**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**

# **Contents**

[A. **Contents** 2](#_Toc195632278)

[**B.** **Table of Figures** 3](#_Toc195632279)

[**C.** **Role of Each Member** 4](#_Toc195632280)

[**D.** **Project Description** 5](#_Toc195632281)

[**E.** **Introduction** 5](#_Toc195632282)

[**F.** **Control Flags** 5](#_Toc195632283)

[**G.** **Generation of Data** 6](#_Toc195632284)

[**H.** **polar NRZ ensemble creation** 6](#_Toc195632285)

[**I.** **Uni polar NRZ ensemble creation** 7](#_Toc195632286)

[**J.** **polarRZ ensemble creation** 8](#_Toc195632287)

[**K.** **Random initial time shift** 9](#_Toc195632288)

[**L.** **Getting cell arrays ready to calculate the statistical mean and autocorrelation:** 10](#_Toc195632289)

[**M.** **Questions** 12](#_Toc195632290)

[**1.** **Statistical Mean** 12](#_Toc195632291)

[**1.1.** **Hand Analysis** 12](#_Toc195632292)

[**1.2.**  **Code Snippet** 12](#_Toc195632293)

[**1.3.** **Plotting the Statistical Mean:** 13](#_Toc195632294)

[**2.** **Statistical Autocorrelation** 14](#_Toc195632295)

[**2.1.** **Hand Analysis** 14](#_Toc195632296)

[**2.2.** **Code Snippet** 15](#_Toc195632297)

[**2.3.** **Plotting the statistical autocorrelation** 16](#_Toc195632298)

[**3.** **Is the Process Stationary** 17](#_Toc195632299)

[**4.** **The time mean and autocorrelation function for one waveform** 18](#_Toc195632300)

[**4.1.** **Average Time Mean** 18](#_Toc195632301)

[**4.3.** **Average Time Auto Correction** 21](#_Toc195632302)

[**5.** **Is The Random Process Ergodic** 23](#_Toc195632303)

[**6.** **the PSD & Bandwidth of the Ensemble** 24](#_Toc195632304)

[**N.** **Appendix** 26](#_Toc195632305)

# **Table of Figures**

[Figure 1 Rx and Tx path 5](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632306)

[Figure 2 ADC Binary Output 6](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632307)

[Figure 3 PolarNRZ Realizations 7](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632308)

[Figure 4 Uni Polar Realizations 8](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632309)

[Figure 5 PolarRZ Realization 9](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632310)

[Figure 6 Realization Shifted 10](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632311)

[Figure 7 Plot of Statistical Mean 13](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632312)

[Figure 8 Auto Correction plot 16](#_Toc195632313)

[Figure 9 autocorrelation at two different times 17](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632314)

[Figure 10 Time Mean for Uni Polar 18](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632315)

[Figure 11 Time Mean for Polar NRZ 19](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632316)

[Figure 12 Time Mean for Polar RZ 19](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632317)

[Figure 13 Time Mean Vs Realization 20](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632318)

[Figure 14 Time Auto Correlation 21](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632319)

[Figure 15 Comparison Between Time and Statistical Mean 23](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632320)

[Figure 16 PSD plot of the Ensemble 24](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\Communication%20Project%20-%201.docx#_Toc195632321)

# **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 |

# **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).

# **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.

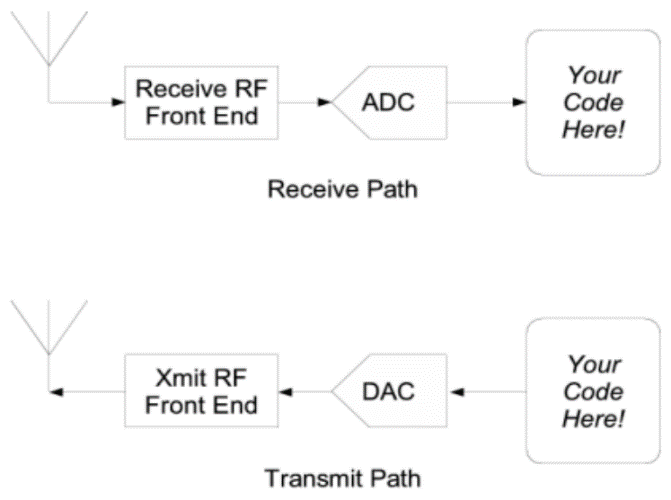


Figure 1 Rx and Tx path

# **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 |

# **Generation of Data**



Using the function: “**Randi**” to generate random binary data of size 500x101

(500 waveforms each with 101 bits). This data represents the binary bits that need to be encoded.

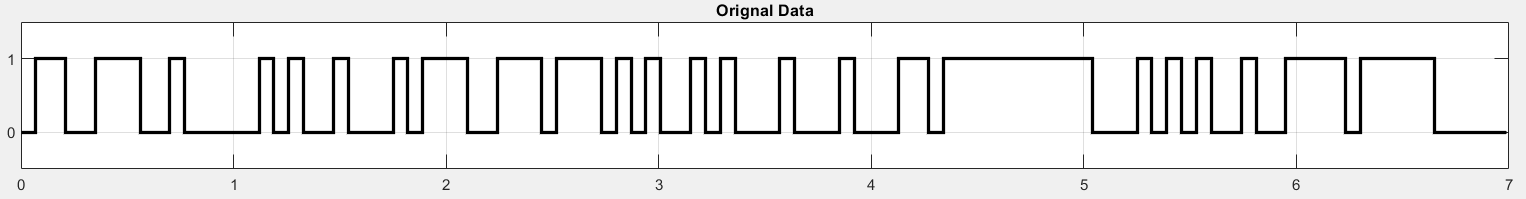
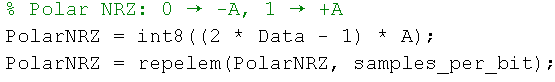


Figure 2 ADC Binary Output

For the line codes we will use this function:

# **polar NRZ ensemble creation**



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

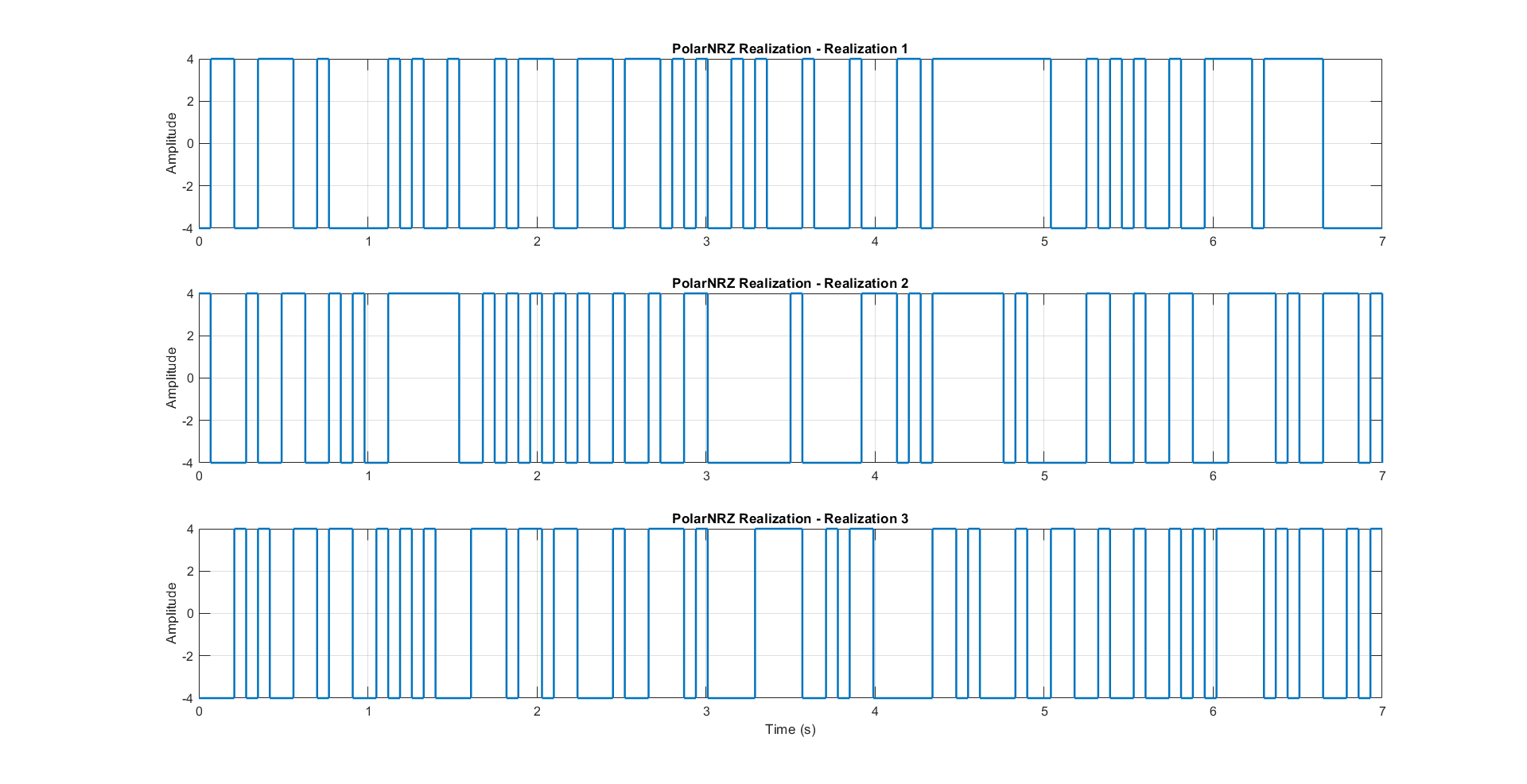


Figure 3 PolarNRZ Realizations

# **Uni polar NRZ ensemble creation**

* We then generate unipolar NRZ amplitudes along with its realization.
* We convert data (1,0) to 1 →A ,0→ 0 to have uni\_polar\_NRZ.

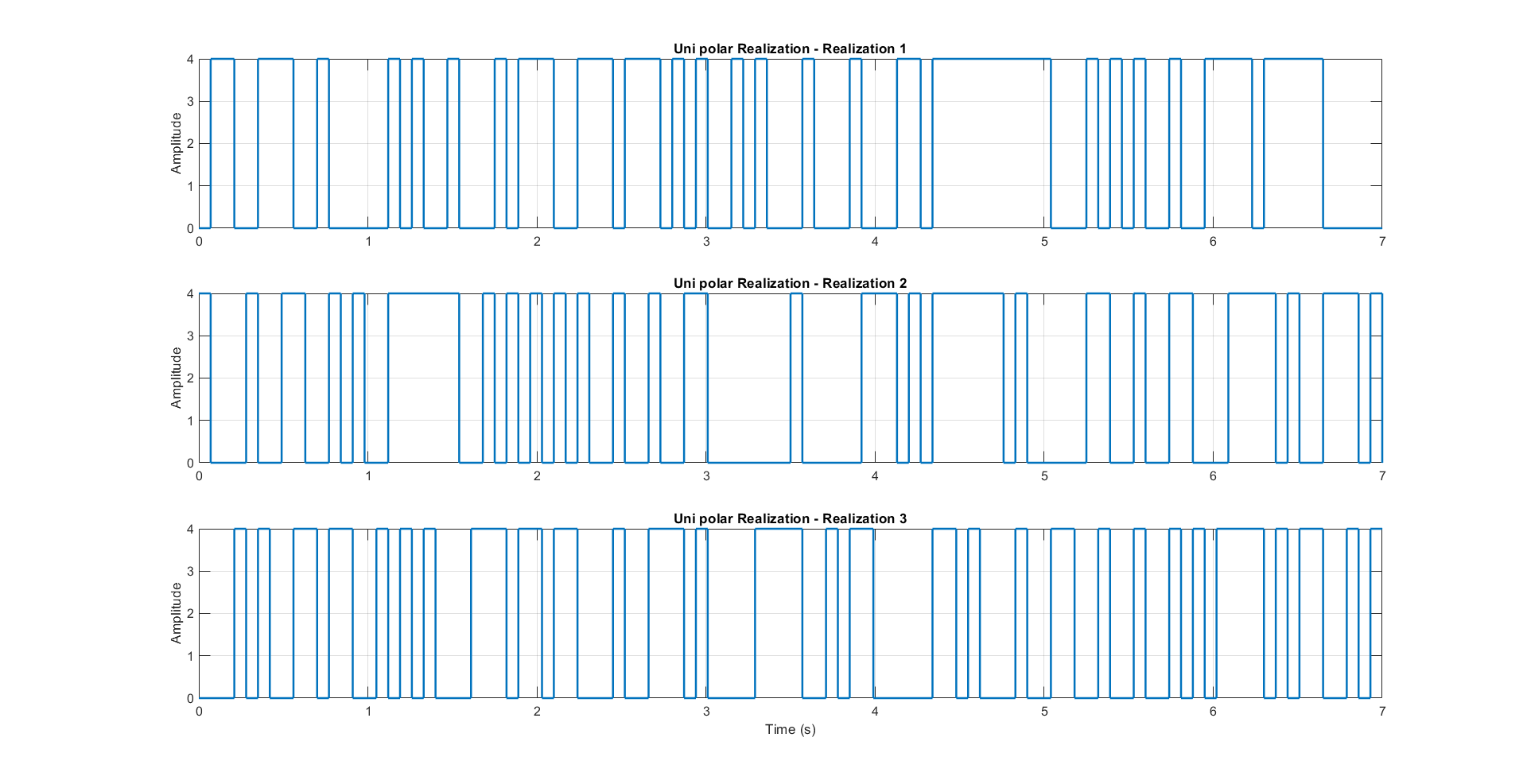
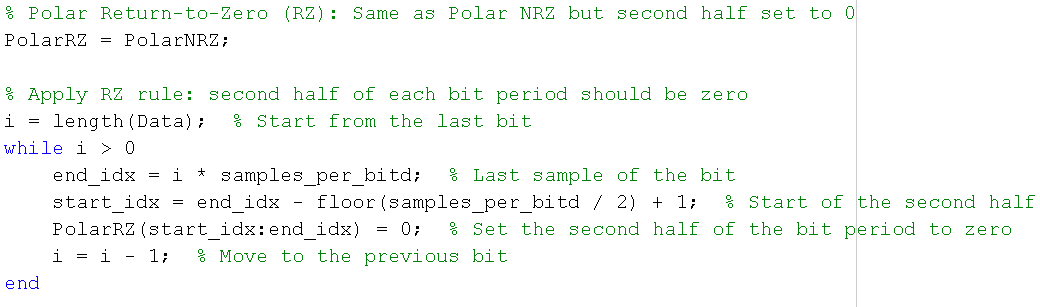


Figure 4 Uni Polar Realizations

# **polarRZ ensemble creation**



* 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.

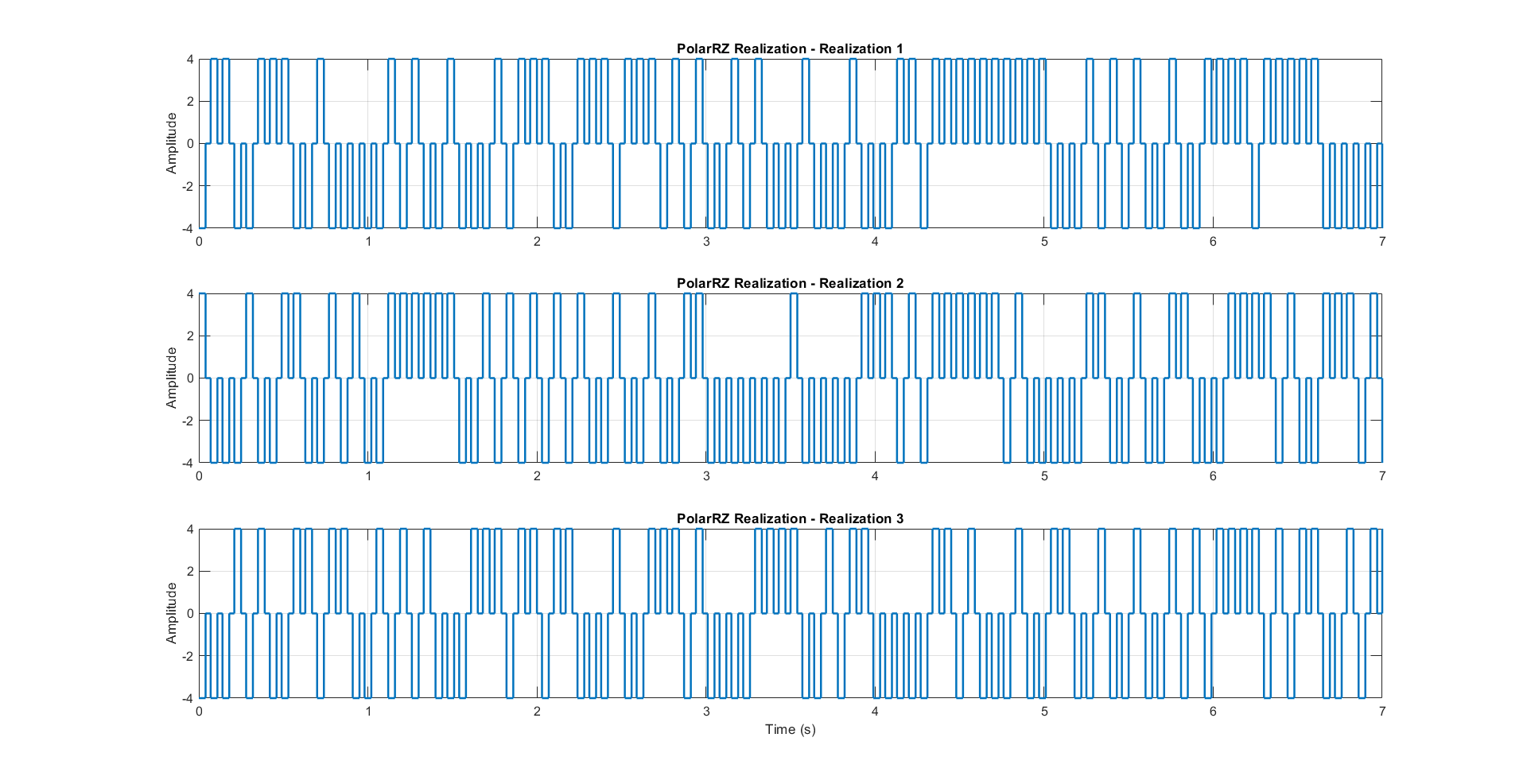
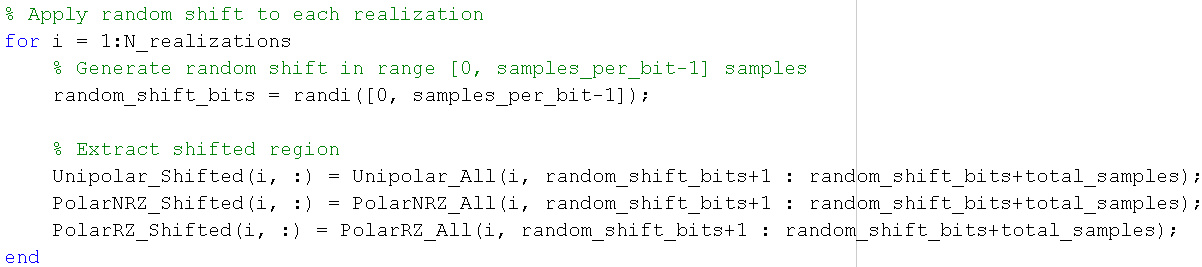


Figure 5 PolarRZ Realization

# **Random initial time shift**

For the random shift we made this function:

* 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).

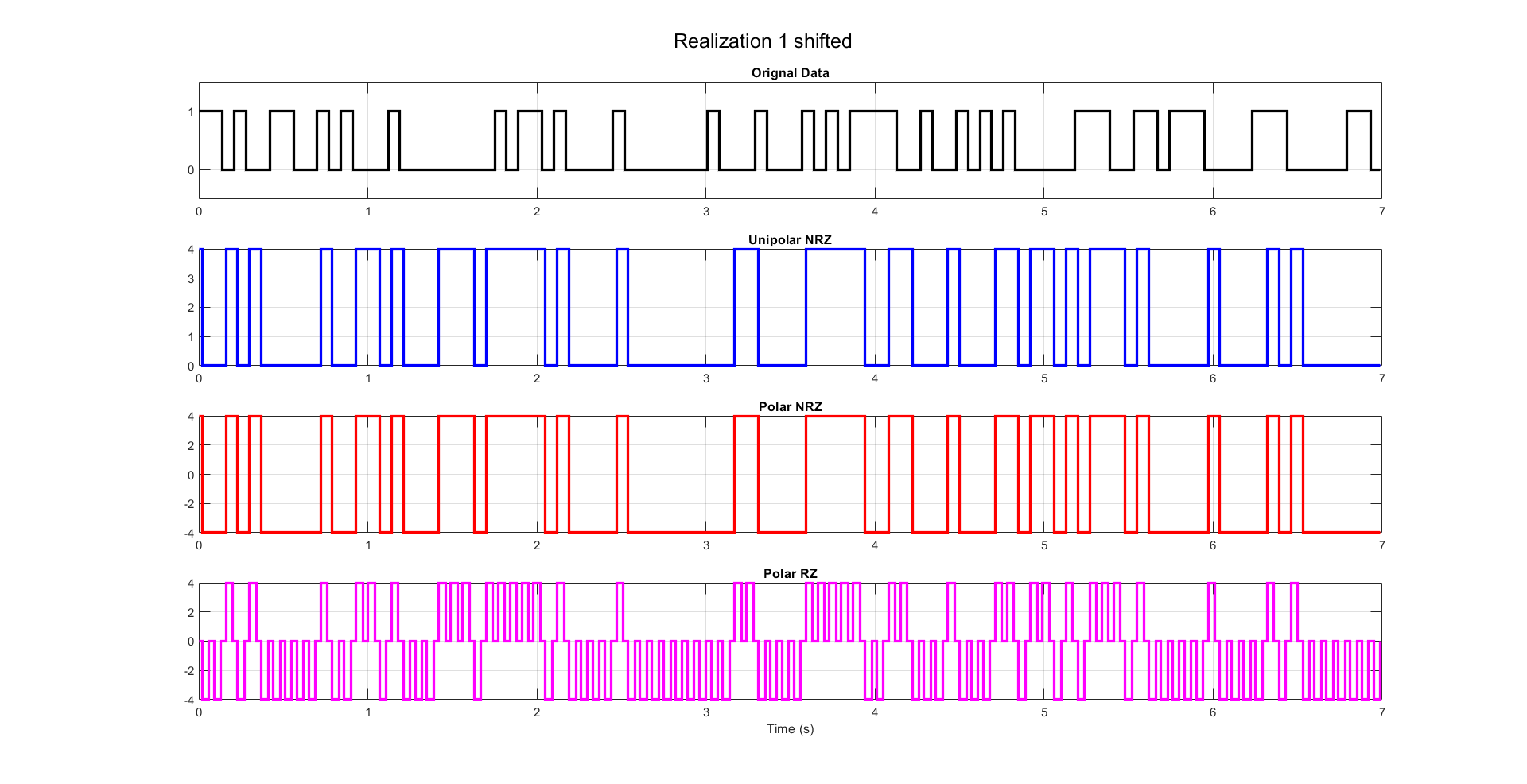
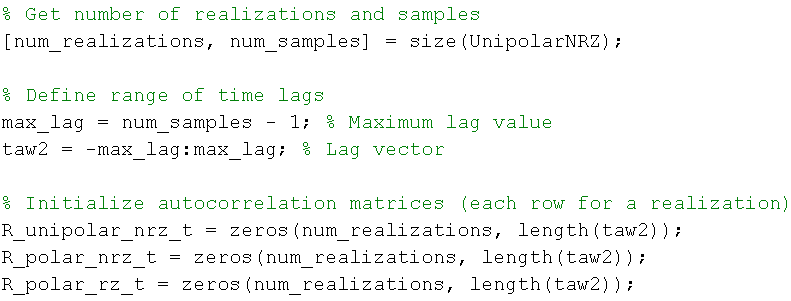


Figure 6 Realization Shifted

# **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:

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



# **Questions**

## **Statistical Mean**

### **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”

𝜇𝑋(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:

𝜇𝑋\_𝑃𝑁𝑅𝑍(𝑡) = 4 ∗ 0.5 + (−4) ∗ 0.5 = 0 (𝐶𝑜𝑛𝑠𝑡𝑎𝑛𝑡 𝑎𝑐𝑟𝑜𝑠𝑠 𝑡𝑖𝑚𝑒).

𝜇𝑋\_𝑃𝑅𝑍(𝑡) = 4 ∗ 0.5 + (−4) ∗ 0.5 = 0 (𝐶𝑜𝑛𝑠𝑡𝑎𝑛𝑡 𝑎𝑐𝑟𝑜𝑠𝑠 𝑡𝑖𝑚𝑒).

### **Code Snippet**

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

### **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.

## **Statistical Autocorrelation**

### **Hand Analysis**

𝑹𝑿(𝝉) = 𝑬[𝑿(𝝉) 𝑿(𝒕 + 𝝉)] = ∑ 𝑿(𝝉) 𝑿(𝒕 + 𝝉) 𝑷(𝑿(𝝉) 𝑿(𝒕 + 𝝉))

* + **For Unipolar NRZ:**

We have 2 cases **(Considering T to be 70ms or 7 samples),**

1. |𝝉| < 𝑻

𝑅𝑋(𝜏) = 𝐸[𝑋(𝜏) 𝑋(𝑡 + 𝜏)]

= 42 ∗ 𝑃(4,4) + 02 ∗ 𝑃(0,0) + 4 ∗ 0 ∗ 𝑃(0,4) + 0 ∗ 4 ∗ 𝑃(4,0)

= 42 ∗ 𝑃(4,4)

𝑃(4,4) = 𝑃(𝑋(𝑡 + 𝜏) = 4 | 𝑋(𝑡) = 4) ∗ 𝑃(𝑋(𝑡) = 4)

𝑃(𝑋(𝑡 + 𝜏) = 4 | 𝑋(𝑡) = 4) = 𝑃(𝑇̅) + 𝑃(𝑇) ∗ 𝑃(𝑋(𝑡 + 𝜏) = 4)

1. |𝝉| > 𝑻

𝑅𝑋(𝜏) = 𝐸[𝑋(𝜏) 𝑋(𝑡 + 𝜏)]

= 42 ∗ 0.5 ∗ 0.5 + 02 ∗ 0.5 ∗ 0.5 + 4 ∗ 0 ∗ 0.5 ∗ 0.5 + 0 ∗ 4 ∗ 0.5 ∗ 0.5

= 42 ∗ 0.5 ∗ 0.5

= 4

* + And using the same flow, we can find that the ACF for “**Polar NRZ**” is
  + And similarly, the ACF for “**Polar RZ**” is

**And as we know:**

𝑇𝑜𝑡𝑎𝑙 𝑃𝑜𝑤𝑒𝑟 = 𝑅𝑋(0) & 𝐷𝐶 𝑃𝑜𝑤𝑒𝑟 = 𝑅𝑋(∞).

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** |

### **Code Snippet**

**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).

### **Plotting the statistical autocorrelation**

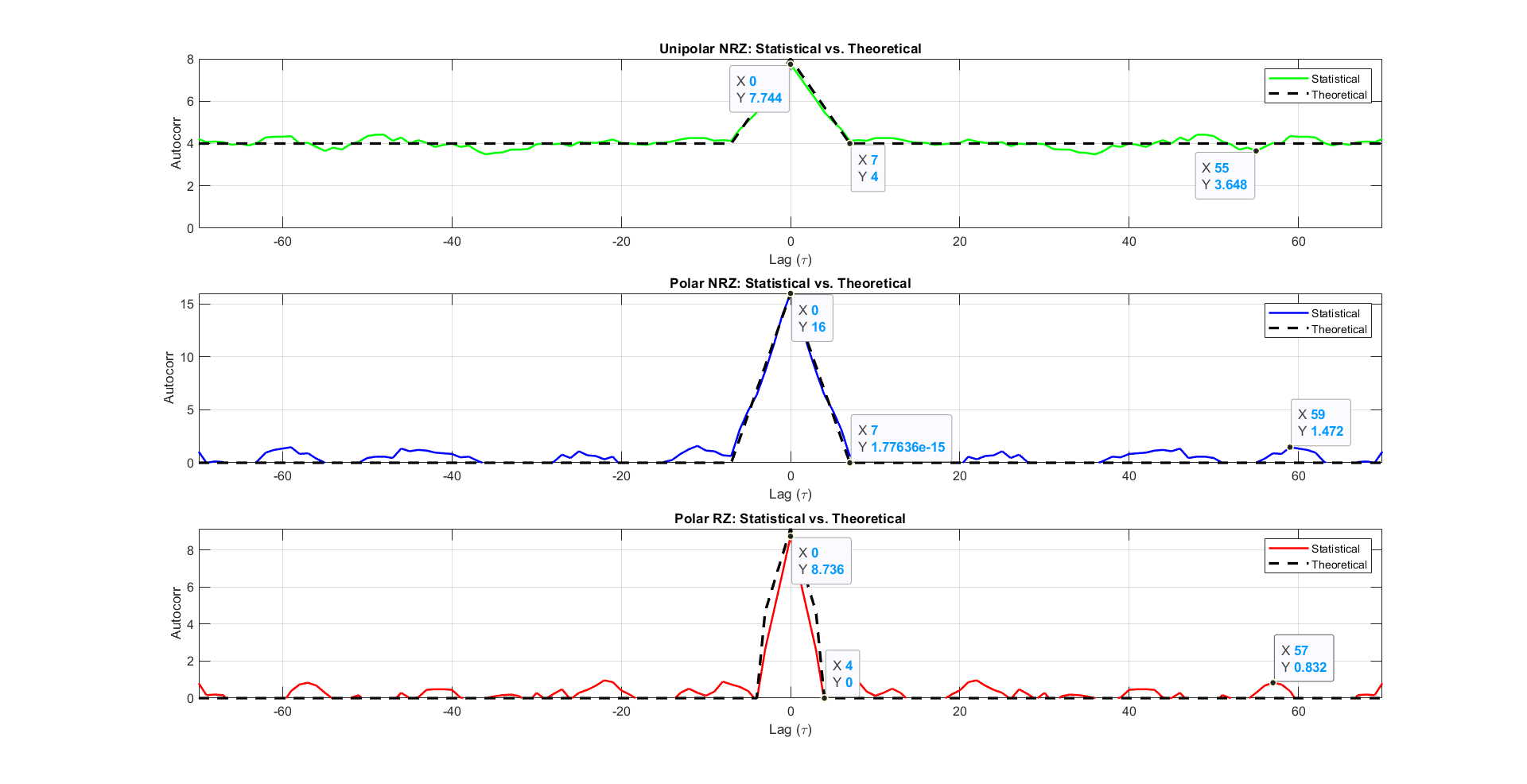
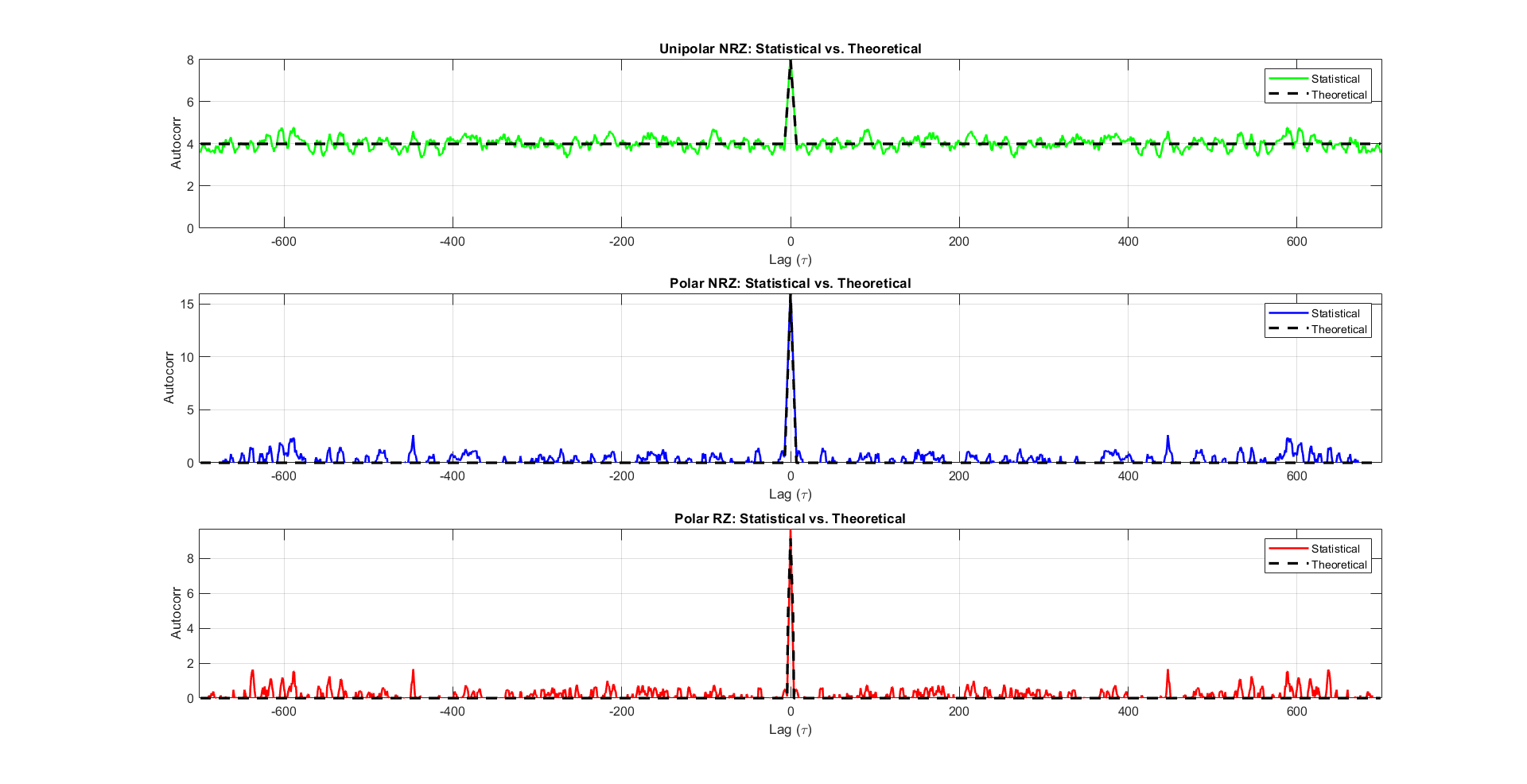


Figure 9 Statistical Auto Correction plot zoomed

Figure 8 Statistical Auto Correction plot



The resulting autocorrelation values are plotted against the corresponding time delays (τ). We observe that at τ = 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.

## **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).**

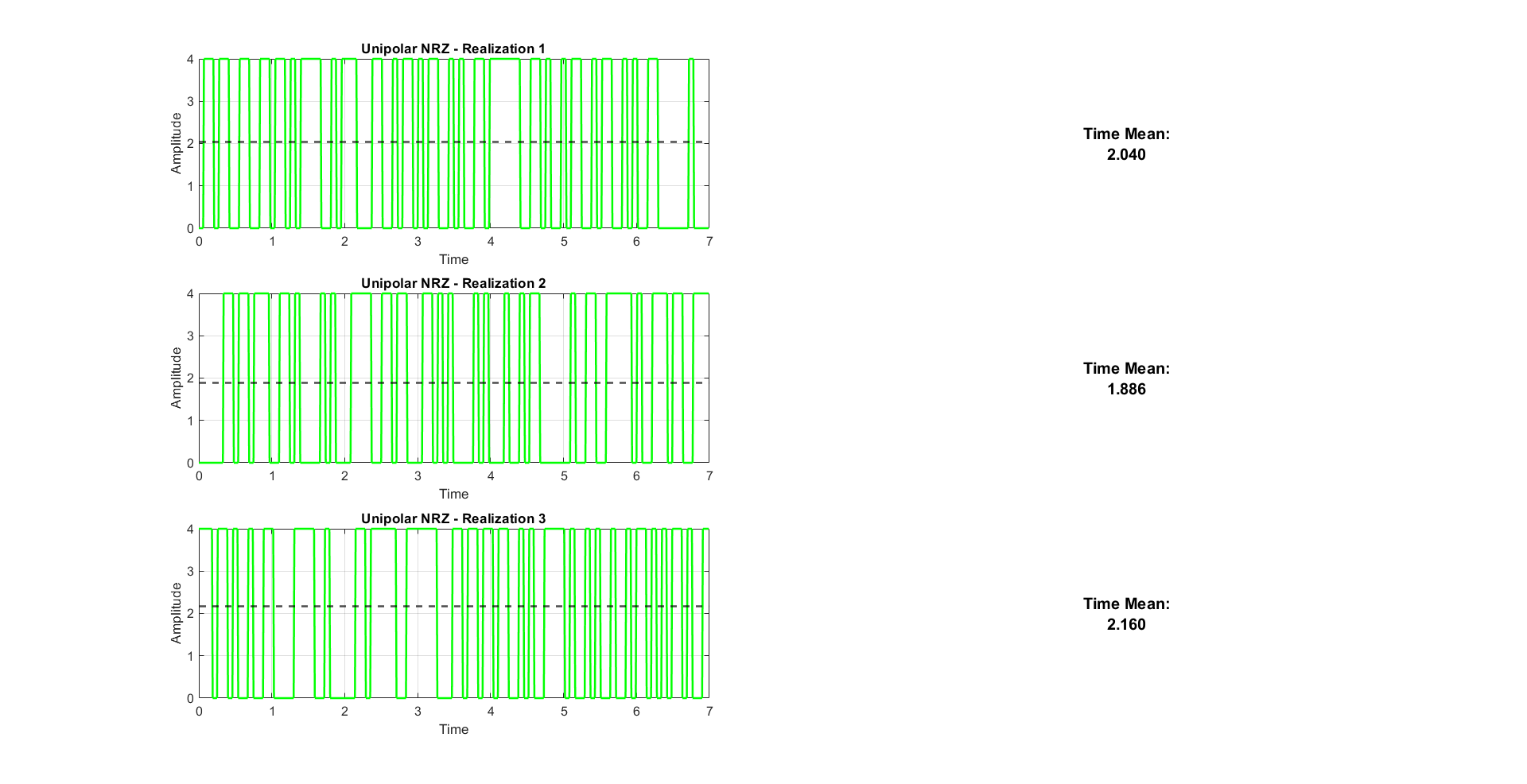
Yes, the process is stationary (WSSP) because the mean is constant function in time as shown in Figure 7 Plot of Statistical Mean and the autocorrelation depends only on the time difference not the absolute time.

## **The time mean and autocorrelation function for one waveform**

### **Time Mean**

* 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



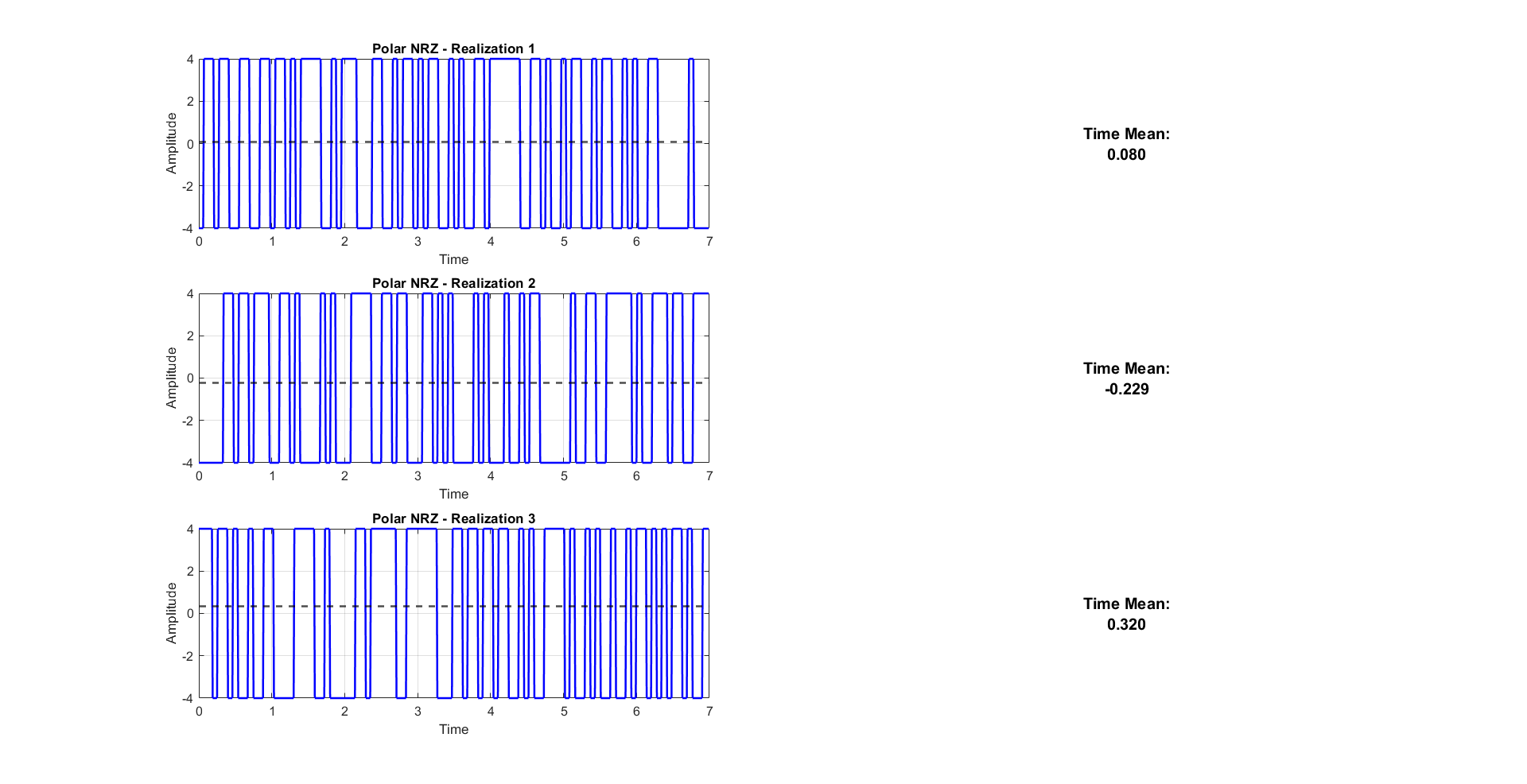
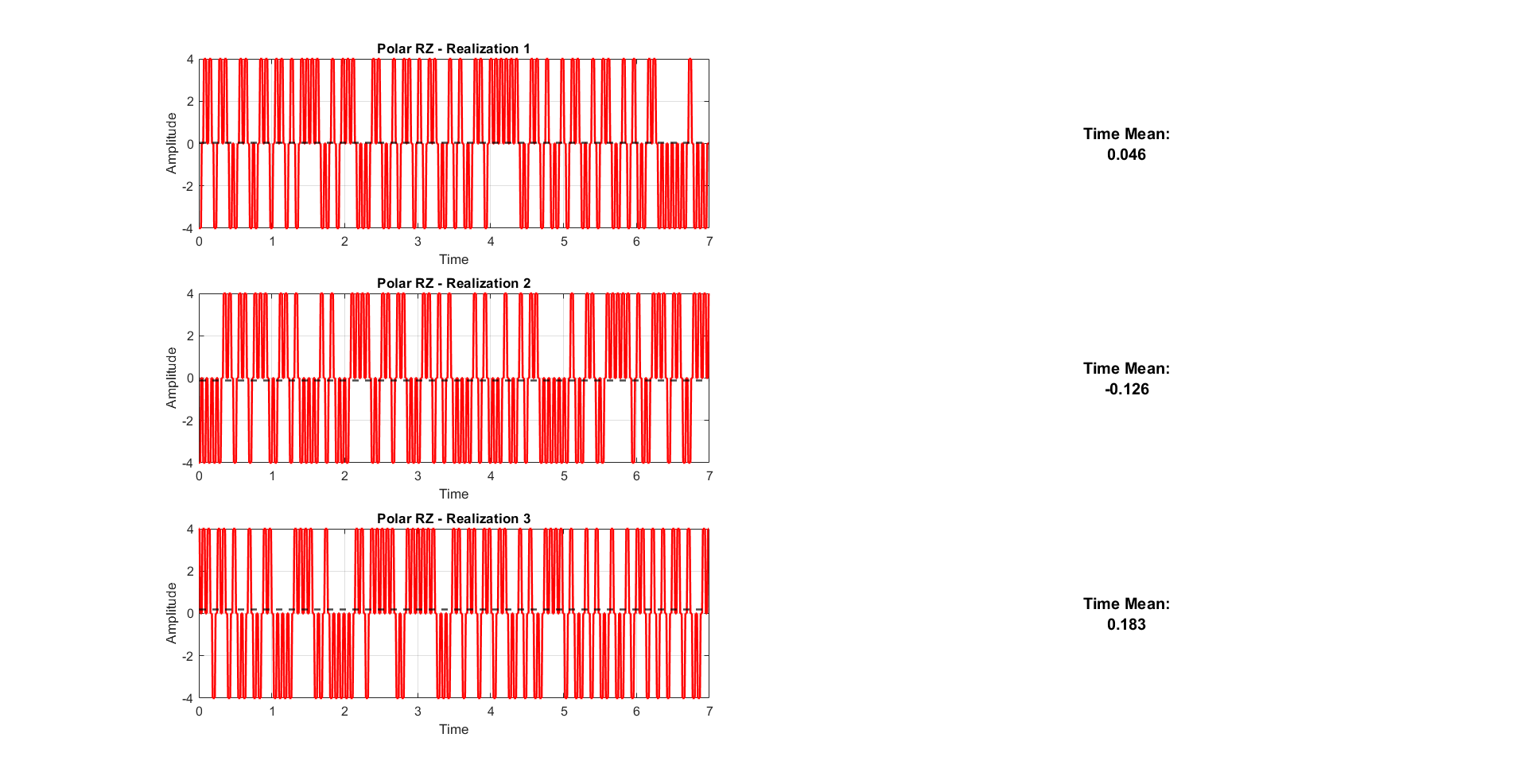
* As expected, polar RZ & NRZ have almost zero mean and the uni polar has mean around 2 Because its amplitude ranges from 0 : 4

Figure 12 Time Mean for Polar NRZ

Figure 13 Time Mean for Polar RZ

* 1. **Time Mean Vs Realization**

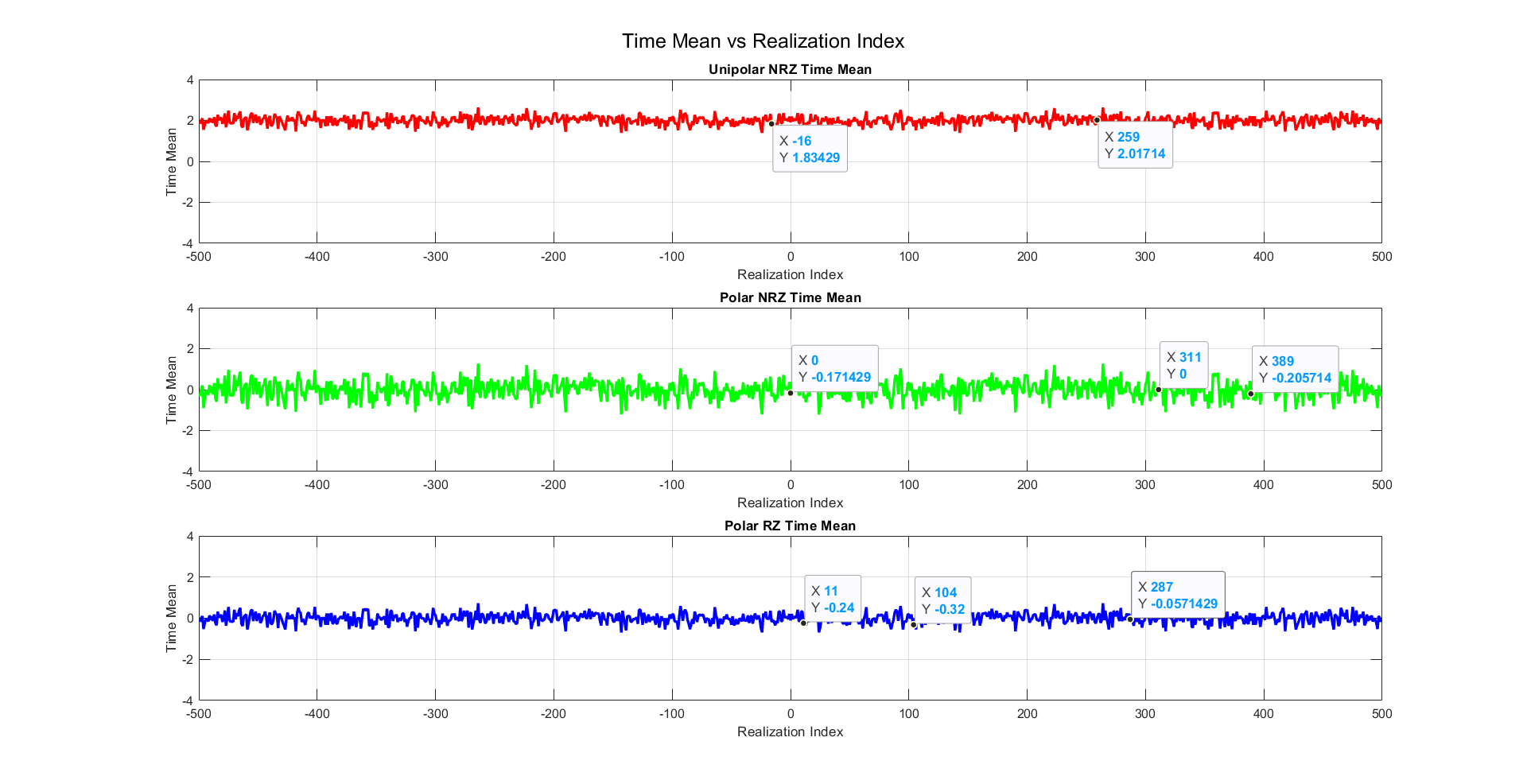
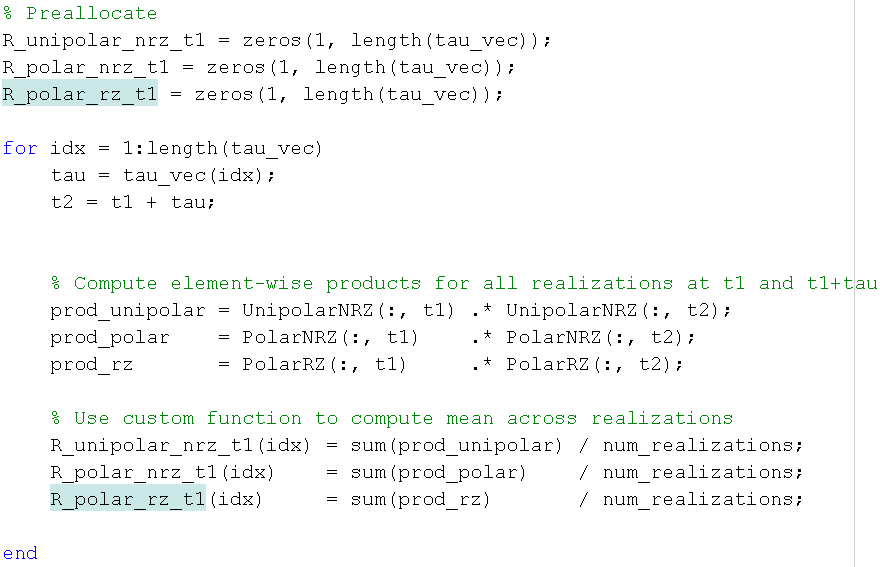


Figure 14 Time Mean Vs Realization

* 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.

### **Time Auto Correlation**

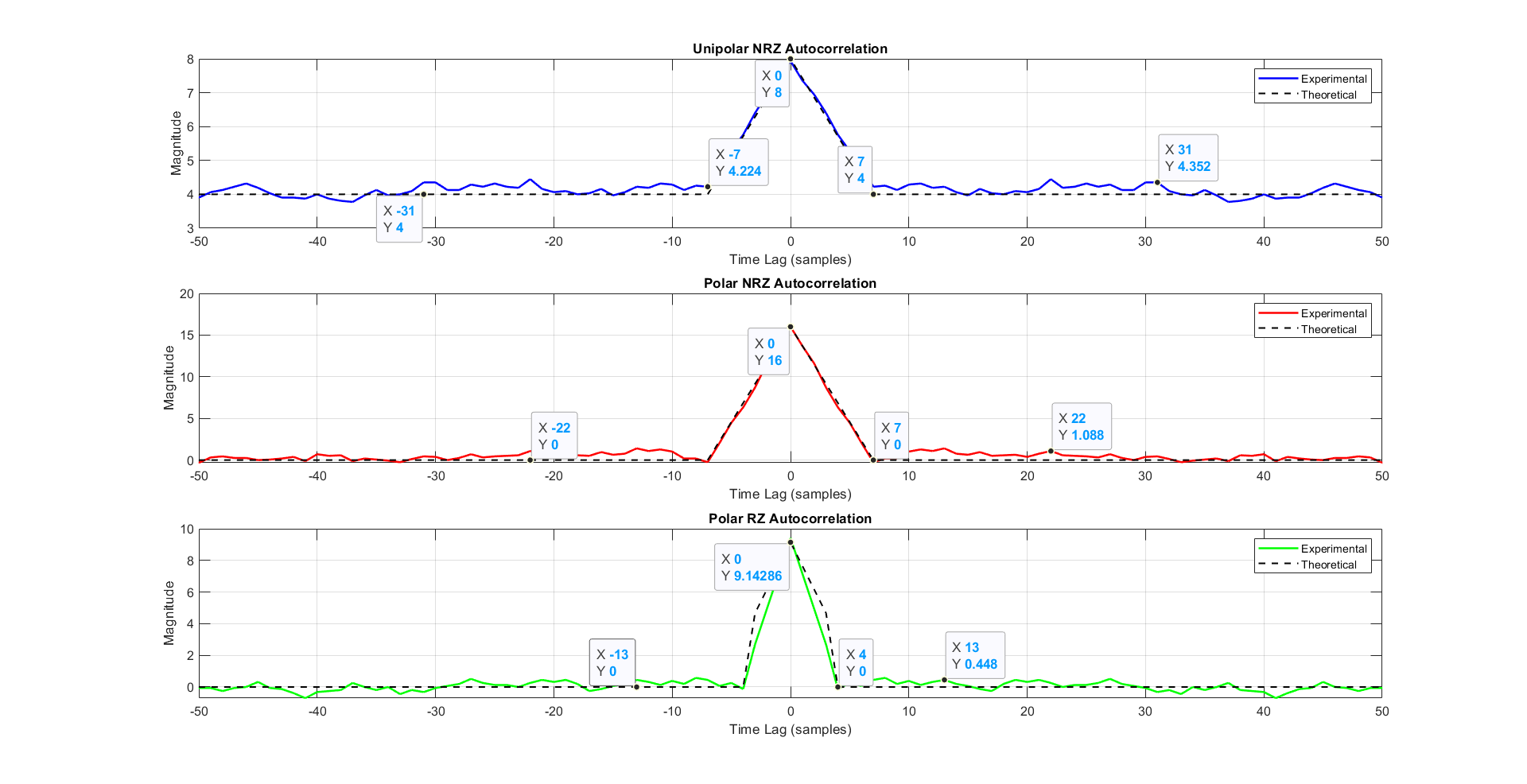
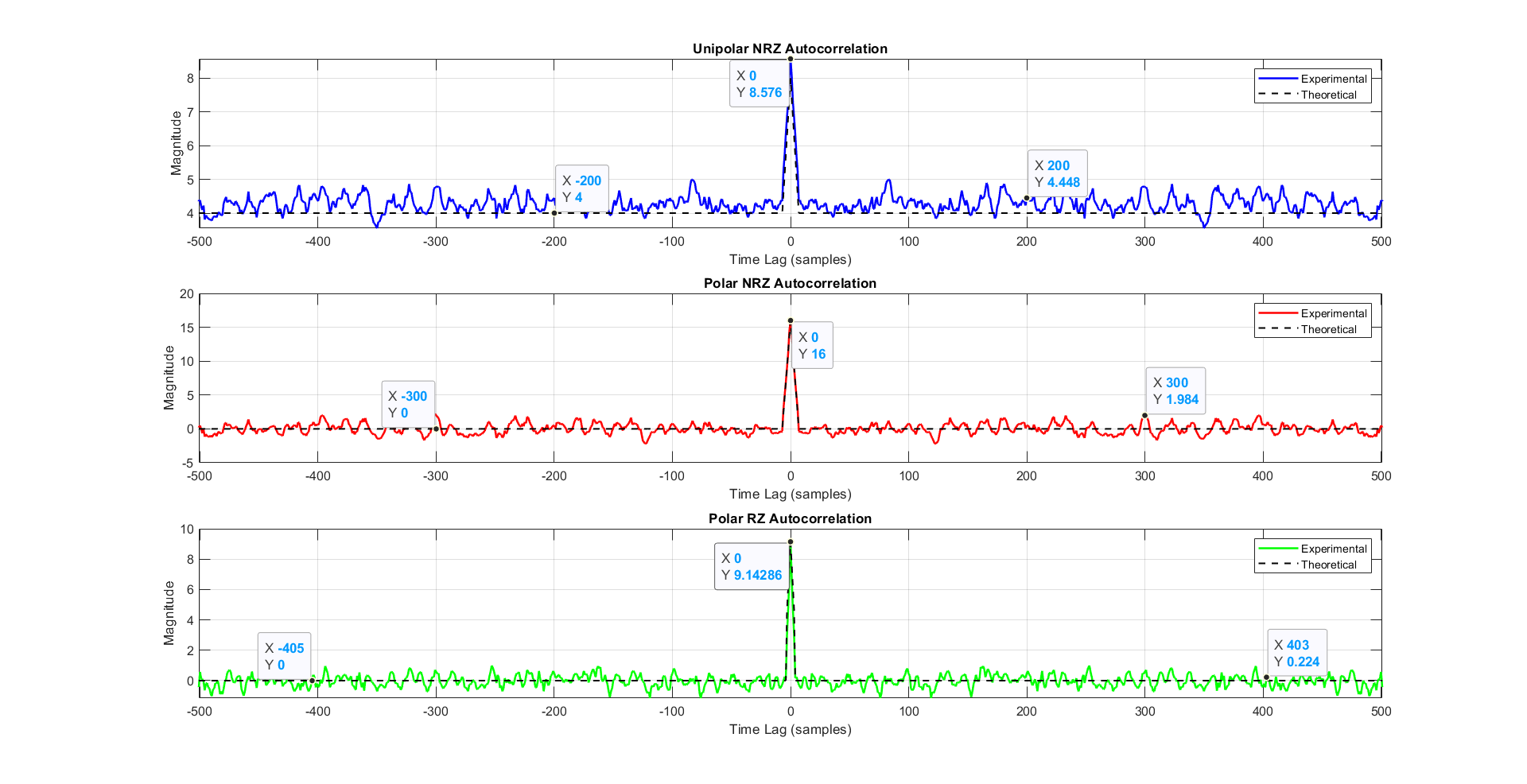
For the time Auto Correlation we’re going to use this function



### **Time Auto Correlation for one wave form:**

Figure 15 Time Auto Correction plot zoomed

Figure 16 Time Auto Correction plot



As shown in the graphs:

* The time autocorrelation is closely same as the ensemble  
  autocorrelation.
* The autocorrelation function has maximum at 𝛕 = 𝟎 and it is an even function.

## **Is The Random Process Ergodic?**

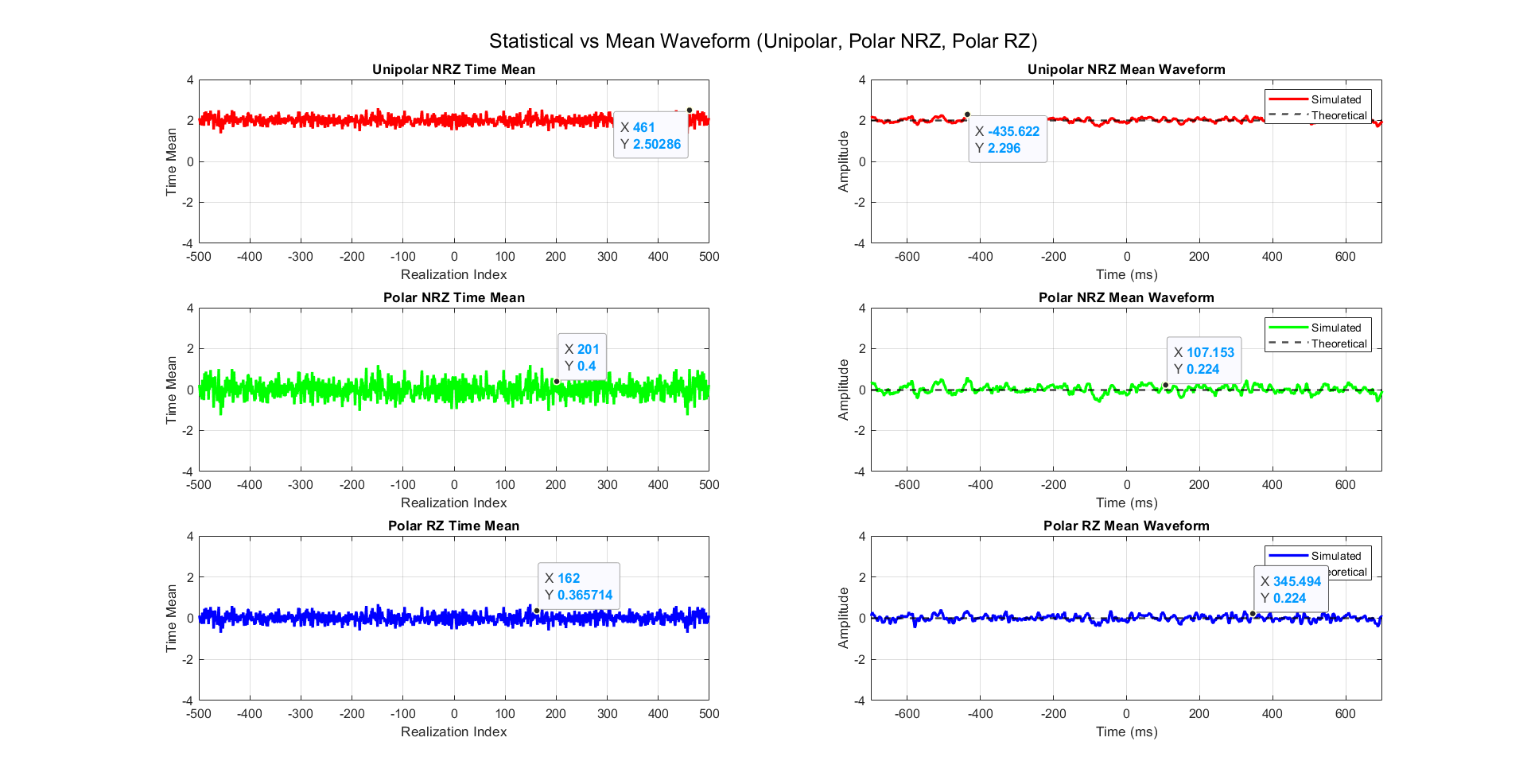


Figure 17 Time Mean vs Statistical

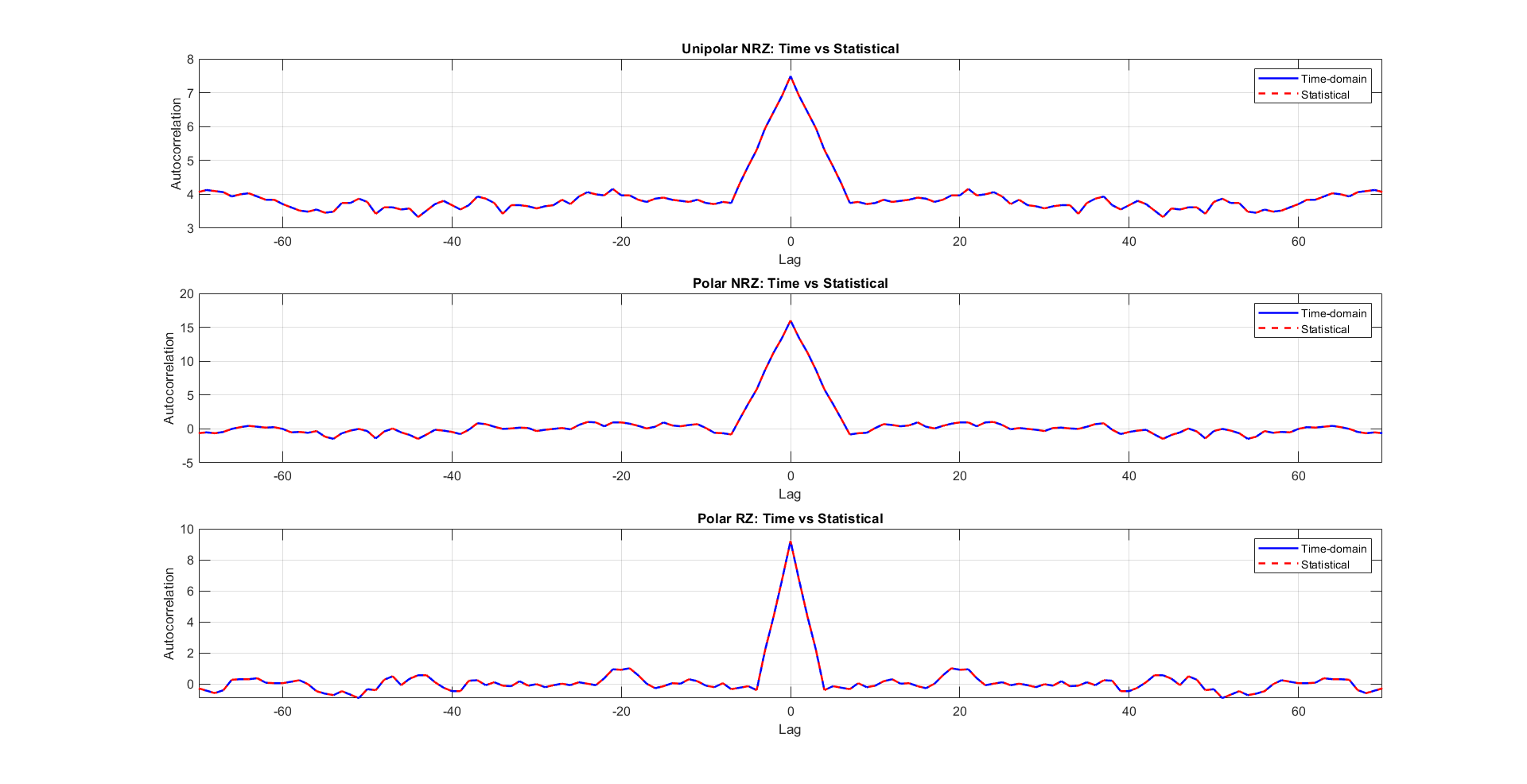


Figure 18 Time Auto Correlation Vs Statistical

* 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



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**

## **the PSD & Bandwidth of the Ensemble**

function BW = find\_bandwidth(PSD, freq\_axis)

% Function to find the -3dB bandwidth from PSD

PSD\_max = max(PSD); % Find max power

threshold = PSD\_max / 2; % -3dB threshold (half power)

% Find indices where PSD is above threshold

valid\_indices = find(PSD >= threshold);

% Check if we found any valid points

if isempty(valid\_indices)

BW = 0; % If no valid range is found, return 0

return;

end

* We take the Fourier transform of the statistical autocorrelation then centralize the graph around zero.
* since
* Then we normalize the PSD Amplitude through dividing by Fs

**For the BW**

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

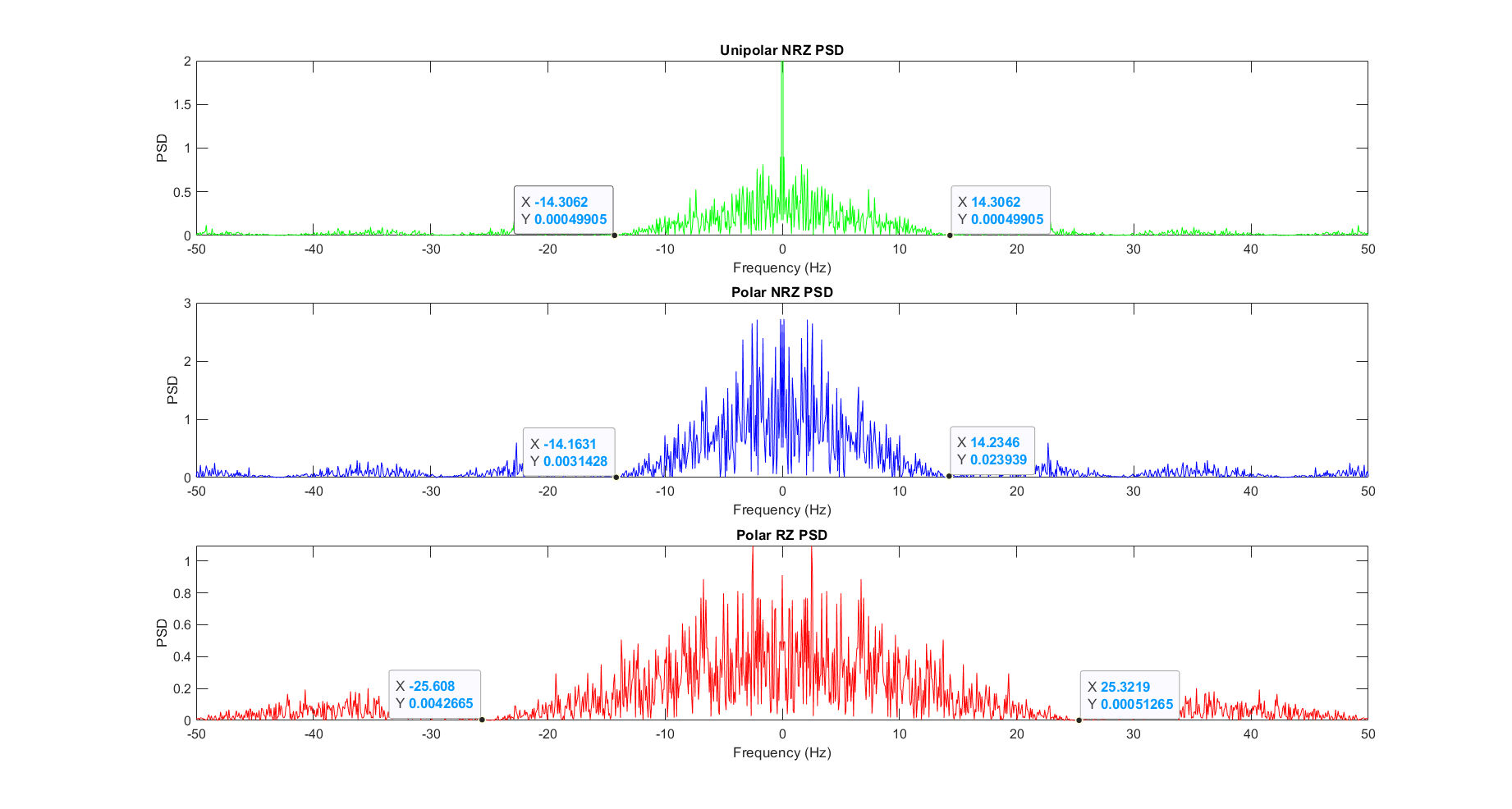


Figure 19 PSD plot of the Ensemble

**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 14.23 hz
* BW of the polar RZ is the double of bitrate which approximately equal 25.32 hz

# **Appendix**

clc;

clear;

close all;

% Parameters

A = 4; % Amplitude

N\_realizations = 500; % Number of waveforms (ensemble size)

num\_bits = 100+1; % Bits per waveform and one extra bit for shifting

bit\_duration = 70e-3; % 70 ms per bit

dac\_interval = 10e-3; % DAC updates every 10 ms

samples\_per\_bit = round(bit\_duration / dac\_interval); % 7 samples per bit

total\_time = num\_bits \* bit\_duration; % Total waveform duration

t = 0:dac\_interval:(total\_time - dac\_interval); % Time vector

% Preallocate matrices for efficiency

Unipolar\_All = zeros(N\_realizations, length(t), 'int8');

PolarNRZ\_All = zeros(N\_realizations, length(t), 'int8');

PolarRZ\_All = zeros(N\_realizations, length(t), 'int8');

% Generate and store 500 realizations

for i = 1:N\_realizations

Data = randi([0, 1], 1, num\_bits, 'int8'); % Random bit sequence

%encode the data

[Unipolar, PolarNRZ, PolarRZ] = generate\_linecodes(Data, A, samples\_per\_bit);

% Store in matrices

Unipolar\_All(i,:) = Unipolar;

PolarNRZ\_All(i,:) = PolarNRZ;

PolarRZ\_All(i,:) = PolarRZ;

% Plot the first realization

if i==1

plot\_linecodes(Data, Unipolar, PolarNRZ, PolarRZ, t, num\_bits-1,'Realization 1');

end

end

% Plot the first 5 realizations as a sample

figure;

for i = 1:5

subplot(5,1,i);

stairs(t, Unipolar\_All(i,:), 'b', 'LineWidth', 1.5);

title(['Unipolar NRZ - Realization ' num2str(i)]);

grid on;

end

xlabel('Time (s)');

% Apply random shift to all realizations

[Unipolar\_Shifted, PolarNRZ\_Shifted, PolarRZ\_Shifted] =...

apply\_random\_shift\_fixed\_size(Unipolar\_All, PolarNRZ\_All, PolarRZ\_All, samples\_per\_bit);

t\_shifted = t(1:length(Unipolar\_Shifted)); % Ensure the time vector matches

%plot after shift

plot\_linecodes(Data, Unipolar\_Shifted, PolarNRZ\_Shifted, PolarRZ\_Shifted,...

t\_shifted, num\_bits-1, 'Realization 1 shifted');

% Convert to double for accuracy

Unipolar\_Shifted = double(Unipolar\_Shifted);

PolarNRZ\_Shifted = double(PolarNRZ\_Shifted);

PolarRZ\_Shifted = double(PolarRZ\_Shifted);

%calculate the mean acrros time (question 1)

Unipolar\_Mean = calculate\_mean(Unipolar\_Shifted);

PolarNRZ\_Mean = calculate\_mean(PolarNRZ\_Shifted);

PolarRZ\_Mean = calculate\_mean(PolarRZ\_Shifted);

%plot the mean across time

plot\_mean\_waveforms(t\_shifted, Unipolar\_Mean, PolarNRZ\_Mean, PolarRZ\_Mean, A);

% Compute variance for each code line

Unipolar\_Var = calculate\_variance(Unipolar\_Shifted);

PolarNRZ\_Var = calculate\_variance(PolarNRZ\_Shifted);

PolarRZ\_Var = calculate\_variance(PolarRZ\_Shifted);

%plot the vatiance across time

plot\_variance(t\_shifted, Unipolar\_Var, PolarNRZ\_Var, PolarRZ\_Var);

% Determine max\_lag dynamically

max\_lag = size(Unipolar\_Shifted, 2) - 1;

%calculate the autocorrelation (question 3)

[Unipolar\_AutoCorr, PolarNRZ\_AutoCorr, PolarRZ\_AutoCorr] = ...

compute\_stat\_autocorr(Unipolar\_Shifted, PolarNRZ\_Shifted, PolarRZ\_Shifted, max\_lag);

%plot the autocorrelation

plot\_autocorrelation(Unipolar\_AutoCorr, PolarNRZ\_AutoCorr, PolarRZ\_AutoCorr, max\_lag)

% Compute time autocorrelation

[R\_unipolar\_nrz\_t, R\_polar\_nrz\_t, R\_polar\_rz\_t, taw] =...

compute\_time\_autocorr(Unipolar\_Shifted, PolarNRZ\_Shifted, PolarRZ\_Shifted);

% Plot the time autocorrelation

plot\_time\_autocorrelation(R\_unipolar\_nrz\_t, R\_polar\_nrz\_t, R\_polar\_rz\_t, taw, max\_lag);

% Compute time mean for each line code

Unipolar\_TimeMean = compute\_time\_mean(Unipolar\_Shifted);

PolarNRZ\_TimeMean = compute\_time\_mean(PolarNRZ\_Shifted);

PolarRZ\_TimeMean = compute\_time\_mean(PolarRZ\_Shifted);

% Plot the time mean

plot\_realizations\_with\_mean(t\_shifted, Unipolar\_TimeMean, Unipolar\_Shifted, 'Unipolar NRZ', 'g');

plot\_realizations\_with\_mean(t\_shifted, PolarNRZ\_TimeMean, PolarNRZ\_Shifted, 'Polar NRZ', 'b');

plot\_realizations\_with\_mean(t\_shifted, PolarRZ\_TimeMean, PolarRZ\_Shifted, 'Polar RZ', 'r');

%Plot time mean vs realization

plot\_time\_mean\_vs\_realization(Unipolar\_TimeMean, PolarNRZ\_TimeMean, PolarRZ\_TimeMean, A);

fs = 100; % Sampling frequency

[BW\_unipolar, BW\_polarNRZ, BW\_polarRZ] = ...

estimate\_all\_bandwidths(R\_unipolar\_nrz\_t, R\_polar\_nrz\_t, R\_polar\_rz\_t, fs);

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

function [Unipolar, PolarNRZ, PolarRZ] = generate\_linecodes(Data, A, samples\_per\_bit)

% Ensure input Data is of type int8

Data = int8(Data);

% Convert samples\_per\_bit to double for safe calculations

samples\_per\_bitd = double(samples\_per\_bit);

% Unipolar NRZ: 0 → 0V, 1 → A

Unipolar = int8(Data \* A);

Unipolar = repelem(Unipolar, samples\_per\_bit); % Repeat each bit for duration

% Polar NRZ: 0 → -A, 1 → +A

PolarNRZ = int8((2 \* Data - 1) \* A);

PolarNRZ = repelem(PolarNRZ, samples\_per\_bit);

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

PolarRZ = PolarNRZ;

% Apply RZ rule: second half of each bit period should be zero

i = length(Data); % Start from the last bit

while i > 0

end\_idx = i \* samples\_per\_bitd; % Last sample of the bit

start\_idx = end\_idx - floor(samples\_per\_bitd / 2) + 1; % Start of the second half

PolarRZ(start\_idx:end\_idx) = 0; % Set the second half of the bit period to zero

i = i - 1; % Move to the previous bit

end

end

function plot\_linecodes(Data, Unipolar, PolarNRZ, PolarRZ, t, num\_bits\_to\_show, plot\_title)

% Ensure num\_bits\_to\_show does not exceed the actual number of bits

num\_samples\_per\_bit = ceil(length(t) / length(Data));

num\_samples\_to\_show = num\_bits\_to\_show \* num\_samples\_per\_bit;

% Trim the signals to display only the required number of bits

t\_show = t(1:num\_samples\_to\_show);

Unipolar\_show = Unipolar(1, 1:num\_samples\_to\_show); % Select row 1 explicitly

PolarNRZ\_show = PolarNRZ(1, 1:num\_samples\_to\_show);

PolarRZ\_show = PolarRZ(1, 1:num\_samples\_to\_show);

% Convert Data into a sample-wise representation for accurate plotting

Data\_show = repelem(Data(1:num\_bits\_to\_show), num\_samples\_per\_bit);

Data\_t = t(1:length(Data\_show)); % Adjust time axis

% Plot the signals

figure;

sgtitle(plot\_title); % Set a title for the entire figure

subplot(4,1,1);

stairs(Data\_t, Data\_show, 'k', 'LineWidth', 2);

title('Orignal Data');

ylim([-0.5, 1.5]); % Keep the binary level range

yticks([0 1]);

yticklabels({'0', '1'});

grid on;

subplot(4,1,2);

stairs(t\_show, Unipolar\_show, 'b', 'LineWidth', 2);

title('Unipolar NRZ');

grid on;

subplot(4,1,3);

stairs(t\_show, PolarNRZ\_show, 'r', 'LineWidth', 2);

title('Polar NRZ');

grid on;

subplot(4,1,4);

stairs(t\_show, PolarRZ\_show, 'm', 'LineWidth', 2);

title('Polar RZ');

grid on;

xlabel('Time (s)');

end

function [Unipolar\_Shifted, PolarNRZ\_Shifted, PolarRZ\_Shifted] =...

apply\_random\_shift\_fixed\_size(Unipolar\_All, PolarNRZ\_All, PolarRZ\_All, samples\_per\_bit)

% Define parameters

N\_realizations = size(Unipolar\_All, 1); % 500 realizations

extended\_samples = size(Unipolar\_All, 2); % 707 samples

total\_samples = 700; % Fixed output size

% Initialize shifted matrices

Unipolar\_Shifted = zeros(N\_realizations, total\_samples, 'int8');

PolarNRZ\_Shifted = zeros(N\_realizations, total\_samples, 'int8');

PolarRZ\_Shifted = zeros(N\_realizations, total\_samples, 'int8');

% Apply random shift to each realization

for i = 1:N\_realizations

% Generate random shift in range [0, samples\_per\_bit-1] samples

random\_shift\_bits = randi([0, samples\_per\_bit-1]);

% Extract shifted region

Unipolar\_Shifted(i, :) = Unipolar\_All(i, random\_shift\_bits+1 : random\_shift\_bits+total\_samples);

PolarNRZ\_Shifted(i, :) = PolarNRZ\_All(i, random\_shift\_bits+1 : random\_shift\_bits+total\_samples);

PolarRZ\_Shifted(i, :) = PolarRZ\_All(i, random\_shift\_bits+1 : random\_shift\_bits+total\_samples);

end

end

function mean\_waveform = calculate\_mean(waveform\_matrix)

% Calculates the mean across all realizations without using the mean function

% waveform\_matrix: Matrix where each row is a realization

[num\_realizations, num\_samples] = size(waveform\_matrix); % Get matrix dimensions

mean\_waveform = sum(waveform\_matrix, 1) / num\_realizations; % Sum and divide by count

end

function plot\_mean\_waveforms(t, Unipolar\_Mean, PolarNRZ\_Mean, PolarRZ\_Mean, A)

% Function to plot the mean waveforms of different line codes across time

% Inputs:

% t - Time vector

% Unipolar\_Mean - Mean waveform of Unipolar NRZ

% PolarNRZ\_Mean - Mean waveform of Polar NRZ

% PolarRZ\_Mean - Mean waveform of Polar RZ

% A - Amplitude limit

figure;

subplot(3,1,1);

plot(t, Unipolar\_Mean, 'r', 'LineWidth', 2);

xlabel('Time (s)');

ylabel('Amplitude');

title('Mean of Unipolar NRZ');

grid on;

ylim([-A, A]); % Set y-axis limits

subplot(3,1,2);

plot(t, PolarNRZ\_Mean, 'g', 'LineWidth', 2);

xlabel('Time (s)');

ylabel('Amplitude');

title('Mean of Polar NRZ');

grid on;

ylim([-A, A]);

subplot(3,1,3);

plot(t, PolarRZ\_Mean, 'b', 'LineWidth', 2);

xlabel('Time (s)');

ylabel('Amplitude');

title('Mean of Polar RZ');

grid on;

ylim([-A, A]);

% Add a super title for the whole figure

sgtitle('Mean of Different Line Codes');

end

function variance\_waveform = calculate\_variance(waveform\_matrix)

% Calculates the variance across all realizations (column-wise)

% waveform\_matrix: Matrix where each row is a realization (num\_realizations x num\_samples)

% Returns: variance\_waveform (1 x num\_samples), representing the variance of each sample point

% Compute the mean using the previously implemented function

mean\_waveform = calculate\_mean(waveform\_matrix);

[num\_realizations, num\_samples] = size(waveform\_matrix); % Get dimensions

% Ensure mean\_waveform is the same size for element-wise subtraction

mean\_waveform = repmat(mean\_waveform, num\_realizations, 1);

% Compute variance manually using the variance formula

variance\_waveform = sum((waveform\_matrix - mean\_waveform).^2, 1) / num\_realizations; % Population variance

end

function plot\_variance(t, Unipolar\_Var, PolarNRZ\_Var, PolarRZ\_Var)

% Plots the variance of different line codes over time

% t: Time vector

% Unipolar\_Var, PolarNRZ\_Var, PolarRZ\_Var: Variance waveforms

figure;

subplot(3,1,1);

plot(t, Unipolar\_Var, 'r', 'LineWidth', 2);

xlabel('Time (s)');

ylabel('Variance');

title('Variance of Unipolar NRZ');

grid on;

subplot(3,1,2);

plot(t, PolarNRZ\_Var, 'g', 'LineWidth', 2);

xlabel('Time (s)');

ylabel('Variance');

title('Variance of Polar NRZ');

grid on;

subplot(3,1,3);

plot(t, PolarRZ\_Var, 'b', 'LineWidth', 2);

xlabel('Time (s)');

ylabel('Variance');

title('Variance of Polar RZ');

grid on;

% Add a super title for clarity

sgtitle('Variance of Different Line Codes');

end

function [Unipolar\_AutoCorr, PolarNRZ\_AutoCorr, PolarRZ\_AutoCorr] =...

compute\_stat\_autocorr(Unipolar\_Shifted, PolarNRZ\_Shifted, PolarRZ\_Shifted, max\_lag)

% Compute Statistical Autocorrelation for given signals using calculate\_mean

% Inputs:

% Unipolar\_Shifted - Shifted signal matrix for Unipolar NRZ

% PolarNRZ\_Shifted - Shifted signal matrix for Polar NRZ

% PolarRZ\_Shifted - Shifted signal matrix for Polar RZ

% Outputs:

% Unipolar\_AutoCorr - Computed autocorrelation for Unipolar NRZ

% PolarNRZ\_AutoCorr - Computed autocorrelation for Polar NRZ

% PolarRZ\_AutoCorr - Computed autocorrelation for Polar RZ

% Set x-axis limits dynamically

x\_limit = max\_lag / 10;

% Initialize autocorrelation arrays

Unipolar\_AutoCorr = zeros(1, max\_lag + 1);

PolarNRZ\_AutoCorr = zeros(1, max\_lag + 1);

PolarRZ\_AutoCorr = zeros(1, max\_lag + 1);

% Compute mean autocorrelation using calculate\_mean function

for i = 0:max\_lag

Unipolar\_AutoCorr(i+1) = calculate\_mean(Unipolar\_Shifted(:, 1) .\* Unipolar\_Shifted(:, i+1));

PolarNRZ\_AutoCorr(i+1) = calculate\_mean(PolarNRZ\_Shifted(:, 1) .\* PolarNRZ\_Shifted(:, i+1));

PolarRZ\_AutoCorr(i+1) = calculate\_mean(PolarRZ\_Shifted(:, 1) .\* PolarRZ\_Shifted(:, i+1));

end

% Time axis for plotting

t = -max\_lag:max\_lag;

% Compute symmetric autocorrelation values

Unipolar\_AutoCorr = [fliplr(Unipolar\_AutoCorr), Unipolar\_AutoCorr(2:end)];

PolarNRZ\_AutoCorr = [fliplr(PolarNRZ\_AutoCorr), PolarNRZ\_AutoCorr(2:end)];

PolarRZ\_AutoCorr = [fliplr(PolarRZ\_AutoCorr), PolarRZ\_AutoCorr(2:end)];

end

function plot\_autocorrelation(Unipolar\_AutoCorr, PolarNRZ\_AutoCorr, PolarRZ\_AutoCorr, max\_lag)

% Plots the statistical autocorrelation of Unipolar NRZ, Polar NRZ, and Polar RZ

% Inputs:

% Unipolar\_AutoCorr - Autocorrelation for Unipolar NRZ

% PolarNRZ\_AutoCorr - Autocorrelation for Polar NRZ

% PolarRZ\_AutoCorr - Autocorrelation for Polar RZ

% max\_lag - Maximum lag value used for the time axis

% Time axis

t = -max\_lag:max\_lag;

% Compute x-axis limit

x\_limit = max\_lag / 10;

% Plot results

figure("name", "Statistical Autocorrelation");

subplot(3,1,1);

plot(t, Unipolar\_AutoCorr, 'g');

xlim([-x\_limit x\_limit]);

xlabel("Time axis");

ylabel("Autocorr axis");

title("Unipolar NRZ Statistical Autocorrelation");

subplot(3,1,2);

plot(t, PolarNRZ\_AutoCorr, 'b');

xlim([-x\_limit x\_limit]);

xlabel("Time axis");

ylabel("Autocorr axis");

title("Polar NRZ Statistical Autocorrelation");

subplot(3,1,3);

plot(t, PolarRZ\_AutoCorr, 'r');

xlim([-x\_limit x\_limit]);

xlabel("Time axis");

ylabel("Autocorr axis");

title("Polar RZ Statistical Autocorrelation");

end

function TimeMean = compute\_time\_mean(waveform\_matrix)

% Computes the time mean for each realization of a given waveform

% Inputs:

% waveform\_matrix - Matrix where each row represents a realization

% Output:

% TimeMean - Column vector containing the time mean for each realization

% Compute time mean for each realization (mean along rows)

TimeMean = sum(waveform\_matrix, 2) / size(waveform\_matrix, 2);

end

function plot\_realizations\_with\_mean(t\_shifted, Signals\_TimeMean, signals\_waveform, signal\_name, color)

% Plots the first 3 realizations of a signal in a 3x2 grid and displays their time means as text.

%

% Inputs:

% t\_shifted - Time vector

% Signals\_TimeMean - Vector of time means (one per realization)

% signals\_waveform - Matrix where each row is a realization

% signal\_name - Name of the signal (string) for labeling

% color - Plot color (e.g., 'g' for green)

figure('Name', [signal\_name, ' - Realizations and Time Mean']);

for i = 1:3

% First Column: Plot the waveform realization

subplot(3,2,(i-1)\*2+1);

plot(t\_shifted, signals\_waveform(i,:), color, 'LineWidth', 1.5); % Plot waveform

hold on;

yline(Signals\_TimeMean(i), '--k', 'LineWidth', 1.5); % Add time mean line

hold off;

xlabel('Time');

ylabel('Amplitude');

title([signal\_name, ' - Realization ', num2str(i)]);

grid on;

% Second Column: Display time mean as a text box

subplot(3,2,(i-1)\*2+2);

axis off; % Hide axes for a clean text display

text(0.5, 0.5, sprintf('Time Mean:\n%.3f', Signals\_TimeMean(i)), ...

'FontSize', 12, 'FontWeight', 'bold', 'HorizontalAlignment', 'center', 'BackgroundColor', 'w');

end

end

function plot\_time\_mean\_vs\_realization(unipolar\_mean, polarNRZ\_mean, polarRZ\_mean, A)

% Function to plot the time mean vs realization index for different signals separately

%

% Inputs:

% - unipolar\_mean: Time mean of Unipolar NRZ (1xN vector)

% - polarNRZ\_mean: Time mean of Polar NRZ (1xN vector)

% - polarRZ\_mean: Time mean of Polar RZ (1xN vector)

% - A: Amplitude limit for y-axis

% Define the x-axis (Index based on row size)

num\_realizations = length(unipolar\_mean);

realization\_indices = 1:num\_realizations;

% Create a figure for subplots

figure;

% Subplot 1: Unipolar NRZ

subplot(3,1,1);

plot(realization\_indices, unipolar\_mean, 'r', 'LineWidth', 2);

grid on;

xlabel('Realization Index');

ylabel('Time Mean');

title('Unipolar NRZ Time Mean');

ylim([-A, A]); % Set y-axis limits

% Subplot 2: Polar NRZ

subplot(3,1,2);

plot(realization\_indices, polarNRZ\_mean, 'g', 'LineWidth', 2);

grid on;

xlabel('Realization Index');

ylabel('Time Mean');

title('Polar NRZ Time Mean');

ylim([-A, A]);

% Subplot 3: Polar RZ

subplot(3,1,3);

plot(realization\_indices, polarRZ\_mean, 'b', 'LineWidth', 2);

grid on;

xlabel('Realization Index');

ylabel('Time Mean');

title('Polar RZ Time Mean');

ylim([-A, A]);

% Add a super title for the whole figure

sgtitle('Time Mean vs Realization Index');

end

function [R\_unipolar\_nrz\_t, R\_polar\_nrz\_t, R\_polar\_rz\_t, taw2] = ...

compute\_time\_autocorr(UnipolarNRZ, PolarNRZ, PolarRZ)

% Computes time autocorrelation for each realization separately.

%

% Inputs:

% UnipolarNRZ - Matrix containing all realizations for Unipolar NRZ

% PolarNRZ - Matrix containing all realizations for Polar NRZ

% PolarRZ - Matrix containing all realizations for Polar RZ

%

% Outputs:

% R\_unipolar\_nrz\_t - Time autocorrelation for Unipolar NRZ (each row computed separately)

% R\_polar\_nrz\_t - Time autocorrelation for Polar NRZ (each row computed separately)

% R\_polar\_rz\_t - Time autocorrelation for Polar RZ (each row computed separately)

% Get number of realizations and samples

[num\_realizations, num\_samples] = size(UnipolarNRZ);

% Define range of time lags

max\_lag = num\_samples - 1; % Maximum lag value

taw2 = -max\_lag:max\_lag; % Lag vector

% Initialize autocorrelation matrices (each row for a realization)

R\_unipolar\_nrz\_t = zeros(num\_realizations, length(taw2));

R\_polar\_nrz\_t = zeros(num\_realizations, length(taw2));

R\_polar\_rz\_t = zeros(num\_realizations, length(taw2));

% Compute autocorrelation for each realization separately

for r = 1:num\_realizations

for i = taw2

M = i + max\_lag + 1; % Shift index to fit within array bounds

% Compute sum for each lag (dot product of signal with shifted version)

if i >= 0

valid\_samples = num\_samples - i;

R\_unipolar\_nrz\_t(r, M) = sum(UnipolarNRZ(r, 1:valid\_samples) .\* UnipolarNRZ(r, i+1:num\_samples)) / valid\_samples;

R\_polar\_nrz\_t(r, M) = sum(PolarNRZ(r, 1:valid\_samples) .\* PolarNRZ(r, i+1:num\_samples)) / valid\_samples;

R\_polar\_rz\_t(r, M) = sum(PolarRZ(r, 1:valid\_samples) .\* PolarRZ(r, i+1:num\_samples)) / valid\_samples;

else

valid\_samples = num\_samples + i;

R\_unipolar\_nrz\_t(r, M) = sum(UnipolarNRZ(r, -i+1:num\_samples) .\* UnipolarNRZ(r, 1:valid\_samples)) / valid\_samples;

R\_polar\_nrz\_t(r, M) = sum(PolarNRZ(r, -i+1:num\_samples) .\* PolarNRZ(r, 1:valid\_samples)) / valid\_samples;

R\_polar\_rz\_t(r, M) = sum(PolarRZ(r, -i+1:num\_samples) .\* PolarRZ(r, 1:valid\_samples)) / valid\_samples;

end

end

end

end

function plot\_time\_autocorrelation(R\_unipolar, R\_polarNRZ, R\_polarRZ, taw, max\_lag)

% Plots the time autocorrelation of the first realization for each waveform type.

%

% Inputs:

% R\_unipolar - Matrix containing time autocorrelation for Unipolar NRZ (each row is a realization)

% R\_polarNRZ - Matrix containing time autocorrelation for Polar NRZ (each row is a realization)

% R\_polarRZ - Matrix containing time autocorrelation for Polar RZ (each row is a realization)

% taw - Vector of lag values

% max\_lag - Maximum lag value to set axis limits dynamically

% Set dynamic x-axis limits based on max\_lag

x\_limit = max\_lag / 10;

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

% Plot Unipolar NRZ

subplot(3,1,1);

plot(taw, R\_unipolar(1, :), 'b', 'LineWidth', 1.5);

grid on;

xlim([-x\_limit x\_limit]); % Adjust only the x-axis dynamically

xlabel('Time Lag (samples)');

ylabel('Magnitude');

title('Time Autocorrelation of Unipolar NRZ');

% Plot Polar NRZ

subplot(3,1,2);

plot(taw, R\_polarNRZ(1, :), 'r', 'LineWidth', 1.5);

grid on;

xlim([-x\_limit x\_limit]); % Adjust only the x-axis dynamically

xlabel('Time Lag (samples)');

ylabel('Magnitude');

title('Time Autocorrelation of Polar NRZ');

% Plot Polar RZ

subplot(3,1,3);

plot(taw, R\_polarRZ(1, :), 'g', 'LineWidth', 1.5);

grid on;

xlim([-x\_limit x\_limit]); % Adjust only the x-axis dynamically

xlabel('Time Lag (samples)');

ylabel('Magnitude');

title('Time Autocorrelation of Polar RZ');

end

function [BW\_unipolar, BW\_polarNRZ, BW\_polarRZ] = ...

estimate\_all\_bandwidths(R\_unipolar\_nrz, R\_polar\_nrz, R\_polar\_rz, fs)

% Function to estimate the bandwidth for Unipolar NRZ, Polar NRZ, and Polar RZ

% using FFT and power spectral density (PSD) method.

% Extract the first realization

R\_unipolar\_nrz = R\_unipolar\_nrz(:,1);

R\_polar\_nrz = R\_polar\_nrz(:,1);

R\_polar\_rz = R\_polar\_rz(:,1);

% Number of samples (adjust dynamically)

n = length(R\_unipolar\_nrz);

% Compute FFT coefficients

unipolar\_nrz\_coe = fft(R\_unipolar\_nrz) / n;

polar\_nrz\_coe = fft(R\_polar\_nrz) / n;

polar\_rz\_coe = fft(R\_polar\_rz) / n;

% Compute magnitude of FFT (PSD estimation)

amplitude\_unipolar\_nrz = abs(unipolar\_nrz\_coe);

amplitude\_polar\_nrz = abs(polar\_nrz\_coe);

amplitude\_polar\_rz = abs(polar\_rz\_coe);

% Frequency axis mapping (centered around 0)

freq\_axis = (-n/2:n/2-1) \* (fs / n); % Corrected frequency scaling

% Shift FFT for centered display

amp\_unipolar\_nrz = fftshift(amplitude\_unipolar\_nrz);

amp\_polar\_nrz = fftshift(amplitude\_polar\_nrz);

amp\_polar\_rz = fftshift(amplitude\_polar\_rz);

% Estimate bandwidths using -3dB method

BW\_unipolar = find\_bandwidth(amp\_unipolar\_nrz, freq\_axis);

BW\_polarNRZ = find\_bandwidth(amp\_polar\_nrz, freq\_axis);

BW\_polarRZ = find\_bandwidth(amp\_polar\_rz, freq\_axis);

%{

% Display Bandwidth Values

disp(['BW Unipolar NRZ: ', num2str(BW\_unipolar), ' Hz']);

disp(['BW Polar NRZ: ', num2str(BW\_polarNRZ), ' Hz']);

disp(['BW Polar RZ: ', num2str(BW\_polarRZ), ' Hz']);

%}

% Plot PSDs

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

subplot(3,1,1);

plot(freq\_axis, amp\_unipolar\_nrz, 'g');

ylim([0 max(amp\_unipolar\_nrz)\*1.1]); xlim([-fs/2 fs/2]);

grid on;

xlabel('Frequency (Hz)');

ylabel('Magnitude');

title('PSD of Unipolar NRZ');

subplot(3,1,2);

plot(freq\_axis, amp\_polar\_nrz, 'b');

ylim([0 max(amp\_polar\_nrz)\*1.1]); xlim([-fs/2 fs/2]);

grid on;

xlabel('Frequency (Hz)');

ylabel('Magnitude');

title('PSD of Polar NRZ');

subplot(3,1,3);

plot(freq\_axis, amp\_polar\_rz, 'r');

ylim([0 max(amp\_polar\_rz)\*1.1]); xlim([-fs/2 fs/2]);

grid on;

xlabel('Frequency (Hz)');

ylabel('Magnitude');

title('PSD of Polar RZ');

end

function BW = find\_bandwidth(PSD, freq\_axis)

% Function to find the -3dB bandwidth from PSD

PSD\_max = max(PSD); % Find max power

threshold = PSD\_max / 2; % -3dB threshold (half power)

% Find indices where PSD is above threshold

valid\_indices = find(PSD >= threshold);

% Check if we found any valid points

if isempty(valid\_indices)

BW = 0; % If no valid range is found, return 0

return;

end

% Ensure indices are within valid bounds

min\_index = max(1, min(valid\_indices)); % Ensure it's at least 1

max\_index = min(length(freq\_axis), max(valid\_indices)); % Ensure it's within bounds

% Compute bandwidth as the difference between first and last crossing points

BW = abs(freq\_axis(max\_index) - freq\_axis(min\_index));

end