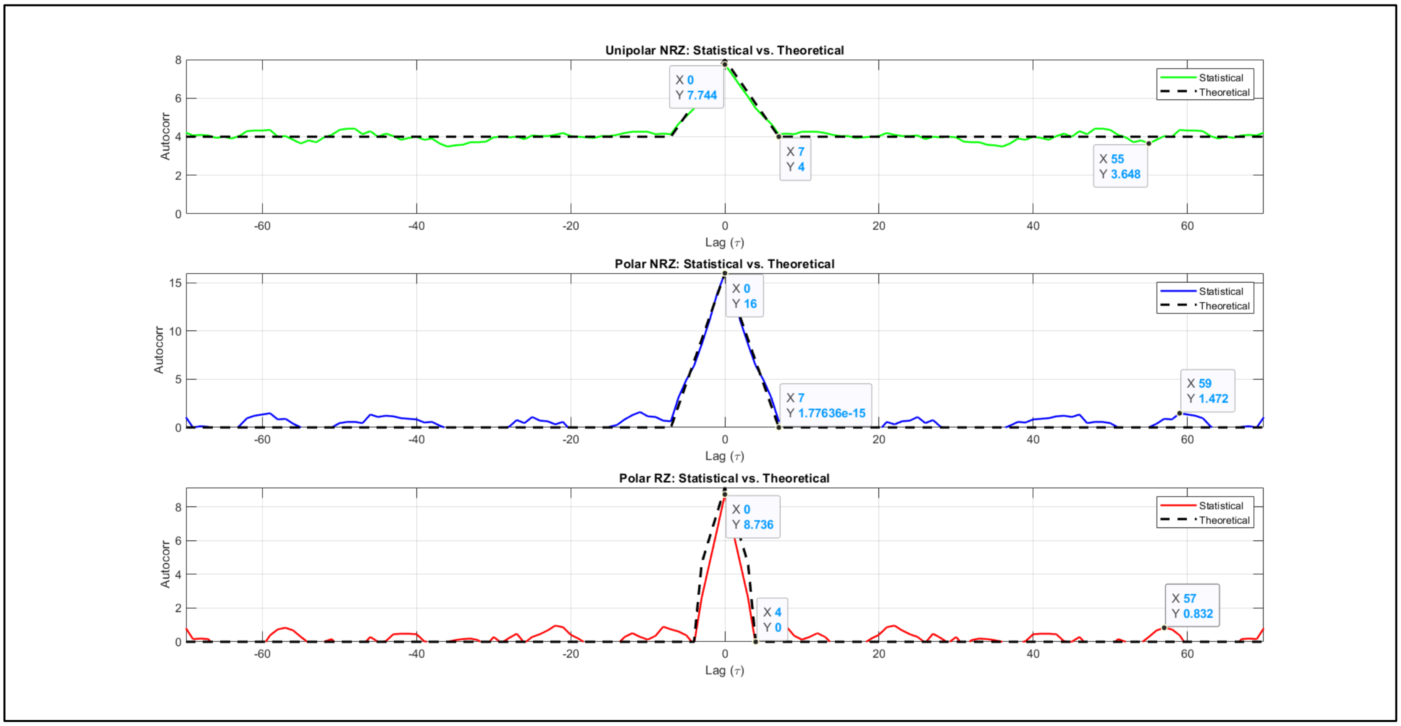

Figure 9 Statistical Auto Correction plot zoomed

The resulting autocorrelation values are plotted against the corresponding time delays (τ). We observe that at $\tau = 0$ the autocorrelation with the point itself is maximum, indicating perfect correlation.

- **Uni polar:** The autocorrelation becomes constant after 7 samples, as we calculated to be the bit duration and it's around 4, The maximum at zero equals 7.744 ≈ 8.

- **Polar NRZ:** The autocorrelation becomes constant after 7 samples, as we calculated to be the bit duration and it's around zero, The maximum at zero equals 16.

- **Polar RZ:** The autocorrelation becomes constant after 4 samples, as we calculated to be the half bit duration and it's around zero, The maximum at zero equals 8.736 ≈ 9.147.

3. Is the Process Stationary

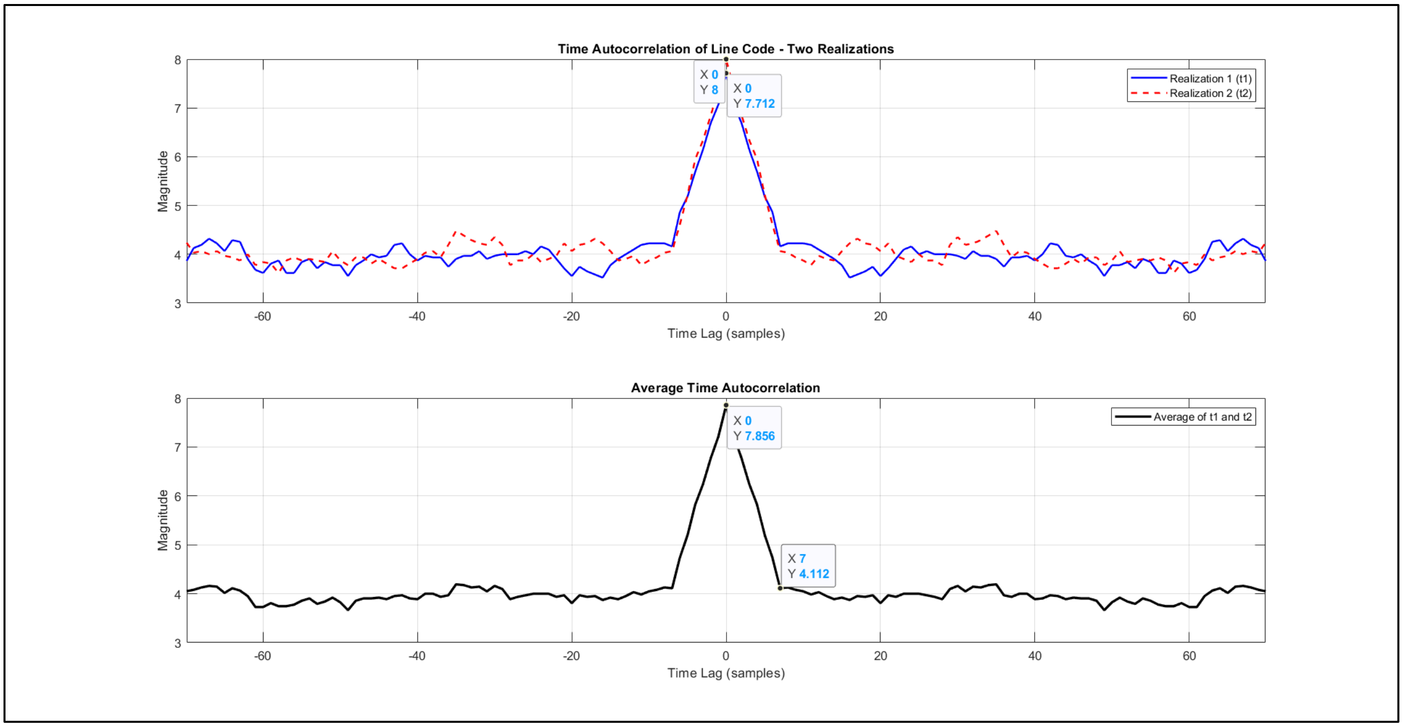

Figure 10 autocorrelation at two different times

- For the **mean**, as shown in section 1 figure 7 the **mean** ≈ **constant with time**.
- For the **autocorrelation**, as shown in figure 9 the **autocorrelation R(t1=1) ≈ R(t2=8).**

Yes, the process is stationary (WSSP) because the mean is constant function in time as shown in Figure 7 Plot of Statistical Mean and the autocorrelation depends only on the time difference not the absolute time.

4. The time mean and autocorrelation function for one waveform

4.1.Time Mean

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

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

- We add the values of a realization across time instant then divide by the number of samples (700 sample per realization).

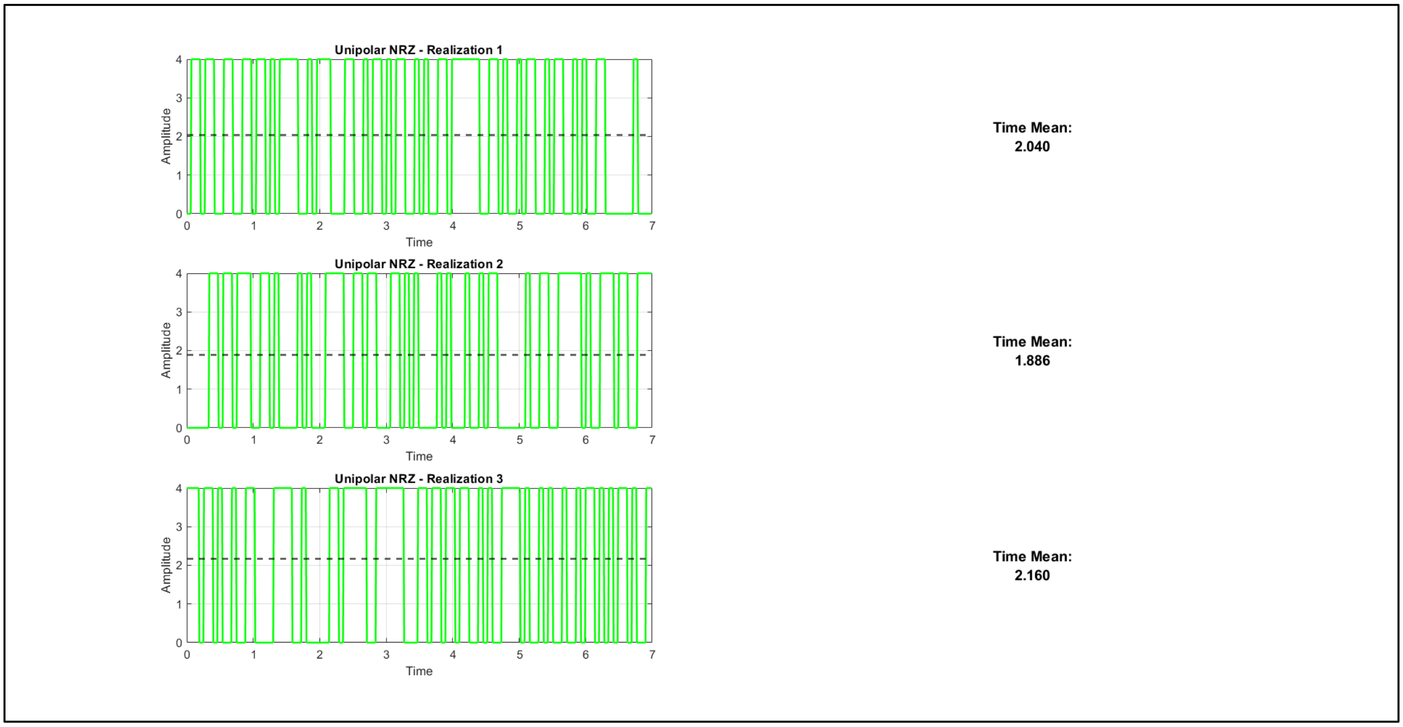

Figure 11 Time Mean for Uni Polar

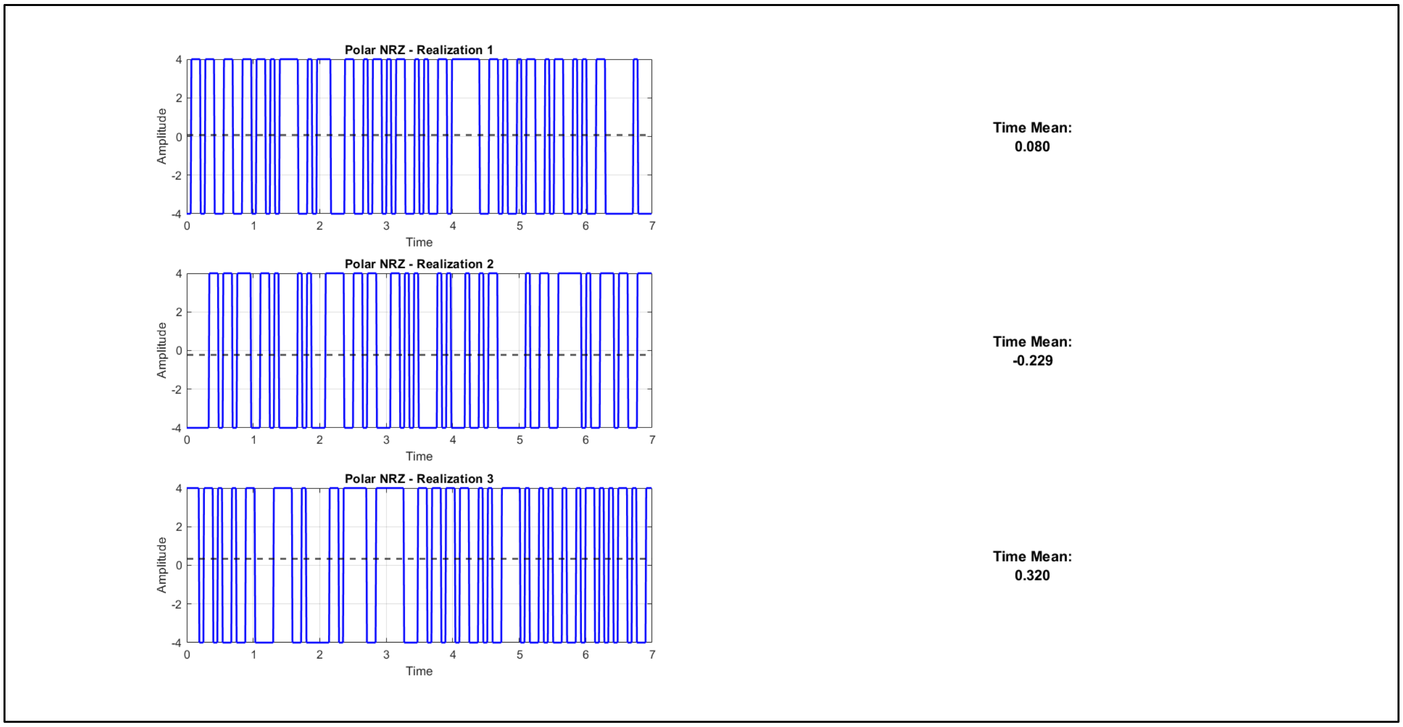

Figure 12 Time Mean for Polar NRZ

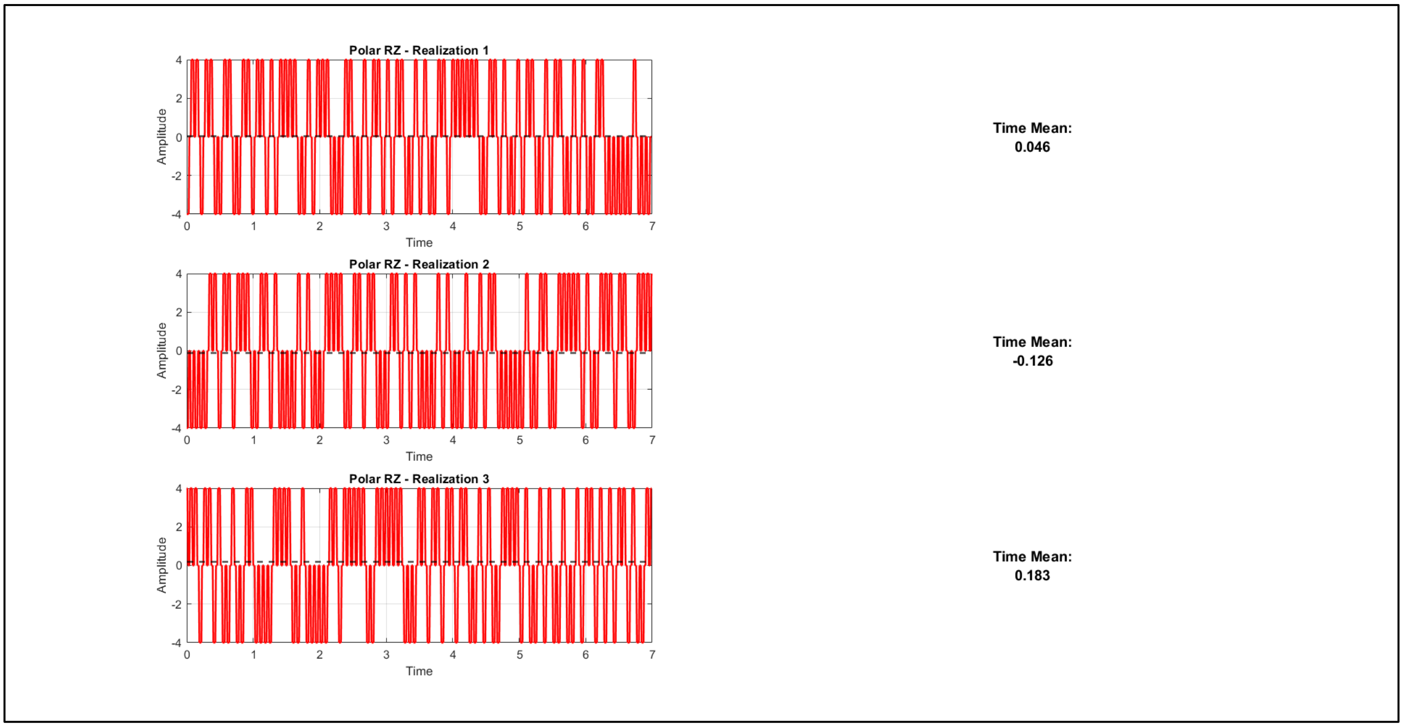

Figure 13 Time Mean for Polar RZ

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

4.2.Time Mean Vs Realization

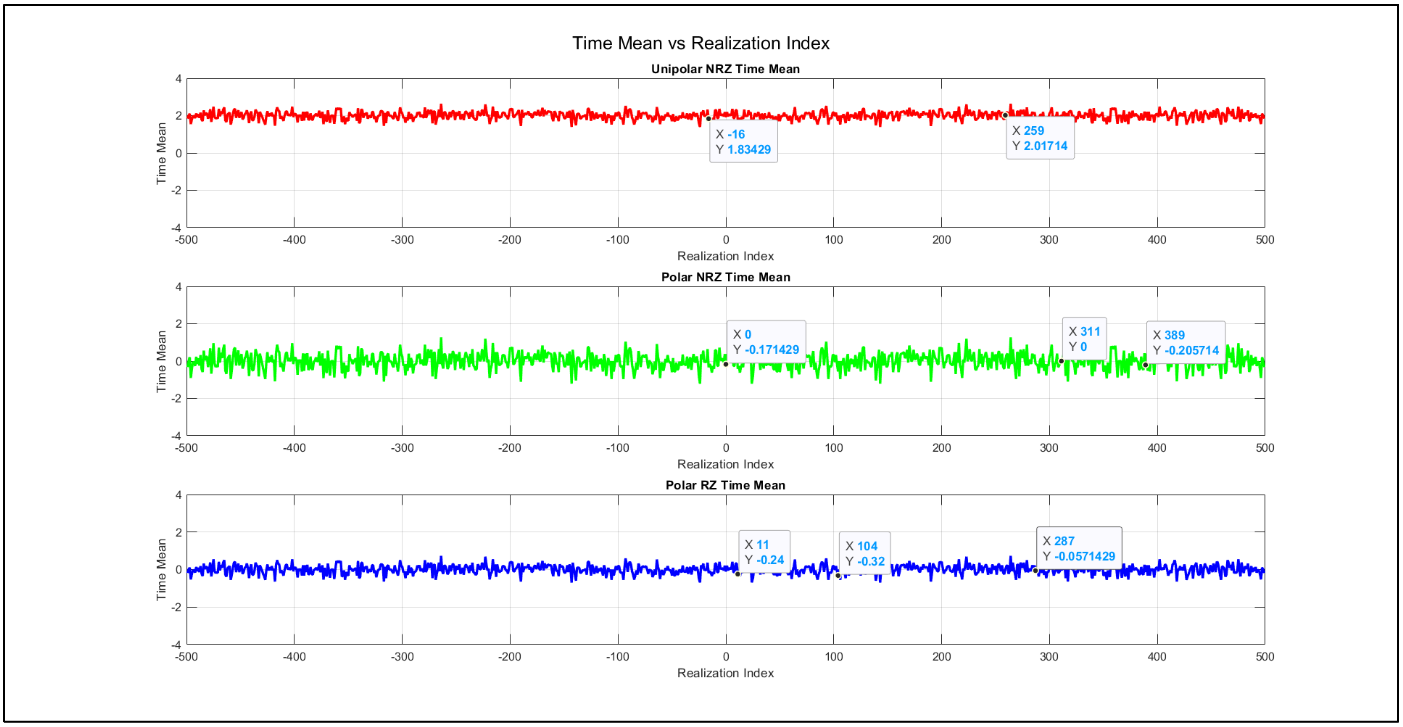

Figure 14 Time Mean Vs Realization

- As expected the time mean is almost equal to the statistical mean.
- Polar RZ & NRZ have almost zero mean and the uni polar has mean around 2.

4.3.Time Auto Correlation

For the time Auto Correlation we're going to use this function

```
function [R_unipolar_nrz_t1, R_polar_nrz_t1, R_polar_rz_t1, tau_vec] = ...
    compute_time_autocorr(UnipolarNRZ, PolarNRZ, PolarRZ, t1)...
```

```
% Preallocate
R_unipolar_nrz_t1 = zeros(1, length(tau_vec));
R_polar_nrz_t1 = zeros(1, length(tau_vec));
R_polar_rz_t1 = zeros(1, length(tau_vec));

for idx = 1:length(tau_vec)
    tau = tau_vec(idx);
    t2 = t1 + tau;


    % Compute element-wise products for all realizations at t1 and t1+tau
    prod_unipolar = UnipolarNRZ(:, t1) .* UnipolarNRZ(:, t2);
    prod_polar    = PolarNRZ(:, t1)    .* PolarNRZ(:, t2);
    prod_rz       = PolarRZ(:, t1)     .* PolarRZ(:, t2);

    % Use custom function to compute mean across realizations
    R_unipolar_nrz_t1(idx) = sum(prod_unipolar) / num_realizations;
    R_polar_nrz_t1(idx)    = sum(prod_polar)    / num_realizations;
    R_polar_rz_t1(idx)     = sum(prod_rz)       / num_realizations;

end
```

- The time autocorrelation is calculated by $r_{xx} = \frac{1}{N}\sum_{n=0}^{N-1} x(n)x(n-k)$ of the first waveform.

4.4.Time Auto Correlation for one wave form:

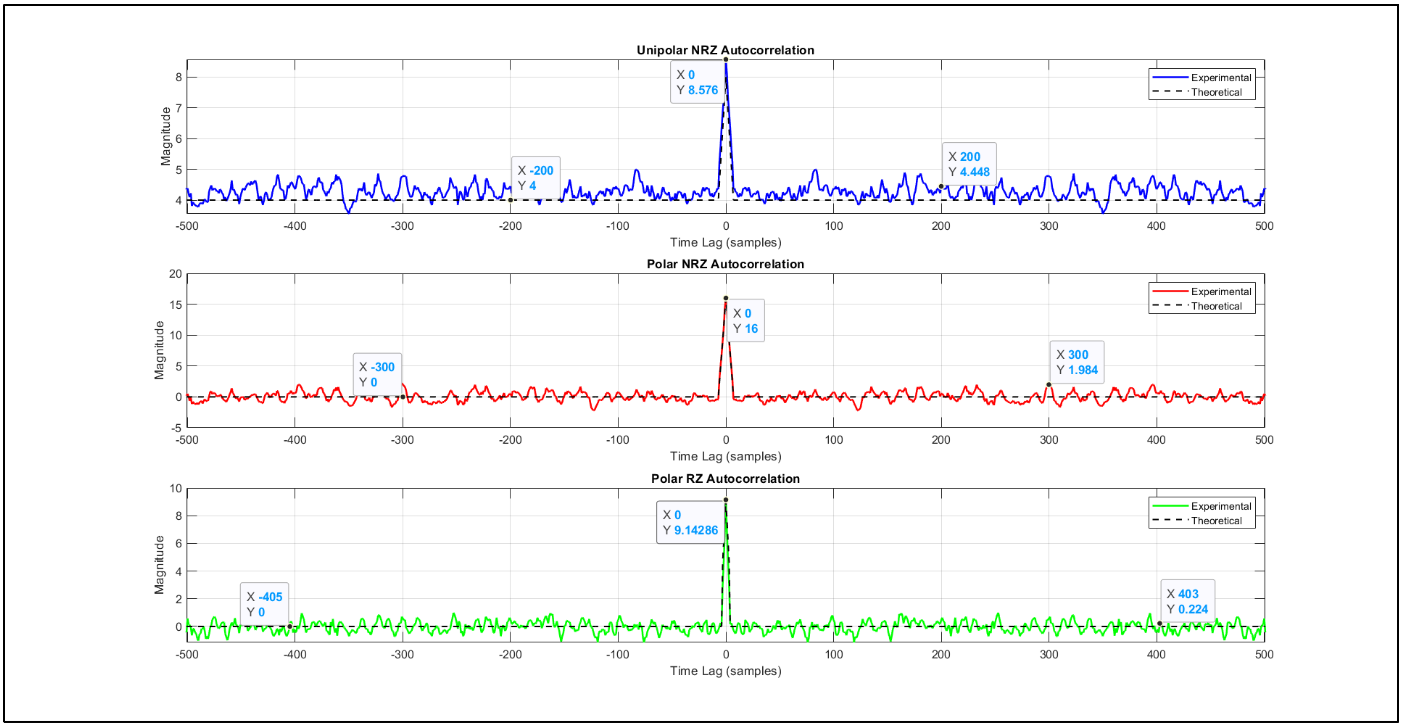

Figure 16 Time Auto Correction plot

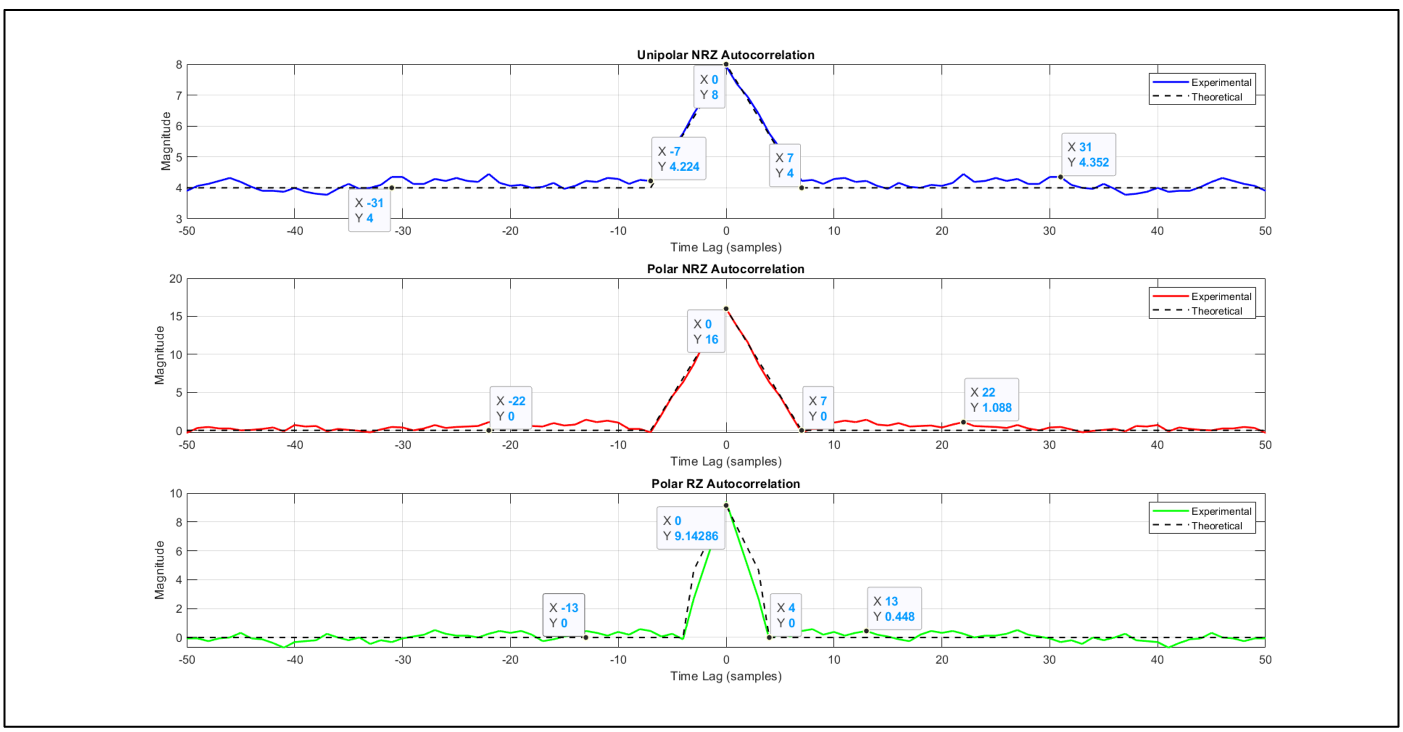

Figure 15 Time Auto Correction plot zoomed

As shown in the graphs:

- The time autocorrelation is closely same as the ensemble autocorrelation.
- The autocorrelation function has maximum at $\tau = 0$ and it is an even function.

5. Is The Random Process Ergodic?

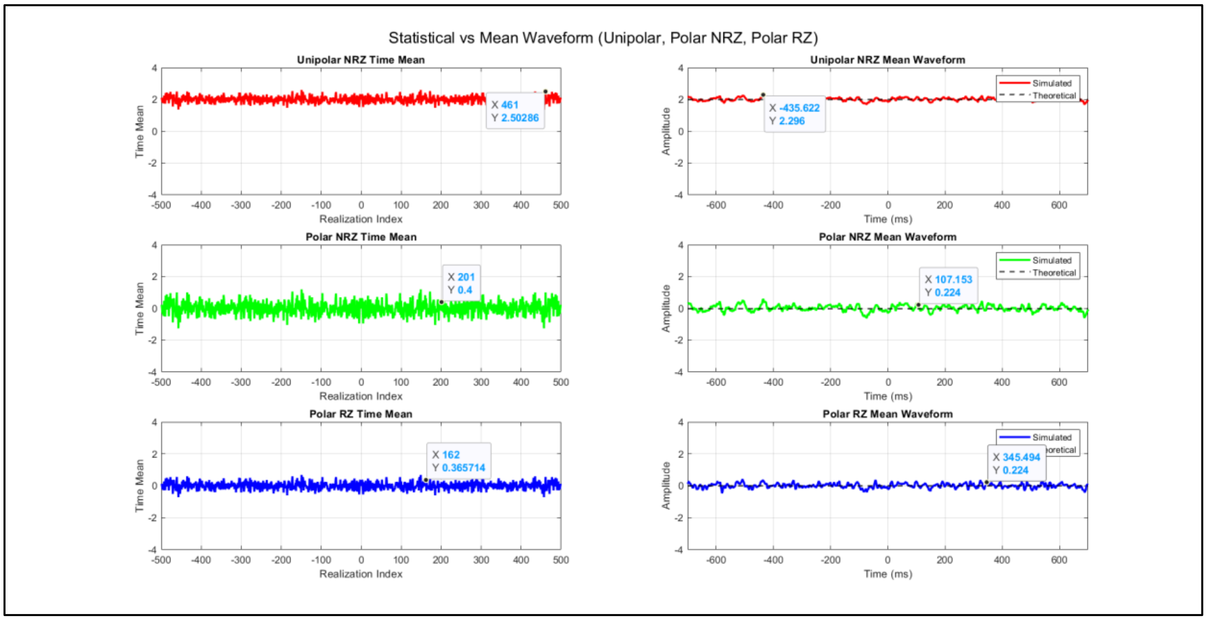

Figure 17 Time Mean vs Statistical

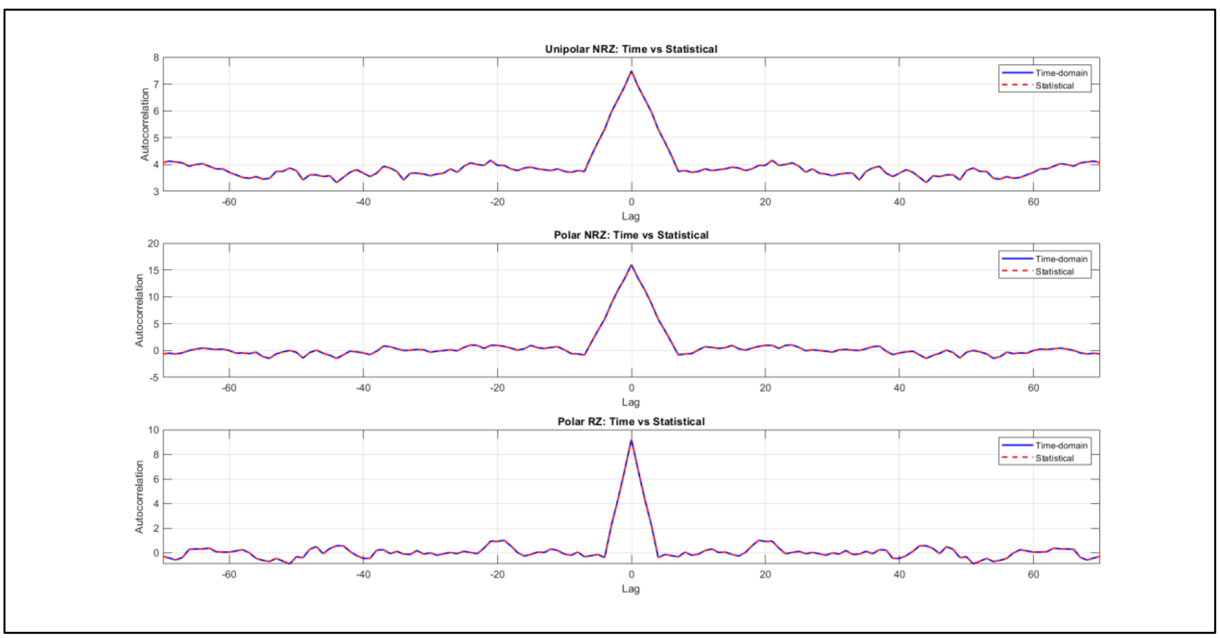

Figure 18 Time Auto Correlation Vs Statistical

- For the **mean**, the Time mean is almost equal to the statistical mean.

- For the **Auto Correlation,** the Time looks almost identical to the statistical.
  But, There not fully identical as we ran this code snippet

  ```
  disp(R_unipolar_full - Unipolar_AutoCorr);
  ```

  And the result was **0.5760, so they are almost Identical.**

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

6. the PSD & Bandwidth of the Ensemble

6.1.PSD using fft:

For the **PSD,** we are going to use this function:

```
function [PSD_unipolar ,PSD_polarNRZ ,PSD_polarRZ] =...
    plot_linecode_psd(R_Unipolar, R_PolarNRZ, R_PolarRZ, fs, A, Tb)
```

```
% Compute FFTs of autocorrelations
fft_unipolar = fft(R_Unipolar) / n;
fft_polarNRZ = fft(R_PolarNRZ) / n;
fft_polarRZ  = fft(R_PolarRZ) / n;

% Compute PSD magnitudes
PSD_unipolar = abs(fft_unipolar);
PSD_polarNRZ = abs(fft_polarNRZ);
PSD_polarRZ  = abs(fft_polarRZ);

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

% Center the FFTs for proper plotting
PSD_unipolar = A*fftshift(PSD_unipolar);
PSD_polarNRZ = A*fftshift(PSD_polarNRZ);
PSD_polarRZ  = A*fftshift(PSD_polarRZ);
```

- We take the Fourier transform of the avg time autocorrelation **= 0.5*(R(t1)+ R(t2))** then centralize the graph around zero.

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

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

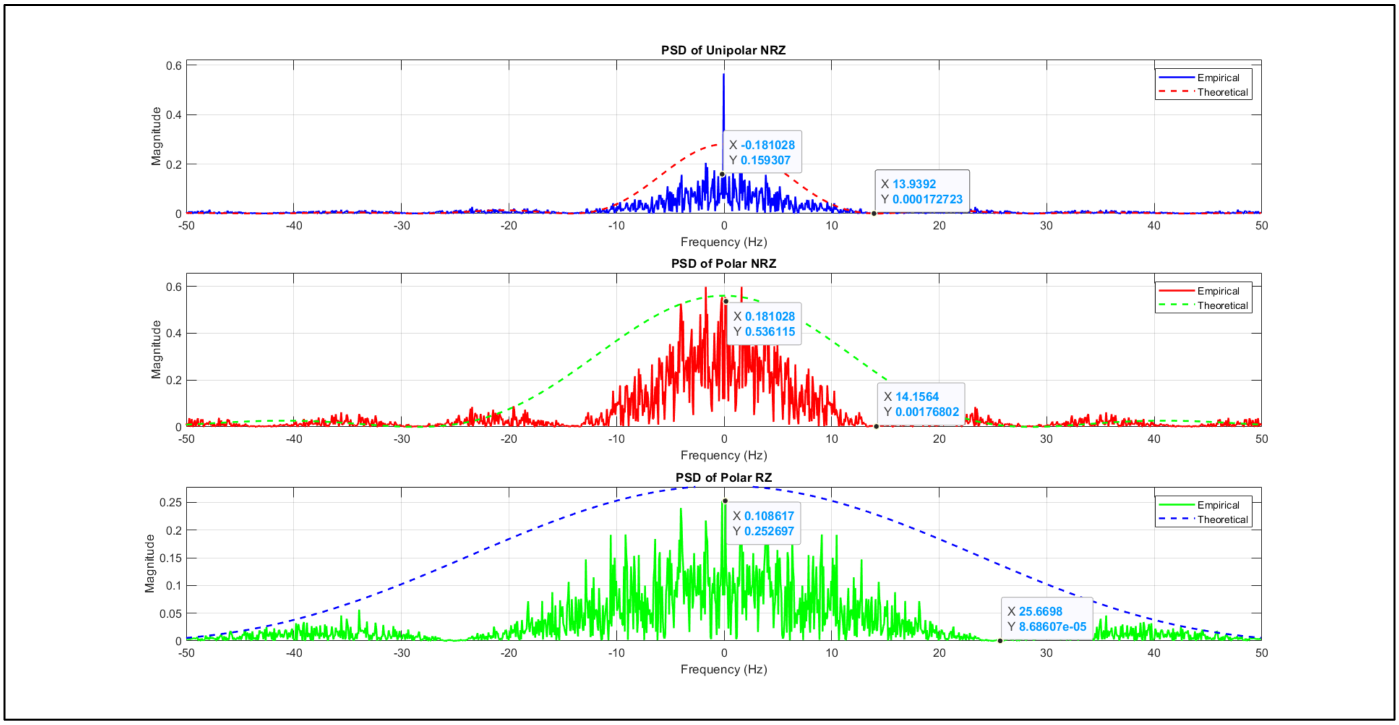

Figure 19 PSD plot of the Ensemble

Annotations

- in polar RZ & NRZ : we have sinc^2 function without delta at zero frequency (NO DC)
- in uni polar NRZ : we have sinc^2 function with delta at zero frequency (there is DC)
- BW of the unipolar NRZ & polar NRZ is the bitrate which approximately equal 14 hz
- BW of the polar RZ is the double of bitrate which approximately equal 25.66 hz

6.2.Theoritical PSD:

From **references[1], [2]**, we found out that the PSDs are:

Line Code	PSD
Uni Polar	$S(f)=A^2/4*Tb*(\sin(\pi fTb)/\pi fTb)^2 + A^2/4* \delta(f)$
Polar NRZ	$S(f)=A^2*Tb*(\sin(\pi fTb/2)/\pi fTb/2)^2$
Polar RZ	$S(f)=A^2*Tb*(\sin(\pi fTb/4)/\pi fTb/4)^2$

Note that:

- Uni polar has a DC pulse which is noticeable in figure 19
- Polar don’t have the DC pulse
- Polar RZ has double the frequency of Polar NRZ
- A=4, Tb = 70 ms

So by comparing the practical vs theoretical:

Line Code	Theoritcal PSD at f=0	Paractical PSD at f=0
Uni Polar	$A^2/4*Tb$ =0.28	0.159
Polar NRZ	$A^2/2*Tb$ =0.56	0.536
Polar RZ	$A^2/4*Tb$ =0.28	0.252

For **BW:**

Line Code	Theoritcal BW	Paractical BW
Uni Polar	1/Tb = 14.285 Hz	13.972 Hz
Polar NRZ	1/Tb = 14.285 Hz	14.15 Hz
Polar RZ	2/Tb = 28.57 Hz	25.66 Hz

N. References:

[1] B. P. Lathi and Z. Ding, *Modern Digital and Analog Communication Systems*, 3rd ed. New York, NY, USA: Oxford University Press, 2009, ch. 7.

[2] S. R. Iyer, “Line Coding PSD – ECE 4001,” *Electronic Duo*, Sep. 2017. [Online]. Available: https://electronicduo.blogspot.com/2017/09/line-coding-psd-ece-4001.html

[3] https://github.com/youefkh05/Digital_Communication_Radio

O. Appendix

```
clc;
clear;
close all;

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

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

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

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

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

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

% plot first three realization
plot_realizations(Unipolar_All, t, 3, 'Uni polar Realization');
plot_realizations(PolarNRZ_All, t, 3, 'PolarNRZ Realization');
plot_realizations(PolarRZ_All, t, 3, 'PolarRZ Realization');


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

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

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

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

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

%plot the mean across time
plot_mean_waveforms(t_shifted, Unipolar_Mean, PolarNRZ_Mean, PolarRZ_Mean, 2, 0, 0, A);

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

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

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

% calculate the autocorrelation (question 3)
[Unipolar_AutoCorr, PolarNRZ_AutoCorr, PolarRZ_AutoCorr] = ...
       compute_stat_autocorr(Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted, max_lag);

% plot the autocorrelation
plot_autocorrelation(Unipolar_AutoCorr, PolarNRZ_AutoCorr, PolarRZ_AutoCorr, max_lag, 100*bit_duration, A);

% Compute time autocorrelation
t1=1;
t2=8;
[R_unipolar_t, R_polar_nrz_t, R_polar_rz_t, taw] =...
    compute_time_autocorr(Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted, t1);

% Plot the time autocorrelation
plot_time_autocorrelation(R_unipolar_t, R_polar_nrz_t, R_polar_rz_t, taw, max_lag, 100*bit_duration, A);

% Check for Staionary (question 2)
[R_unipolar_t2, R_polar_nrz_t2, R_polar_rz_t2, taw2] =...
    compute_time_autocorr(Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted, t2);

plot_two_realizations(R_unipolar_t, R_unipolar_t2, taw, max_lag);

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

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

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

% Check for ergodic (question 5)
plot_mean_time_vs_statistical(t_shifted, ...
                             Unipolar_Mean, PolarNRZ_Mean, PolarRZ_Mean, ...
                             2, 0, 0, ...
                             Unipolar_TimeMean, PolarNRZ_TimeMean, PolarRZ_TimeMean, A);

plot_combined_autocorrelation(R_unipolar_t, R_polar_nrz_t, R_polar_rz_t, ...
                                       taw, max_lag, 100*bit_duration, A, ...
                                       Unipolar_AutoCorr, PolarNRZ_AutoCorr, PolarRZ_AutoCorr);

% Plotting the PSD (Question 6)

[R_avg_unipolar, R_avg_polar_nrz, R_avg_polar_rz, tau_full] = ...
    get_Ravg(Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted, t1, t2);

Ts=bit_duration/samples_per_bit;% Sampling Time
fs = 1/Ts;                      % Sampling Frequency

[PSD_unipolar ,PSD_polarNRZ ,PSD_polarRZ] =...
    plot_linecode_psd(R_avg_unipolar, R_avg_polar_nrz, R_avg_polar_rz, fs, A, bit_duration);


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

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

```
    Data = int8(Data);

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

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

    % Polar NRZ: 0 ? -A, 1 ? +A
    PolarNRZ = int8((2 * Data - 1) * A);
    PolarNRZ = repelem(PolarNRZ, samples_per_bit);

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

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

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

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

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

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

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

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

    subplot(4,1,3);
    stairs(t_show, PolarNRZ_show, 'r', 'LineWidth', 2);
    title('Polar NRZ');
    grid on;

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

    xlabel('Time (s)');
end

function plot_realizations(linecode_all, t, N, title_prefix)
% plot_realizations - Plots the first N realizations of a given line code
%
% Inputs:
%   linecode_all   : Matrix of size [realizations x time] for a line code
%   t              : Time vector corresponding to samples
%   N              : Number of realizations to plot (e.g., 5)
%   title_prefix   : Title prefix string (e.g., 'Unipolar NRZ')
%
% Example usage:
```

```
%   plot_realizations(Unipolar_All, t, 5, 'Unipolar NRZ');

    figure;
    for i = 1:N
        subplot(N, 1, i);
        stairs(t, linecode_all(i, :), 'LineWidth', 1.5);
        title([title_prefix ' - Realization ' num2str(i)]);
        ylabel('Amplitude');
        xlim([0 7]); % Focus on the first 7 seconds
        grid on;
        if i == N
            xlabel('Time (s)');
        end
    end

end

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

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

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

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

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

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

end

function plot_mean_waveforms(t, Unipolar_Mean, PolarNRZ_Mean, PolarRZ_Mean, ...
                             Unipolar_Theoretical, PolarNRZ_Theoretical, PolarRZ_Theoretical, A)
% Function to plot the mean waveforms of different line codes across time
% with their corresponding theoretical mean values
%
% Inputs:
%   t                    - Time vector (ignored in favor of fixed [-700, 700] mapping)
%   Unipolar_Mean        - Simulated mean waveform of Unipolar NRZ
%   PolarNRZ_Mean        - Simulated mean waveform of Polar NRZ
%   PolarRZ_Mean         - Simulated mean waveform of Polar RZ
%   Unipolar_Theoretical - Theoretical mean value for Unipolar NRZ
%   PolarNRZ_Theoretical - Theoretical mean value for Polar NRZ
%   PolarRZ_Theoretical  - Theoretical mean value for Polar RZ
%   A                    - Amplitude limit for y-axis

    % Create new time vector from -700 to 700 ms based on length of input
    t_ms = linspace(-700, 700, length(t));

    figure;

    % --- Unipolar NRZ ---
    subplot(3,1,1);
    plot(t_ms, Unipolar_Mean, 'r', 'LineWidth', 2); hold on;
    yline(Unipolar_Theoretical, '--k', 'LineWidth', 1.5);
    xlabel('Time (ms)');
    ylabel('Amplitude');
    title('Mean of Unipolar NRZ');
    grid on;
    ylim([-A, A]);
    xlim([-length(t)-1, length(t)+1]);
    legend('Simulated', 'Theoretical');
```

```
    % --- Polar NRZ ---
    subplot(3,1,2);
    plot(t_ms, PolarNRZ_Mean, 'g', 'LineWidth', 2); hold on;
    yline(PolarNRZ_Theoretical, '--k', 'LineWidth', 1.5);
    xlabel('Time (ms)');
    ylabel('Amplitude');
    title('Mean of Polar NRZ');
    grid on;
    ylim([-A, A]);
    xlim([-length(t)-1, length(t)+1]);
    legend('Simulated', 'Theoretical');

    % --- Polar RZ ---
    subplot(3,1,3);
    plot(t_ms, PolarRZ_Mean, 'b', 'LineWidth', 2); hold on;
    yline(PolarRZ_Theoretical, '--k', 'LineWidth', 1.5);
    xlabel('Time (ms)');
    ylabel('Amplitude');
    title('Mean of Polar RZ');
    grid on;
    ylim([-A, A]);
    xlim([-length(t)-1, length(t)+1]);
    legend('Simulated', 'Theoretical');

    sgtitle('Mean of Different Line Codes with Theoretical Values');
end

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

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

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

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

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

end

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

    figure;

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

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

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

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

function [Unipolar_AutoCorr, PolarNRZ_AutoCorr, PolarRZ_AutoCorr] =...
    compute_stat_autocorr(Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted, max_lag)
    % Compute Statistical Autocorrelation for given signals using calculate_mean
```

```
    % Inputs:
    %   Unipolar_Shifted - Shifted signal matrix for Unipolar NRZ
    %   PolarNRZ_Shifted - Shifted signal matrix for Polar NRZ
    %   PolarRZ_Shifted - Shifted signal matrix for Polar RZ
    % Outputs:
    %   Unipolar_AutoCorr - Computed autocorrelation for Unipolar NRZ
    %   PolarNRZ_AutoCorr - Computed autocorrelation for Polar NRZ
    %   PolarRZ_AutoCorr - Computed autocorrelation for Polar RZ

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

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

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

end

function plot_autocorrelation(Unipolar_AutoCorr, PolarNRZ_AutoCorr, PolarRZ_AutoCorr, max_lag, Tb, A)
    % Plots statistical and theoretical autocorrelation for 3 line codes
    % Inputs:
    %   Unipolar_AutoCorr, PolarNRZ_AutoCorr, PolarRZ_AutoCorr - Statistical autocorrelations
    %   max_lag - Maximum lag
    %   Tb - Bit duration
    %   A - Amplitude

    % Time axis for lags
    t = -max_lag:max_lag;
    tau = abs(t);  % Use absolute lag for symmetry
    x_limit = max_lag / 10;

    % -------- Theoretical Expressions -------- %

    % Unipolar NRZ
    Rx_Unipolar = (tau < Tb) .* ((A^2 / 2) .* (1 - ( tau / (2*Tb)))) + ...
                  (tau >= Tb) .* (A^2 / 4);
    % Polar NRZ
    Rx_PolarNRZ = (tau < Tb) .* (A^2 .* (1 - (tau / Tb)));

    % Polar RZ
    Rx_PolarRZ = (tau < Tb/2) .* ((4/7) * A^2 .* (1 - (8 * tau ./ (7 * Tb))));

    % -------- Plotting -------- %

    figure("Name", "Statistical & Theoretical Autocorrelation");

    subplot(3,1,1);
    plot(t, Unipolar_AutoCorr, 'g', 'LineWidth', 1.5); hold on;
    plot(t, Rx_Unipolar, '--k', 'LineWidth', 2);
    legend('Statistical', 'Theoretical');
    xlim([-701, 701]);
    ylim([0, inf]);
    xlabel("Lag (\tau)");
    ylabel("Autocorr");
    title("Unipolar NRZ: Statistical vs. Theoretical");
    grid on;

    subplot(3,1,2);
    plot(t, PolarNRZ_AutoCorr, 'b', 'LineWidth', 1.5); hold on;
    plot(t, Rx_PolarNRZ, '--k', 'LineWidth', 2);
    legend('Statistical', 'Theoretical');
    xlim([-701, 701]);
    ylim([0, inf]);
    xlabel("Lag (\tau)");
    ylabel("Autocorr");
    title("Polar NRZ: Statistical vs. Theoretical");
    grid on;

    subplot(3,1,3);
    plot(t, PolarRZ_AutoCorr, 'r', 'LineWidth', 1.5); hold on;
    plot(t, Rx_PolarRZ, '--k', 'LineWidth', 2);
```

```matlab
    legend('Statistical', 'Theoretical');
    xlim([-701, 701]);
    ylim([0, inf]);
    xlabel("Lag (\tau)");
    ylabel("Autocorr");
    title("Polar RZ: Statistical vs. Theoretical");
    grid on;
end

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

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

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

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


    for i = 1:3
        % First Column: Plot the waveform realization
        subplot(3,2,(i-1)*2+1);
        plot(t_shifted, signals_waveform(i,:), color, 'LineWidth', 1.5); % Plot waveform
        hold on;
        yline(Signals_TimeMean(i), '--k', 'LineWidth', 1.5); % Add time mean line
        hold off;
        xlabel('Time');
        ylabel('Amplitude');
        title([signal_name, ' - Realization ', num2str(i)]);
        grid on;

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

function plot_time_mean_vs_realization(unipolar_mean, polarNRZ_mean, polarRZ_mean, A)
    % Function to plot the time mean vs realization index (symmetric around 0)
    %
    % Inputs:
    % - unipolar_mean: Time mean of Unipolar NRZ (1xN or Nx1 vector)
    % - polarNRZ_mean: Time mean of Polar NRZ (1xN or Nx1 vector)
    % - polarRZ_mean: Time mean of Polar RZ (1xN or Nx1 vector)
    % - A: Amplitude limit for y-axis

    % Ensure row vectors
    unipolar_mean = unipolar_mean(:).';
    polarNRZ_mean = polarNRZ_mean(:).';
    polarRZ_mean = polarRZ_mean(:).';

    N = length(unipolar_mean);
    realization_indices = -N+1:N-1;  % match mirrored length = 2N - 1

    % Mirror signals (excluding duplicated center)
    unipolar_mirrored = [fliplr(unipolar_mean(2:end)), unipolar_mean];
    polarNRZ_mirrored = [fliplr(polarNRZ_mean(2:end)), polarNRZ_mean];
    polarRZ_mirrored = [fliplr(polarRZ_mean(2:end)), polarRZ_mean];

    figure;

    subplot(3,1,1);
    plot(realization_indices, unipolar_mirrored, 'r', 'LineWidth', 2);
    grid on;
    xlabel('Realization Index');
```

```
    ylabel('Time Mean');
    title('Unipolar NRZ Time Mean');
    ylim([-A, A]);

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

    subplot(3,1,3);
    plot(realization_indices, polarRZ_mirrored, 'b', 'LineWidth', 2);
    grid on;
    xlabel('Realization Index');
    ylabel('Time Mean');
    title('Polar RZ Time Mean');
    ylim([-A, A]);

    sgtitle('Time Mean vs Realization Index');
end

function [R_unipolar_t1, R_polar_nrz_t1, R_polar_rz_t1, tau_vec] = ...
    compute_time_autocorr(UnipolarNRZ, PolarNRZ, PolarRZ, t1)

    % Computes time autocorrelation R(t1, tau) at a fixed t1 for tau = 0:700
    %
    % Inputs:
    %   UnipolarNRZ, PolarNRZ, PolarRZ - Realizations (each row is a signal)
    %   t1 - Time index to fix (must be positive and < num_samples)
    %
    % Outputs:
    %   R_unipolar_t1, R_polar_nrz_t1, R_polar_rz_t1 - Autocorrelation vectors
    %   tau_vec - Vector of lags (tau)

    [num_realizations, num_samples] = size(UnipolarNRZ);
    max_tau = 690;
    tau_vec = 0:max_tau;


    % Preallocate
    R_unipolar_t1 = zeros(1, length(tau_vec));
    R_polar_nrz_t1 = zeros(1, length(tau_vec));
    R_polar_rz_t1 = zeros(1, length(tau_vec));

    for idx = 1:length(tau_vec)
        tau = tau_vec(idx);
        t2 = t1 + tau;


        % Compute element-wise products for all realizations at t1 and t1+tau
        prod_unipolar = UnipolarNRZ(:, t1) .* UnipolarNRZ(:, t2);
        prod_polar    = PolarNRZ(:, t1)    .* PolarNRZ(:, t2);
        prod_rz       = PolarRZ(:, t1)     .* PolarRZ(:, t2);

        % Use custom function to compute mean across realizations
        R_unipolar_t1(idx) = sum(prod_unipolar) / num_realizations;
        R_polar_nrz_t1(idx)    = sum(prod_polar)    / num_realizations;
        R_polar_rz_t1(idx)     = sum(prod_rz)       / num_realizations;

    end
end

function plot_time_autocorrelation(R_unipolar, R_polarNRZ, R_polarRZ, tau_vec, max_lag, Tb, A)
    % Plots experimental and theoretical time autocorrelation for each waveform type.
    %
    % Inputs:
    %   R_unipolar, R_polarNRZ, R_polarRZ - matrices of time autocorrelation (each row = realization)
    %   tau_vec - Vector of lags (non-negative only)
    %   max_lag - Maximum lag value (samples) to define axis limits
    %   Tb - Bit duration
    %   A - Amplitude

    x_limit = max_lag / 10;

    % Full tau range including negative lags
    tau_full = [-flip(tau_vec(2:end)), tau_vec];  % symmetric lags (excluding 0 twice)

    % Flip the autocorrelation to complete the negative half
    R_unipolar_full = [fliplr(R_unipolar(1,2:end)), R_unipolar(1,:)];
```

```
    R_polarNRZ_full = [fliplr(R_polarNRZ(1,2:end)), R_polarNRZ(1,:)];
    R_polarRZ_full  = [fliplr(R_polarRZ(1,2:end)),  R_polarRZ(1,:)];

    % Theoretical expressions
    tau_sec = tau_full * (Tb / 7);  % Convert to seconds assuming 10 samples per Tb

    % -------- Theoretical Expressions -------- %

    Rx_Unipolar = (abs(tau_sec) < Tb) .* ((A^2 / 2) .* (1 - (abs(tau_sec) / (2*Tb)))) + ...
                  (abs(tau_sec) >= Tb) .* (A^2 / 4);

    Rx_PolarNRZ = (abs(tau_sec) < Tb) .* (A^2 .* (1 - abs(tau_sec) / Tb));

    Rx_PolarRZ = (abs(tau_sec) < Tb/2) .* ((4/7) * A^2 .* (1 - (8 * abs(tau_sec) ./ (7 * Tb))));

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

    % Unipolar NRZ
    subplot(3,1,1);
    plot(tau_full, R_unipolar_full, 'b', 'LineWidth', 1.5); hold on;
    plot(tau_full, Rx_Unipolar, '--k', 'LineWidth', 1.2);
    xlim([-501, 501]);
    grid on;
    xlabel('Time Lag (samples)'); ylabel('Magnitude');
    title('Unipolar NRZ Autocorrelation'); legend('Experimental', 'Theoretical');

    % Polar NRZ
    subplot(3,1,2);
    plot(tau_full, R_polarNRZ_full, 'r', 'LineWidth', 1.5); hold on;
    plot(tau_full, Rx_PolarNRZ, '--k', 'LineWidth', 1.2);
    xlim([-501, 501]);
    grid on;
    xlabel('Time Lag (samples)'); ylabel('Magnitude');
    title('Polar NRZ Autocorrelation'); legend('Experimental', 'Theoretical');

    % Polar RZ
    subplot(3,1,3);
    plot(tau_full, R_polarRZ_full, 'g', 'LineWidth', 1.5); hold on;
    plot(tau_full, Rx_PolarRZ, '--k', 'LineWidth', 1.2);
    xlim([-501, 501]);
    grid on;
    xlabel('Time Lag (samples)'); ylabel('Magnitude');
    title('Polar RZ Autocorrelation'); legend('Experimental', 'Theoretical');
end

function plot_mean_time_vs_statistical(t, ...
                                       Unipolar_Mean, PolarNRZ_Mean, PolarRZ_Mean, ...
                                       Unipolar_Theoretical, PolarNRZ_Theoretical, PolarRZ_Theoretical, ...
                                       Unipolar_TimeMean, PolarNRZ_TimeMean, PolarRZ_TimeMean, A)
% Plot statistical (time mean vs realization) and mean waveform vs time side by side
%
% Inputs:
%   t                       - Time vector
%   *_Mean                  - Simulated mean waveforms
%   *_Theoretical           - Theoretical mean values
%   *_TimeMean              - Time mean for each realization
%   A                       - Amplitude limit for Y-axis

    % Ensure row vectors
    Unipolar_TimeMean = Unipolar_TimeMean(:).';
    PolarNRZ_TimeMean = PolarNRZ_TimeMean(:).';
    PolarRZ_TimeMean = PolarRZ_TimeMean(:).';

    % Time axis for waveform mean (same as in original)
    t_ms = linspace(-700, 700, length(t));

    % Realization indices (symmetric for mirroring)
    N = length(Unipolar_TimeMean);
    realization_indices = -N+1:N-1;

    % Mirror time mean data (excluding duplicated center)
    unipolar_mirrored = [fliplr(Unipolar_TimeMean(2:end)), Unipolar_TimeMean];
    polarNRZ_mirrored = [fliplr(PolarNRZ_TimeMean(2:end)), PolarNRZ_TimeMean];
    polarRZ_mirrored = [fliplr(PolarRZ_TimeMean(2:end)), PolarRZ_TimeMean];

    figure;

    % Unipolar
    subplot(3,2,1);  % Row 1, Col 1
    plot(realization_indices, unipolar_mirrored, 'r', 'LineWidth', 2);
```

```
    xlabel('Realization Index'); ylabel('Time Mean');
    title('Unipolar NRZ Time Mean');
    ylim([-A, A]); grid on;

    subplot(3,2,2);  % Row 1, Col 2
    plot(t_ms, Unipolar_Mean, 'r', 'LineWidth', 2); hold on;
    yline(Unipolar_Theoretical, '--k', 'LineWidth', 1.5);
    xlabel('Time (ms)'); ylabel('Amplitude');
    title('Unipolar NRZ Mean Waveform');
    legend('Simulated', 'Theoretical'); grid on;
    ylim([-A, A]); xlim([min(t_ms), max(t_ms)]);

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

    subplot(3,2,4);
    plot(t_ms, PolarNRZ_Mean, 'g', 'LineWidth', 2); hold on;
    yline(PolarNRZ_Theoretical, '--k', 'LineWidth', 1.5);
    xlabel('Time (ms)'); ylabel('Amplitude');
    title('Polar NRZ Mean Waveform');
    legend('Simulated', 'Theoretical'); grid on;
    ylim([-A, A]); xlim([min(t_ms), max(t_ms)]);

    % Polar RZ
    subplot(3,2,5);
    plot(realization_indices, polarRZ_mirrored, 'b', 'LineWidth', 2);
    xlabel('Realization Index'); ylabel('Time Mean');
    title('Polar RZ Time Mean');
    ylim([-A, A]); grid on;

    subplot(3,2,6);
    plot(t_ms, PolarRZ_Mean, 'b', 'LineWidth', 2); hold on;
    yline(PolarRZ_Theoretical, '--k', 'LineWidth', 1.5);
    xlabel('Time (ms)'); ylabel('Amplitude');
    title('Polar RZ Mean Waveform');
    legend('Simulated', 'Theoretical'); grid on;
    ylim([-A, A]); xlim([min(t_ms), max(t_ms)]);

    sgtitle('Statistical vs Mean Waveform (Unipolar, Polar NRZ, Polar RZ)');
end


function [R_avg] = plot_two_realizations(R_linecode1, R_linecode2, tau_vec, max_lag)
    % Plots the time autocorrelation for two realizations of a single line code,
    % and their average.
    %
    % Inputs:
    %   R_linecode1, R_linecode2 - Vectors of time autocorrelation (one per realization)
    %   tau_vec - Vector of lags (non-negative only)
    %   max_lag - Maximum lag value (samples) to define axis limits

    x_limit = max_lag / 10;

    % Ensure inputs are row vectors
    R_linecode1 = R_linecode1(:).';  % force to row vector
    R_linecode2 = R_linecode2(:).';

    % Full tau range including negative lags
    tau_full = [-flip(tau_vec(2:end)), tau_vec];  % symmetric lags

    % Construct full autocorrelation by mirroring
    R_linecode1_full = [fliplr(R_linecode1(2:end)), R_linecode1];
    R_linecode2_full = [fliplr(R_linecode2(2:end)), R_linecode2];

    % Compute average autocorrelation
    R_avg = 0.5 * (R_linecode1_full + R_linecode2_full);

    % Plotting
    figure('Name', 'Time Autocorrelation for Two Realizations + Average');

    % --- First subplot: the two realizations
    subplot(2,1,1);
    plot(tau_full, R_linecode1_full, 'b', 'LineWidth', 1.5); hold on;
    plot(tau_full, R_linecode2_full, 'r--', 'LineWidth', 1.5);
    grid on;
    xlim([-x_limit x_limit]);
    xlabel('Time Lag (samples)');
```

```
    ylabel('Magnitude');
    title('Time Autocorrelation of Line Code - Two Realizations');
    legend('Realization 1 (t1)', 'Realization 2 (t2)');

    % --- Second subplot: average autocorrelation
    subplot(2,1,2);
    plot(tau_full, R_avg, 'k', 'LineWidth', 2);
    grid on;
    xlim([-x_limit x_limit]);
    xlabel('Time Lag (samples)');
    ylabel('Magnitude');
    title('Average Time Autocorrelation');
    legend('Average of t1 and t2');
end

function [R_avg_unipolar, R_avg_polar_nrz, R_avg_polar_rz, tau_full] = ...
    get_Ravg(Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted, t1, t2)
% Computes the average time autocorrelation (R_avg) of each line code over two time instances
%
% Inputs:
%   Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted : matrices with realizations over time
%   t1, t2       : time indices to extract 2 realizations
%
% Outputs:
%   R_avg_unipolar, R_avg_polar_nrz, R_avg_polar_rz : averaged autocorrelations
%   tau_full     : vector of symmetric lag values

    % Compute autocorrelation for two time slices
    [R1_unipolar, R1_polar_nrz, R1_polar_rz, tau_vec] = ...
        compute_time_autocorr(Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted, t1);

    [R2_unipolar, R2_polar_nrz, R2_polar_rz, ~] = ...
        compute_time_autocorr(Unipolar_Shifted, PolarNRZ_Shifted, PolarRZ_Shifted, t2);

    % Symmetric tau range (include negative lags)
    tau_full = [-flip(tau_vec(2:end)), tau_vec];

    % Symmetric autocorrelations
    R1_unipolar_full = [fliplr(R1_unipolar(2:end)), R1_unipolar];
    R2_unipolar_full = [fliplr(R2_unipolar(2:end)), R2_unipolar];

    R1_polar_nrz_full = [fliplr(R1_polar_nrz(2:end)), R1_polar_nrz];
    R2_polar_nrz_full = [fliplr(R2_polar_nrz(2:end)), R2_polar_nrz];

    R1_polar_rz_full = [fliplr(R1_polar_rz(2:end)), R1_polar_rz];
    R2_polar_rz_full = [fliplr(R2_polar_rz(2:end)), R2_polar_rz];

    % Average of the two realizations
    R_avg_unipolar = 0.5 * (R1_unipolar_full + R2_unipolar_full);
    R_avg_polar_nrz = 0.5 * (R1_polar_nrz_full + R2_polar_nrz_full);
    R_avg_polar_rz = 0.5 * (R1_polar_rz_full + R2_polar_rz_full);
end


function plot_combined_autocorrelation(R_unipolar, R_polarNRZ, R_polarRZ, ...
                                       tau_vec, max_lag, Tb, A, ...
                                       Unipolar_AutoCorr, PolarNRZ_AutoCorr, PolarRZ_AutoCorr)
    % Combined plot of time-domain and statistical autocorrelations
    %
    % Inputs:
    %   R_unipolar, R_polarNRZ, R_polarRZ - Time autocorrelation matrices (1 realization)
    %   tau_vec - Non-negative tau vector
    %   max_lag - Max lag for axis limits
    %   Tb - Bit duration
    %   A - Amplitude
    %   *_AutoCorr - Statistical autocorrelation vectors

    x_limit = max_lag / 10;

    % Full symmetric tau vector
    tau_full = [-flip(tau_vec(2:end)), tau_vec];
    tau_sec  = tau_full * (Tb / 7);  % seconds

    % Reconstruct full symmetric time-domain autocorrelation
    R_unipolar_full = [fliplr(R_unipolar(1,2:end)), R_unipolar(1,:)];
    R_polarNRZ_full = [fliplr(R_polarNRZ(1,2:end)), R_polarNRZ(1,:)];
    R_polarRZ_full  = [fliplr(R_polarRZ(1,2:end)),  R_polarRZ(1,:)];

    % Theoretical autocorrelation
    Rx_Unipolar = (abs(tau_sec) < Tb) .* ((A^2 / 2) .* (1 - (abs(tau_sec) / (2*Tb)))) + ...
                  (abs(tau_sec) >= Tb) .* (A^2 / 4);
```

```
    Rx_PolarNRZ = (abs(tau_sec) < Tb) .* (A^2 .* (1 - abs(tau_sec) / Tb));
    Rx_PolarRZ  = (abs(tau_sec) < Tb/2) .* ((4/7) * A^2 .* (1 - (8 * abs(tau_sec) ./ (7 * Tb))));

    % Statistical lag axis
    t = -max_lag:max_lag;
    tau = abs(t);

    % Theoretical for statistical view
    Rx_Unipolar_stat = (tau < Tb) .* ((A^2 / 2) .* (1 - ( tau / (2*Tb)))) + ...
                       (tau >= Tb) .* (A^2 / 4);
    Rx_PolarNRZ_stat = (tau < Tb) .* (A^2 .* (1 - (tau / Tb)));
    Rx_PolarRZ_stat  = (tau < Tb/2) .* ((4/7) * A^2 .* (1 - (8 * tau ./ (7 * Tb))));

    figure('Name', 'Autocorrelation: Time-Domain & Statistical');

       % ---- Comparison: Time vs Statistical Autocorrelation ----
    figure('Name', 'Comparison: Time vs Statistical Autocorrelations');

    % Unipolar NRZ
    subplot(3, 1, 1);
    plot(tau_full, R_unipolar_full, 'b', 'LineWidth', 1.5); hold on;
    plot(t, Unipolar_AutoCorr, '--r', 'LineWidth', 1.5);
    xlim([-x_limit, x_limit]);
    title('Unipolar NRZ: Time vs Statistical');
    xlabel('Lag'); ylabel('Autocorrelation');
    legend('Time-domain', 'Statistical'); grid on;

    % Match the overlapping segment
    min_len = min(length(R_unipolar_full), length(Unipolar_AutoCorr));
    R_trimmed = R_unipolar_full(1:min_len);
    AutoCorr_trimmed = Unipolar_AutoCorr(1:min_len);

    % Now compare
    disp("The Statical and Time differnece is");
    disp(norm(R_trimmed - AutoCorr_trimmed));


    % Polar NRZ
    subplot(3, 1, 2);
    plot(tau_full, R_polarNRZ_full, 'b', 'LineWidth', 1.5); hold on;
    plot(t, PolarNRZ_AutoCorr, '--r', 'LineWidth', 1.5);
    xlim([-x_limit, x_limit]);
    title('Polar NRZ: Time vs Statistical');
    xlabel('Lag'); ylabel('Autocorrelation');
    legend('Time-domain', 'Statistical'); grid on;

    % Polar RZ
    subplot(3, 1, 3);
    plot(tau_full, R_polarRZ_full, 'b', 'LineWidth', 1.5); hold on;
    plot(t, PolarRZ_AutoCorr, '--r', 'LineWidth', 1.5);
    xlim([-x_limit, x_limit]);
    title('Polar RZ: Time vs Statistical');
    xlabel('Lag'); ylabel('Autocorrelation');
    legend('Time-domain', 'Statistical'); grid on;

end

function [PSD_unipolar ,PSD_polarNRZ ,PSD_polarRZ] =...
    plot_linecode_psd(R_Unipolar, R_PolarNRZ, R_PolarRZ, fs, A, Tb)
    % Function to plot PSDs for Unipolar NRZ, Polar NRZ, and Polar RZ line codes
    % using FFT of average autocorrelation sequences and compare with theoretical PSDs.
    %
    % Inputs:
    %   R_Unipolar   - Average autocorrelation of Unipolar NRZ
    %   R_PolarNRZ   - Average autocorrelation of Polar NRZ
    %   R_PolarRZ    - Average autocorrelation of Polar RZ
    %   fs           - Sampling frequency in Hz
    %   A            - Amplitude of the signal
    %   Tb           - Bit period (duration of one bit)

    % Ensure all inputs are column vectors
    R_Unipolar = R_Unipolar(:);
    R_PolarNRZ = R_PolarNRZ(:);
    R_PolarRZ = R_PolarRZ(:);

    % Number of samples
    n = length(R_Unipolar);
    % Index of center (DC component after fftshift)
    center_idx = ceil(n / 2);

    % Remove DC Pulse
```

```
    mu_uni = mean(R_Unipolar(end-10:end));  % Use tail values
    R_Unipolar = R_Unipolar - mu_uni;            % Remove DC

    % Compute FFTs of autocorrelations
    fft_unipolar = fft(R_Unipolar) / n;
    fft_polarNRZ = fft(R_PolarNRZ) / n;
    fft_polarRZ  = fft(R_PolarRZ) / n;

    % Compute PSD magnitudes
    PSD_unipolar = abs(fft_unipolar);
    PSD_polarNRZ = abs(fft_polarNRZ);
    PSD_polarRZ  = abs(fft_polarRZ);

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

    % Center the FFTs for proper plotting
    PSD_unipolar = A*fftshift(PSD_unipolar);
    PSD_polarNRZ = A*fftshift(PSD_polarNRZ);
    PSD_polarRZ  = A*fftshift(PSD_polarRZ);

    % Compute theoretical PSDs
    S_unipolar_nrz = (A^2 * Tb / 4) * (sin(pi * freq_axis * Tb) ./ (pi * freq_axis * Tb)).^2;
    S_polar_nrz = (A^2 * Tb / 2) * (sin(pi * freq_axis * Tb / 2) ./ (pi * freq_axis * Tb / 2)).^2;
    S_polar_rz = (A^2 * Tb / 4) * (sin(pi * freq_axis * Tb / 4) ./ (pi * freq_axis * Tb / 4)).^2;

    % Handle zero frequency case (Dirac delta at f = 0)
    S_unipolar_nrz(freq_axis == 0) = A^2 / 4;
    S_polar_nrz(freq_axis == 0) = 0;  % No delta for Polar NRZ
    S_polar_rz(freq_axis == 0) = 0;  % No delta for Polar RZ

    % Plot PSDs
    figure('Name', 'Power Spectral Density via Average Autocorrelation');

    % Plot Unipolar NRZ
    subplot(3,1,1);
    plot(freq_axis, PSD_unipolar, 'b', 'LineWidth', 1.5);
    hold on;
    plot(freq_axis, S_unipolar_nrz, 'r--', 'LineWidth', 1.5);  % Theoretical PSD
    hold off;
    grid on;
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title('PSD of Unipolar NRZ');
    xlim([-fs/2, fs/2]);
    ylim([0, max(PSD_unipolar)*1.1]);
    legend('Empirical', 'Theoretical');

    % Plot Polar NRZ
    subplot(3,1,2);
    plot(freq_axis, PSD_polarNRZ, 'r', 'LineWidth', 1.5);
    hold on;
    plot(freq_axis, S_polar_nrz, 'g--', 'LineWidth', 1.5);  % Theoretical PSD
    hold off;
    grid on;
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title('PSD of Polar NRZ');
    xlim([-fs/2, fs/2]);
    ylim([0, max(PSD_polarNRZ)*1.1]);
    legend('Empirical', 'Theoretical');

    % Plot Polar RZ
    subplot(3,1,3);
    plot(freq_axis, PSD_polarRZ, 'g', 'LineWidth', 1.5);
    hold on;
    plot(freq_axis, S_polar_rz, 'b--', 'LineWidth', 1.5);  % Theoretical PSD
    hold off;
    grid on;
    xlabel('Frequency (Hz)');
    ylabel('Magnitude');
    title('PSD of Polar RZ');
    xlim([-fs/2, fs/2]);
    ylim([0, max(PSD_polarRZ)*1.1]);
    legend('Empirical', 'Theoretical');
end
```