Cairo University

Cairo University, Faculty of Engineering
Electronics and Electrical Communications Department (EECE)

Digital Communication

Project1_ELC3070

Team:13

Name	ID	Sec
احمد محمد إبراهيم عبده	9220078	1
روان خالد محمد محمد	9220303	2
محمد خالد محمد شتا	9203230	3
احمد خلف محمد علي	9220036	1
خالد محمد رمضان علي	9221589	2

Submitted to: **Eng. Mohamed Khaled**

Date: 28-3-2025

Table of Contents

Table of Figures

The Role of each member

We all contributed to writing the report.

Name	ID
احمد محمد إبراهيم عبده	Theoretical mean, Auto-correlation, (PSD) analysis
روان خالد محمد محمد	Data generation and line codes formation
محمد خالد محمد شتا	Time Auto-correlation function
احمد خلف محمد علي	Statistical Mean, Time Mean, and PSD functions
خالد محمد رمضان علي	Statistical Auto-correlation function

Introduction:

Software radio is a clever and modern way to handle communication, where software, not fixed hardware pieces, controls how signals are created and understood. This setup lets us change things quickly like how we shape the signals or send data without needing to build new equipment every time. It’s like having a toolbox in your computer that can adapt to whatever job you give it.

In this project, we’re diving into this idea by making different kinds of line-coded waveforms and studying them with MATLAB, a program that helps us build and test these signals step by step. We’ll focus on generating waveforms for three line coding methods: Unipolar Signaling, Polar Non-Return-to-Zero (NRZ), and Polar Return-to-Zero (RZ). Each method has its own way of turning bits those 0s and 1s of digital data into signals we can send.

Our job is to create some waveforms for each type, look at simple things like their mean, and figure out how they behave over time. We’ll also ask questions like: Do these signals stay steady no matter when we look at them? And can one waveform tell us everything about the whole group? This helps us understand how software radio can make communication easier and more powerful. By using software to define these waveforms, we’re showing how it can replace clunky old hardware and still get the job done.

Problem description:

The goal of this project is to explore and use line coding techniques for digital communication by working with MATLAB. We want to create big groups of waveforms, called ensembles, and look closely at their statistical properties. Each ensemble will have 500 different waveforms, and each waveform will carry 100 bits of random data, each bit is represented by 7 samples, giving us plenty to study. We’ll also check two important things: whether these waveforms are stationary, meaning their behavior doesn’t shift as time goes on, and whether they are ergodic, which means the overall pattern of the group matches what we’d see in just one waveform over a long period.

The focus is on three types of line coding Unipolar Signaling, Polar Non-Return-to-Zero (NRZ), and Polar Return-to-Zero (RZ), figuring out their strengths and weaknesses. Ultimately, we want to connect our findings to real-world uses in digital communication, showing how software can solve problems that hardware alone can’t handle easily.

This project ties into software radio systems, a modern way of communicating where software takes charge instead of old-fashioned hardware. In software radio, the computer decides how signals are made and sent, so understanding how these waveforms act helps us see how well this system can work. By doing this, we’ll learn how to design and analyze signals that could be used to send data through cables or wireless channels, making communication flexible and efficient.

Control Flags:

To make and study our waveforms, we use a set of key settings in MATLAB that guide how everything works. Here's what they are and why they matter:

- **Number of Waveforms:** This tells us how many waveforms we're creating in total, and it's required to make 500 different realizations to look at.
- **Signal Strength (Amplitude)** : This sets the voltage level for our signals. It could be +4 volts for a "1" or -4 volts for a "0", depending on the line code type.
- **Samples per Bit:** Each bit gets stretched into 7 samples. This mimics the DAC device turning on every 10 milliseconds to make a 70-millisecond bit pulse width.

$$\text{Samples per Bit} = \frac{Pulse\ width\ of\ 1\ bit}{DAC\ timing} = \frac{70}{10} = 7\ samples$$

- **Samples Length:** Every waveform ends up with 700 samples. Since we have 100 bits and 7 samples per bit.

$$\text{Samples length} = \text{samples per bit } \times \text{no, of bits} = 7 \times 100 = 700\ samples$$

- **Number of bits with delay** : We start with 101 bits instead of just 100. That extra bit gives us room to account for adding random shifts up to 7 samples.
- **Line Code Type**: This chooses how we turn bits into signals, The user can pick one type at a time to study, its behavior and statistical analysis.

These settings keep everything on track. They make sure all signals follow the same rules, which is important for testing how they behave in a software radio setup.

Data Generation:

Random binary data is generated for each of the waveforms using the following commands:

```
for i = 1:num_waveforms
    data = randi([0 1], 1, num_bits_with_delay); % choose random number 0 or 1
```

 Loop Structure: The first line initiates a loop that iterates 500 times. This ensures that a unique data sequence is created for each of the 500 waveforms in the ensemble.

 Data Creation: Inside the loop we generate a 1×101 array of random binary values (0 or 1), where. This produces a sequence of 101 bits, representing the binary data to be transmitted, with an extra bit included to accommodate subsequent processing steps like random delays.

Unipolar NRZ Ensemble:

1. MATLAB Implementation Code

```
% Unipolar NRZ Signal
    unipolar_nrz_signal = data * Amplitude;
    unipolar_nrz_waveform = repmat(unipolar_nrz_signal, samples_per_bit, 1);
    unipolar_nrz_waveform = reshape(unipolar_nrz_waveform, 1, []);
```

2. Code Explanation

 Signal Mapping: The first line converts the binary data into Unipolar NRZ voltage levels. Multiplying data by Amplitude maps 0 to 0 V and 1 to +4 V, producing Unipolar NRZ signal.

 Sample Repetition: the second line replicates each voltage level in unipolar signal vertically. with 7 samples , this generates a 7×101 matrix where each column corresponds to one bit's voltage repeated over 7 samples, simulating a consistent signal duration.

 Reshaping: The final line converts the 7×101 matrix into a single row of 707 elements. This flattens the matrix into a continuous waveform, where each bit's voltage persists for 7 samples, aligning with the Unipolar NRZ scheme.

3. Resulted Waveform:

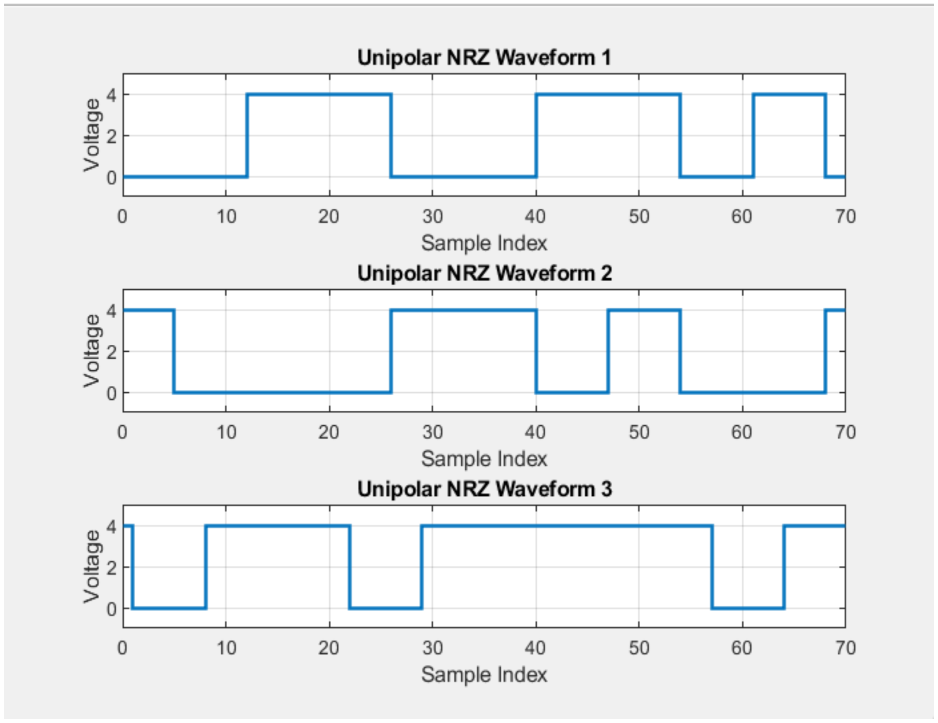


Figure 1. Unipolar Waveform

Polar NRZ Ensemble:

1. MATLAB Implementation Code

```
% polar NRZ Signal
    polar_signal = ((2 * data) - 1) * Amplitude;
    polar_nrz_waveform = repmat(polar_signal, samples_per_bit, 1);
    polar_nrz_waveform = reshape(polar_nrz_waveform, 1, []);
```

2. Code Explanation

 Signal Mapping: The first line transforms the binary data into Polar NRZ voltage levels. The expression **"(2 * data) – 1"** maps 0 to -1 and 1 to 1, which is then scaled by Amplitude to produce -4 V for logic 0 and +4 V for logic 1.

 Sample Repetition: like unipolar, the second line repeats each voltage level in Polar NRZ signal across 7 samples. This creates a 7×101 matrix, where each column represents one bit's voltage repeated over 7 samples, maintaining a steady signal per bit.

 Reshaping: like the unipolar, the final line, transforms the 7×101 matrix into a single row of 707 elements. This forms a continuous waveform, where each bit's voltage persists for 7 samples

3. Resulted Waveform:

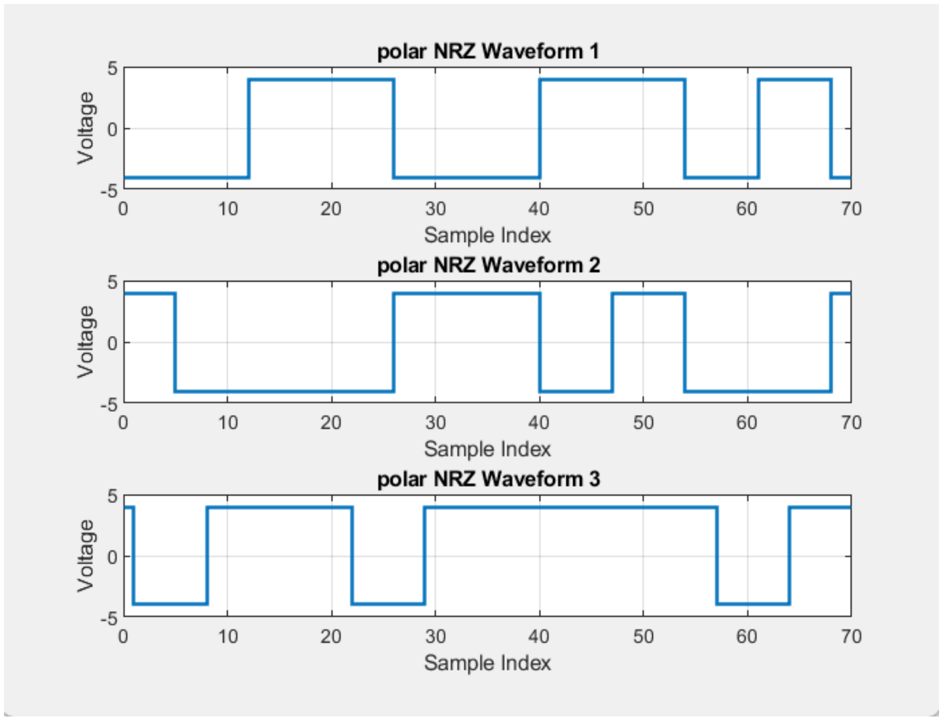


Figure 2. Polar NRZ waveform

Polar RZ Ensemble:

1. MATLAB Implementation Code

```
% polar RZ Signal
    polar_rz_waveform = zeros(samples_per_bit, num_bits_with_delay); % Pre-allocation
to avoid warning and set zeros for last 3 samples
    polar_rz_waveform(1:4, :) = repmat(polar_signal, 4, 1); % first 4 samples are
polar signal , last 3 samples returns to zero
    polar_rz_waveform = reshape(polar_rz_waveform, 1, []);
```

2. Code Explanation

 Matrix Initialization: The first line pre-allocates a matrix of zeros with dimensions 7×101, enhancing performance by avoiding dynamic resizing. It also conveniently sets the last three rows to zero, fulfilling the return-to-zero (RZ) requirement without additional steps later.

 Signal Assignment: The second line, assigns voltage levels from polar signal (-4 V and +4 V) to the first 4 rows of each column. The **"repmat"** function repeats polar signal vertically 4 times, creating a 4×101 matrix that fills rows 1 to 4, while rows 5 to 7 remain 0 V due to pre-allocation step, implementing the return-to-zero feature of Polar RZ.

 Reshaping: like unipolar, The final line converts the 7×101 matrix into a single row of 707 elements. This forms a continuous waveform where each bit's voltage lasts for 4 samples, followed by 3 samples at 0 V, adhering to the Polar RZ scheme.

3. Resulted Waveform:

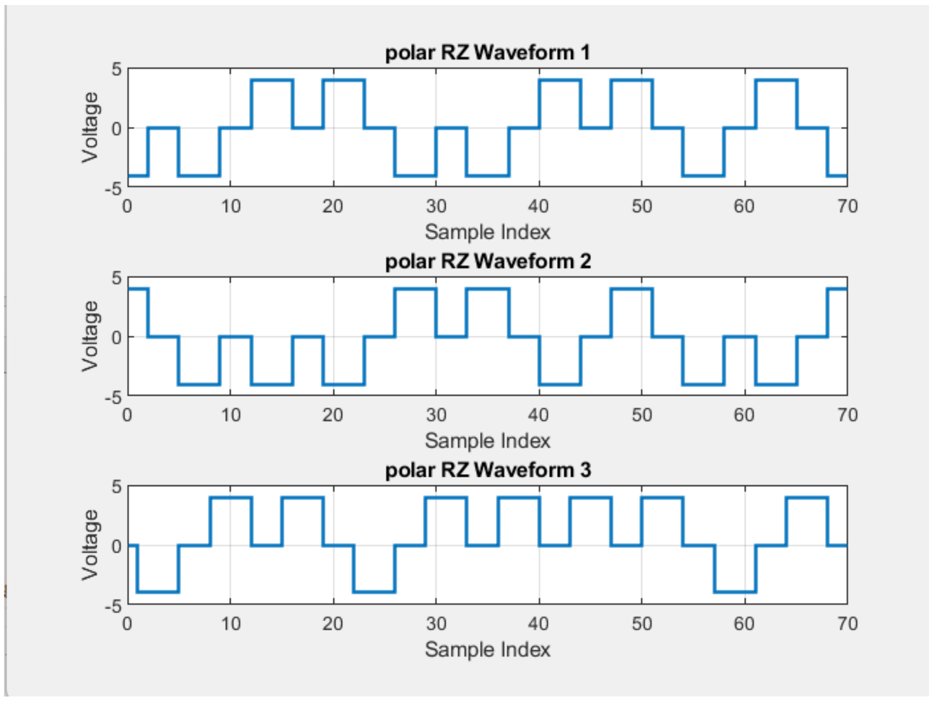


Figure 3. Polar RZ waveform

Delay Generation:

Random initial time delays are applied to each waveform using the following MATLAB code:

```
random_delay = randi([1 7]); % choose random number from 1 to 7
    % Shift waveforms to add random delay
    delayed_unipolar_nrz = unipolar_nrz_waveform(1 + random_delay : end);
    delayed_polar_nrz = polar_nrz_waveform(1 + random_delay : end);
    delayed_polar_rz = polar_rz_waveform(1 + random_delay : end);
```

 Delay Generation: The first line generates a random integer between 1 and 7, inclusive. This value represents the number of samples by which the waveform's start is shifted, simulating variability in transmission timing.

 Waveform Shifting: The subsequent lines shift the Unipolar NRZ, Polar NRZ, and Polar RZ waveforms by random delay samples. For each, the expression selects a portion of the original waveform, starting from the sample at position **"1 + random delay"** (e.g., 2 to 8) and extending to the end. This discards the first random delay samples, effectively delaying the waveform's onset while preserving the remaining sequence.

Finalize the Ensemble:

The waveforms are finalized and stored in their respective ensembles using the following code:

```
% Trim excess samples more than 700
    if length(delayed_unipolar_nrz) > Samples_length
       delayed_unipolar_nrz = delayed_unipolar_nrz(1:Samples_length);
    end

    if length(delayed_polar_nrz) > Samples_length
       delayed_polar_nrz = delayed_polar_nrz(1:Samples_length);
    end

    if length(delayed_polar_rz) > Samples_length
       delayed_polar_rz = delayed_polar_rz(1:Samples_length);
    end
    ensemble_unipolar_nrz(i, :) = delayed_unipolar_nrz;
    ensemble_polar_nrz(i, :) = delayed_polar_nrz;
```

 Trimming Excess Samples: The conditional statements check if the length of each delayed waveform exceeds 700. If true, the expression retains only the first 700 samples. This ensures all waveforms match the required length of 700 samples, standardizing them after random delays.

 Ensemble Storage: The final three lines assign the trimmed waveforms to the i-th row of their respective ensemble matrices. The notation (i, :) specifies the realization index.

Statistical Mean:

The statistical mean of the ensemble represents the expected voltage value across all waveforms at each sample point, assuming the binary data is randomly generated with equal probability. Below, the statistical mean for each line code using the following code snippet.

```
function stat_mean = calc_statistical_mean(ensemble, num_waveforms)
    stat_mean=sum(ensemble, 1)/num_waveforms;
end
```

 Sum: It sums the ensemble along columns at each point in time using sum(ensemble, 1), then divides by 500 to get a row vector with the average value for each sample across all waveforms.

Unipolar NRZ

- **Theoretical Mean:** Since the data bits are equally likely [P(0) = P(1) = 0.5] across all waveforms, the expected voltage per bit is the average of the possible levels:

$$E[Unipolar\ NRZ] = P(0) \times 0 + P(1) \times A = 0.5 \times 0 + 0.5 \times 4 = 2V$$

- **Simulated Mean:**

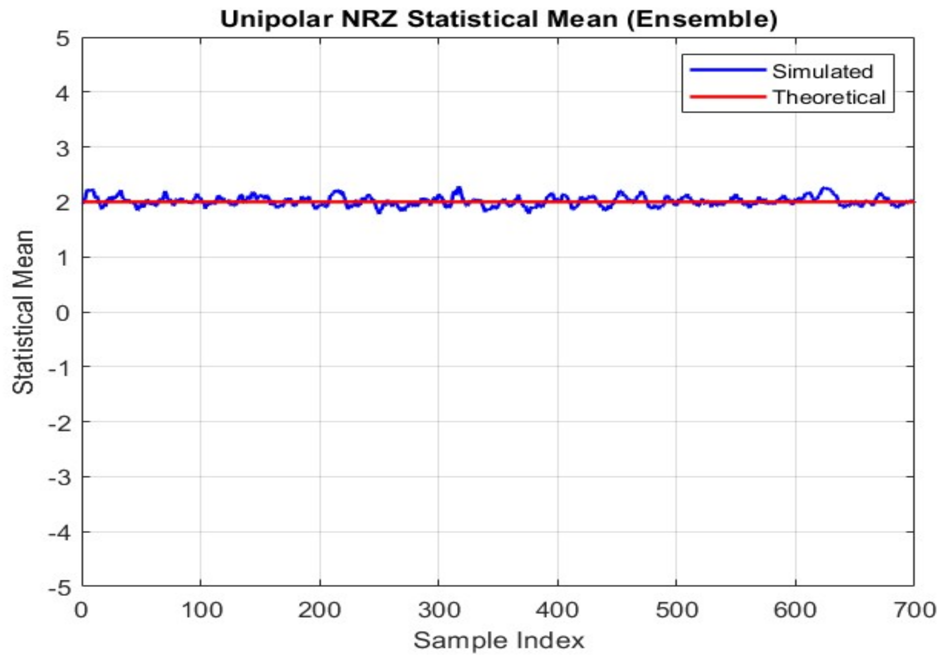


Figure 4.Unipolar Statistical Mean

- The statistical mean of Unipolar NRZ is computed using both simulated and theoretical approaches, showing a strong correlation. The ensemble mean at each point remains fluctuating above and below 2 V, as random data evenly distribute active and zero periods across the 500 waveforms. This confirms the correctness of the result, as the simulated mean is very close to the theoretical.

Polar NRZ

- **Theoretical Mean:** With equal probabilities [P(0) = P(1) = 0.5] across all waveforms, the expected voltage per bit balances the positive and negative levels:

$$E[Polar\ NRZ] = P(0) \times (-A) + P(1) \times A = 0.5 \times (-4) + 0.5 \times 4 = -2 + 2 = 0V$$

- **Simulated Mean:**

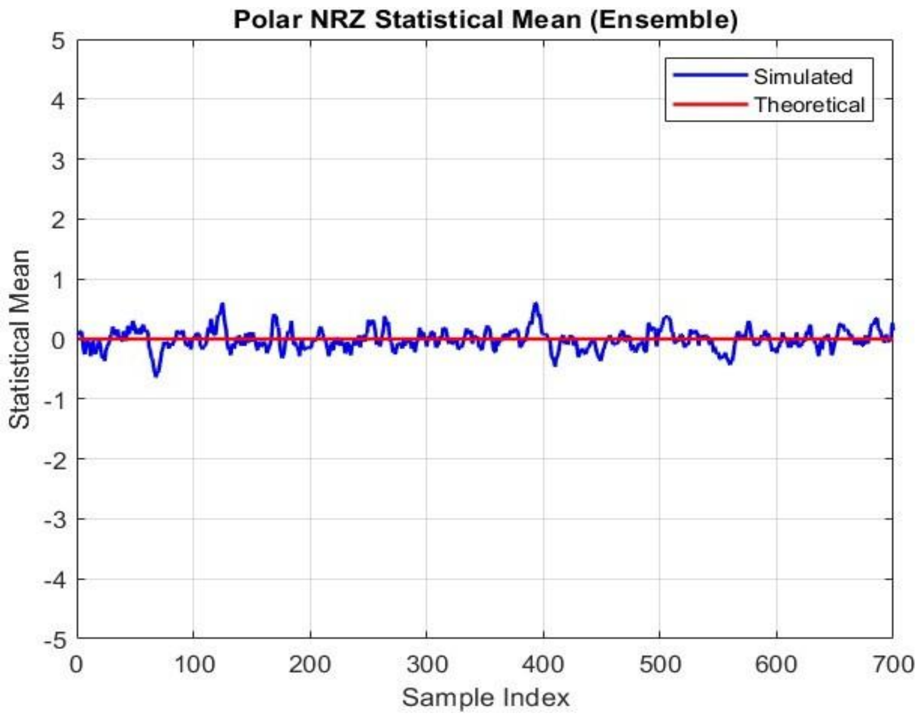


Figure 5. Polar NRZ Statistical Mean

▪ The statistical mean of Polar NRZ is computed using both simulated and theoretical approaches, showing a strong correlation. The ensemble mean at each point remains fluctuating around 0 V, as random data evenly distribute +ve and -ve active periods across the 500 waveforms. This confirms the correctness of the result, as the simulated mean is very close to the theoretical.

Polar RZ

- **Theoretical Mean:** The mean depends on the sample position within a bit due to the return-to-zero feature.
  - ➢ For the first 4 samples (active period), the calculation mirrors Polar NRZ:

$$E[Polar\ RZ,\ active] = 0.5 \times (-4) + 0.5 \times 4 = 0V$$

  - ➢ For the last 3 samples (zero period), the voltage is always 0 V:

$$E[Polar\ RZ,\ zero] = 0V$$

➢ Averaging over the entire 7-sample bit period:

$$E[Polar\ RZ] = \frac{4 \times 0 + 3 \times 0}{7} = 0V$$

The ensemble mean at each sample point remains 0 V, as random delays distribute active and zero periods evenly across the 500 waveforms.

- **Simulated Mean:**

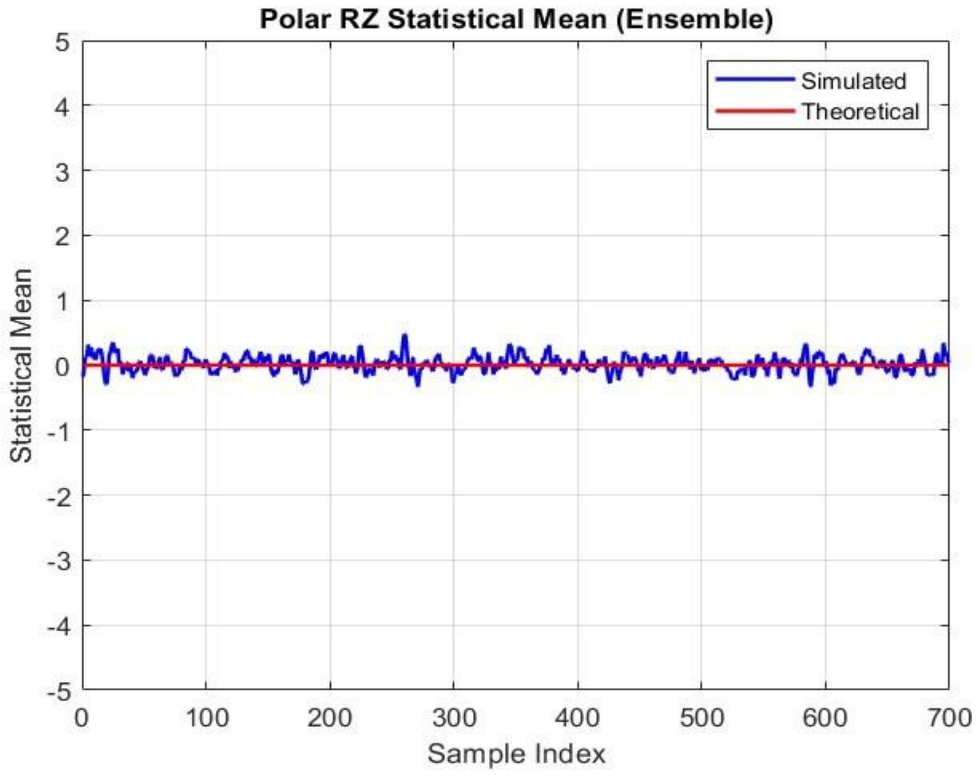


Figure 6.Polar RZ Statistical Mean

▪ The statistical mean of Polar RZ is computed using both simulated and theoretical approaches, showing a strong correlation. The ensemble mean at each point remains fluctuating around 0 V, as random data evenly distribute +ve , -ve active and zero periods across the 500 waveforms. This confirms the correctness of the result, as the simulated mean is very close to the theoretical.

Trial: Unipolar NRZ (5000 realization)

If we increase the number of realizations to 5000 and more , the results will become very close to the theoretical value and may even align with it.

This means that indeed the num_waveforms is a strong control flag that affects the statistical analysis of a digital system.

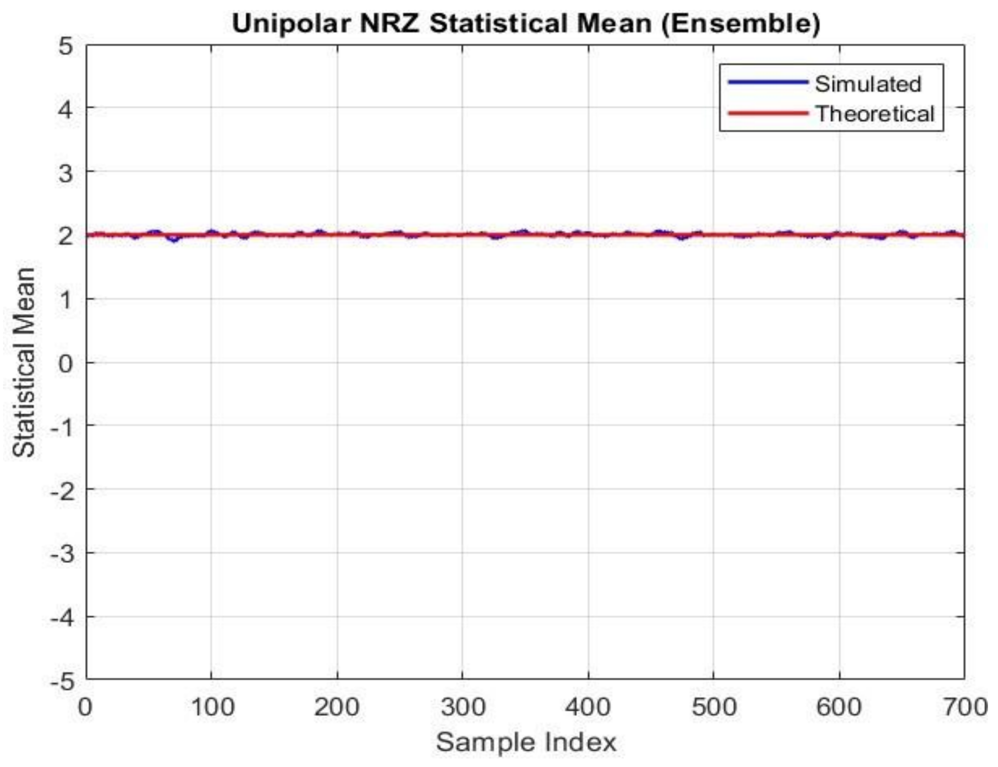


Figure 7.Unipolar NRZ with 5000 realization

- When increasing the number of realizations to 5000, the effect of randomness is reduced when calculating the mean, and the results become more aligned with the theoretical values as the number of realizations approaches infinity.

Statistical Autocorrelation:

Statistical autocorrelation measures how signal values at different time points are correlated across an ensemble of waveforms, for a fixed time delay. It reflects the average similarity between signals at different times, taken across many realizations of the process.

```
% Function to calculate ensemble autocorrelation Rx(tau)
function Rx = calc_ensemble_autocorr(ensemble, num_waveforms, Samples_length, max_tau)
    Rx_pos = zeros(1, max_tau + 1);
    for tau = 0:max_tau
        sum_val = 0;
        for t = 1:(Samples_length - tau)
            for i = 1:num_waveforms
                sum_val = sum_val + ensemble(i, t) * ensemble(i, t + tau);
            end
        end
        Rx_pos(tau + 1) = sum_val / (num_waveforms * (Samples_length - tau));
    end
    Rx = [flip(Rx_pos(2:end)), Rx_pos]; % Mirror for negative lags
end
```

 Pre-allocate array: The first line creates an array of zeros to store autocorrelation values for non-negative lags (0 to max_tau = 699). The +1 accounts for including 700th sample.

 Loop over tau: The second line iterates through all possible lag values to compute autocorrelation at each lag.
 Loop over samples: The next line iterates through valid time indices for a given tau, ensuring no out-of-bound access when shifting by tau.
 Loop over waveforms: This loop iterates through each waveform in the ensemble matrix, processing one waveform at a time.
 Compute product and sum: Next, we multiply the signal at [t] with the signal at [t + tau] for waveform i, adding the result. This accumulates the products for all waveforms at those times.
 Normalize and store: Then we average the produced sum by dividing by the number of waveforms and the number of valid samples for that tau, storing the result in Rx_pos.
 Mirror for negative lags: Lastly, we create the full autocorrelation by mirroring the positive lags to negative lags, then concatenating with Rx_pos to form the symmetric output.

Unipolar NRZ

- **Theoretical Autocorrelation:**

$E(X(t))=(A)*\frac{1}{2}+(0)*\frac{1}{2}=\frac{A}{2}$

For Autocorrelation we have two parts : $t_2-t_1<T_b$ and $t_2-t_1>T_b$

1.For $t_2-t_1<T_b$ and (t_1,t_2) in the same bit

$E(X(t_1)X(t_2))=(A*A)*\frac{1}{2}+(0*0)*\frac{1}{2}=\frac{A^2}{2}$

2.For $t_2-t_1<T_b$ and (t_1,t_2) in adjacent bits

$E(X(t_1)X(t_2))=(A*A)*\frac{1}{4}+(A*0)*\frac{1}{4}+(0*A)*\frac{1}{4}+(0*0)*\frac{1}{4}=\frac{A^2}{4}$

$$R_x(t_1,t_2)=\frac{T_b-\tau}{T_b}*\frac{A^2}{2}+\frac{\tau}{T_b}*\frac{A^2}{4}=\left(1-\frac{\tau}{2T_b}\right)*\frac{A^2}{2}$$

3.For $t_2-t_1>T_b$

$E(X(t_1)X(t_2))=(A*0)*\frac{1}{4}+(0*A)*\frac{1}{4}+(0*0)*\frac{1}{4}+(A*A)*\frac{1}{4}=\frac{A^2}{4}$

$$R_x(t_1,t_2)=\frac{A^2}{4}$$

$$\therefore R_x(\tau)=\left(1-\frac{\tau}{2T_b}\right)*\frac{A^2}{2}\ for|\tau|<T_b\ and\ \frac{A^2}{4}\ otherwise$$

$$R_x(\tau)=\frac{A^2}{4}+\frac{A^2}{4}\Delta(\frac{\tau}{T_b})$$

➢ The theoretical autocorrelation describes a triangular shape centered at $\tau = 0$, spanning from $\tau = -T_b$ to $\tau = T_b$, with a peak of $A^2/2 = 4^2/2 = 8$.

➢ The triangle, scaled by $A^2/_4$, sits on a constant baseline which is the DC value, for $|\tau| \geq T_b$, the triangle vanishes, leaving only the DC level. that will have a value of

$$\frac{A^2}{4} = \frac{4^2}{4} = 4$$

- **Simulated Autocorrelation:**

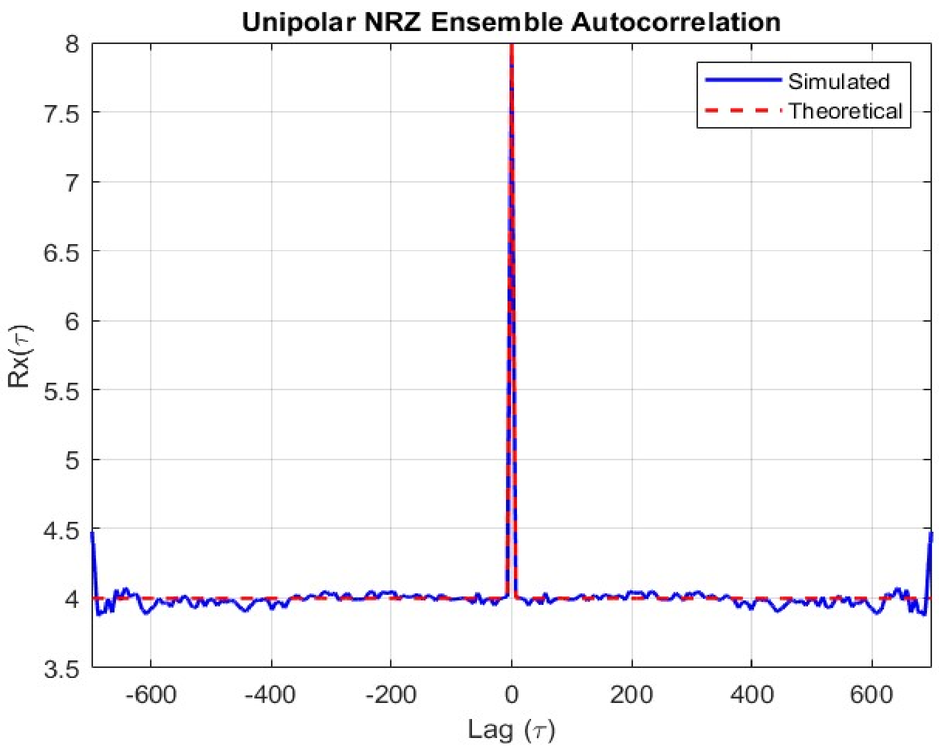


Figure 8.Unipolar NRZ Statistical Autocorrelation

▪ The autocorrelation of Unipolar NRZ is computed using both simulated and theoretical approaches, with the results being very close. The statistical autocorrelation shows that the same bit (at $\tau = 0$) exhibits the highest autocorrelation having amplitude equal [8], while other bits exhibit lower autocorrelation fluctuating around DC value of [4] due to their randomness and independence. This confirms the correctness of the result, as the simulated autocorrelation closely matches the theoretical autocorrelation.

Polar NRZ

- **Theoretical Autocorrelation:**

$E(X(t))=(A) * \frac{1}{2} + (-A) * \frac{1}{2} = 0$

For Autocorrelation we have two parts : $t_2-t_1<T_b$ and $t_2-t_1>T_b$

1.For $t_2-t_1<T_b$ and (t_1,t_2) in the same bit

$E(X(t_1)X(t_2)) = (A * A) * \frac{1}{2} + (-A * -A) * \frac{1}{2} = A^2$

2.For t_2-t_1<T_b and (t_1,t_2) in adjacent bits

$E(X(t_1)X(t_2)) = (A * A) * \frac{1}{4} + (-A * -A) * \frac{1}{4} + (A * -A) * \frac{1}{4} + (-A * A) * \frac{1}{4} = 0$

$R_x(t_1, t_2) = \frac{T_b - \tau}{T_b} * A^2 + \frac{\tau}{T_b} * 0 = \left(1 - \frac{\tau}{T_b}\right) * A^2$

3.For t_2-t_1>T_b

$E(X(t_1)X(t_2)) = (A * A) * \frac{1}{4} + (-A * -A) * \frac{1}{4} + (A * -A) * \frac{1}{4} + \quad (-A * A) * \frac{1}{4} = 0$
$R_x(t_1, t_2) = 0$

$\therefore R_x(\tau) = \left(1 - \frac{\tau}{T_b}\right) * A^2 \; for \; |\tau| < T_b \; and \; 0 \; otherwise$

$R_x(\tau) = A^2 \Delta(\frac{\tau}{T_b})$

- The theoretical autocorrelation describes a triangular shape centered at $\tau = 0$, spanning from $\tau = -T_b$ to $\tau = T_b$, with a peak of $A^2 = 16$.
- The triangle sits on a constant baseline with [0] DC value, and for | τ |≥ T_b , the triangle vanishes and remains a line on 0.

- **Simulated Autocorrelation:**

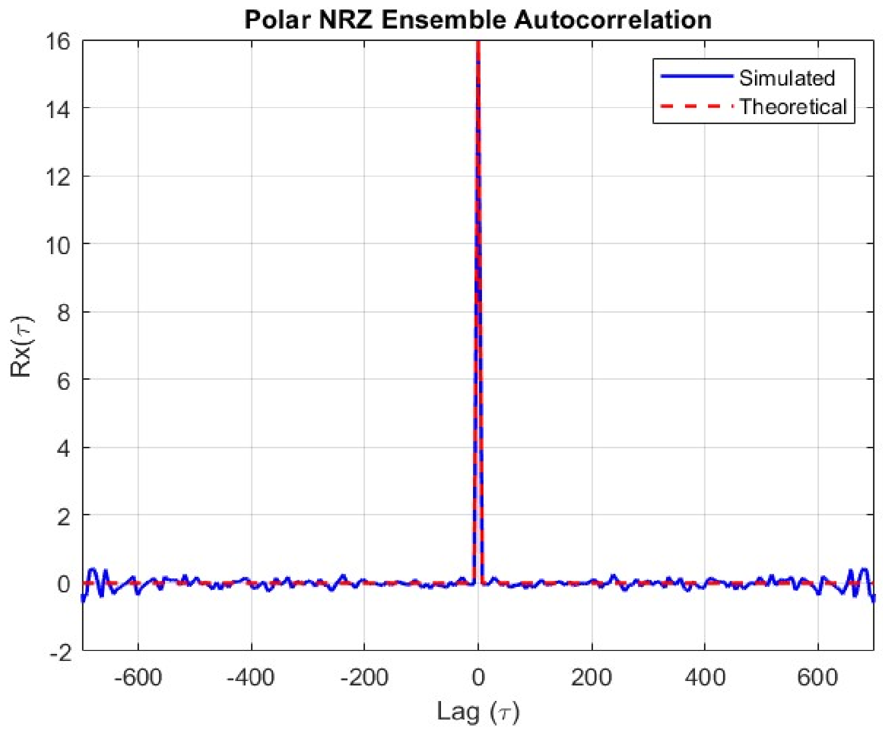


Figure 9.Polar NRZ Statistical Autocorrelation

- The autocorrelation of Polar NRZ is computed using both simulated and theoretical approaches, and the results are very close. The statistical autocorrelation shows that the same bit at (τ = 0) exhibits the highest autocorrelation with amplitude [16] , while other bits exhibit lower autocorrelation fluctuating around [0] due to their randomness and independence. This confirms the correctness of the result, as the simulated autocorrelation closely matches the theoretical autocorrelation.

Polar RZ

- **Theoretical Autocorrelation:**

$x(t) = \begin{cases} \pm A & 0 \le t < DT_b \\ 0 & DT_b \le t < T_b \end{cases}$ where $D = \frac{4}{7}$ $(active\ time)$

$\mathrm{E(X(t))=(A)} * \frac{1}{2} + (-A) * \frac{1}{2} = 0$

For Autocorrelation we have two parts : t_2-$t_1 < DT_b$ and t_2-$t_1 > DT_b$

1.For t_2-$t_1 < DT_p$ and (t_1,t_2) in the same half of bit

$$\mathrm{E(X(t1)X(t2))} = (\mathrm{A} * \mathrm{A}) * \frac{4}{7} + (0 * 0) * \frac{4}{7} = \frac{\mathrm{A}^2 \times 4}{7}$$

2.For t_2-$t_1 < DT_p$ and (t_1,t_2) in adjacent halves

$\mathrm{E(X(t_1)X(t_2))} = (\mathrm{A} * 0) * \frac{1}{4} + (-A * 0) * \frac{1}{4} + (0 * -\mathrm{A}) * \frac{1}{4} + (0 * A) * \frac{1}{4} = 0$

$R_x(\mathrm{t}_1, \mathrm{t}_2) = \frac{DT_b - \tau}{DT_b} * \frac{\mathrm{A}^2 \times 4}{7} + \frac{\tau}{DT_b} * 0 = \left(1 - \frac{\tau}{DT_p}\right) * \frac{\mathrm{A}^2 \times 4}{7}$

3.For t_2-$t_1 > DT_p$

$\mathrm{E(X(t_1)X(t_2))} = (\mathrm{A} * \mathrm{A}) * \frac{1}{16} + (-A * A) * \frac{1}{16} + (\mathrm{A} * -\mathrm{A}) * \frac{1}{16} + (-A * -A) * \frac{1}{16} + (\mathrm{A} * 0) * \frac{1}{8} + (-A * 0) * \frac{1}{8} + (0 * -\mathrm{A}) * \frac{1}{8} + (0 * A) * \frac{1}{8} + (0 * 0) * \frac{1}{4} = 0$

$R_x(\mathrm{t}_1, \mathrm{t}_2) = 0$

$$\therefore R_x(\tau) = \left(1 - \frac{\tau}{DT_b}\right) * \frac{\mathrm{A}^2 \times 4}{7}\ for|\tau| < \mathrm{DT}_b\ and\ 0\ otherwise$$

$R_x(\tau) = \frac{\mathrm{A}^2 \times 4}{7} \Delta(\frac{t}{\mathrm{DT}_b})$

➢ The theoretical autocorrelation describes a triangular shape centered at $\tau = 0$, spanning from $\tau = -DT_b$ to $\tau = DT_b$, with a peak of $\frac{\mathrm{A}^2 \times 4}{7} = \frac{16 \times 4}{7} = 9.12$.

- ➢ The triangle sits on a constant baseline with [0] DC value, and for $| \tau | \geq DT_b$, the triangle vanishes and remains a line on 0.
- **Simulated Autocorrelation:**

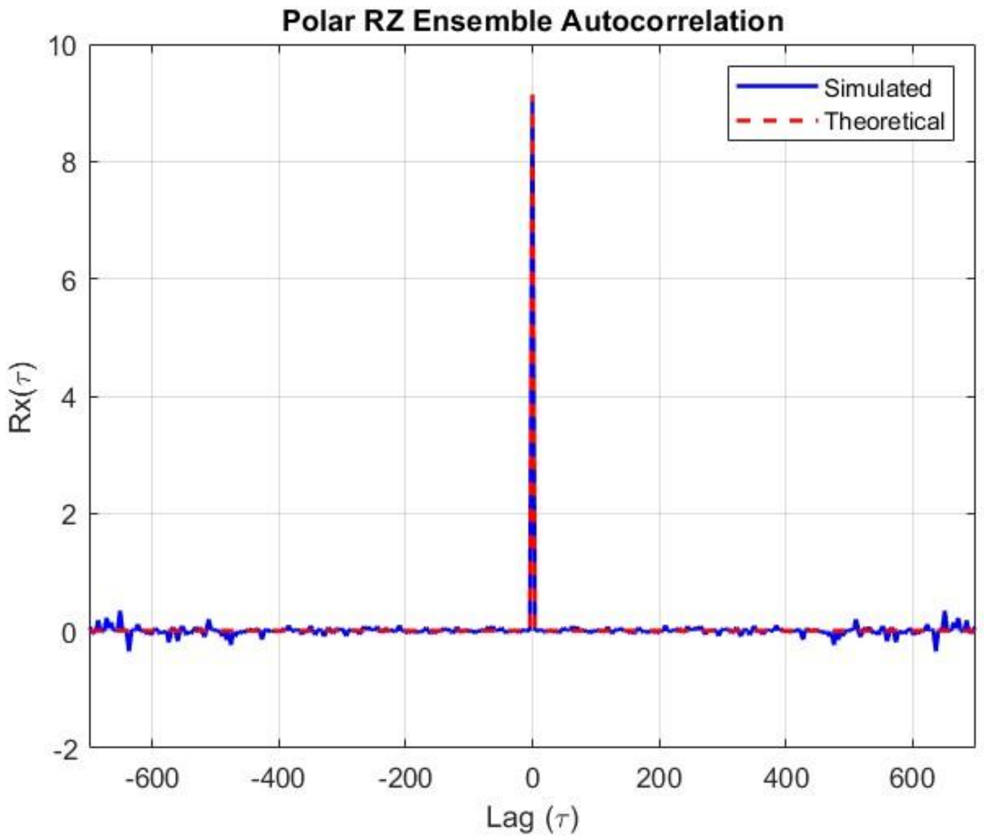


Figure 10.Polar RZ Statistical Autocorrelation

- The autocorrelation of Polar RZ is computed using both simulated and theoretical approaches, and the results are very close. The statistical autocorrelation of Polar RZ shows that the same bit at (τ = 0) exhibits the highest autocorrelation with amplitude 9.12 as expected , while other bits have lower autocorrelation almost [0] due to their randomness and independence. The closeness of the simulated autocorrelation to the theoretical autocorrelation confirms the correctness of the result

Q1] Is this random process stationary?

To ensure wide sense stationarity process, the statistical mean must remain constant, and the autocorrelation should depend on τ rather than time(t).

From our previous work, we verified that the mean remains fluctuating around a constant number with small variation and by increasing the number of realizations to 5000 the variations decreased, and the mean was almost constant.

Now, to ensure dependency on τ, we modified the statistical autocorrelation function above to a function that takes specific time as input, iterates over all τ using a loop, and applies flipping.

RESULTS(1)

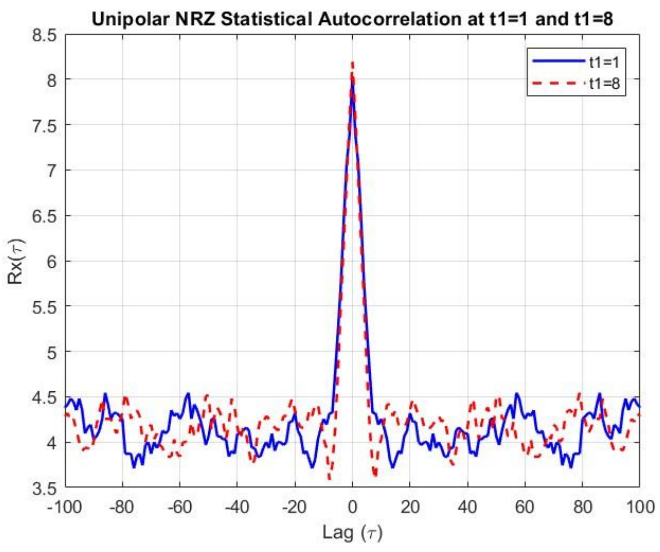


Figure 11. Unipolar NRZ autocorrelation At T1=1 and T1=8

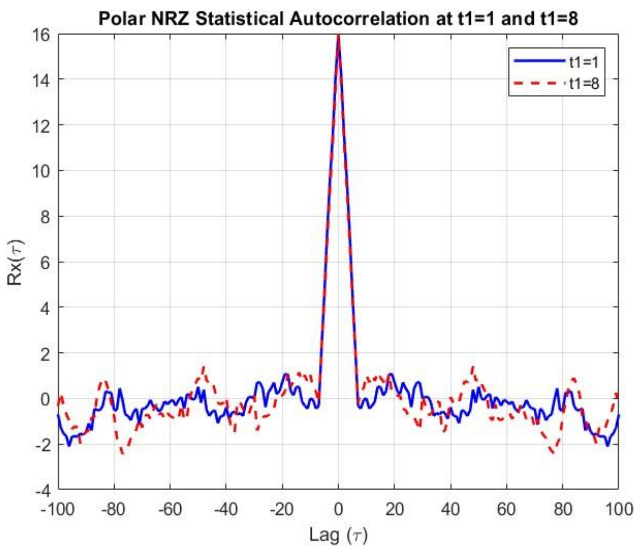


Figure 12. Polar NRZ autocorrelation At T1=1 and T1=8

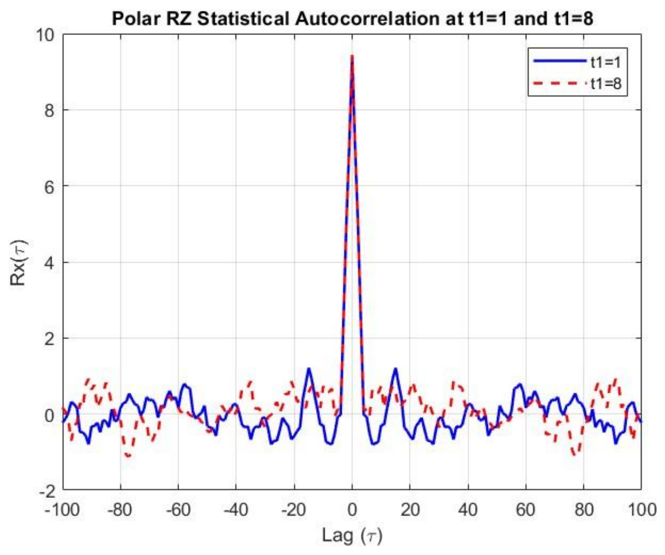


Figure 13. Polar RZ autocorrelation At T1=1 and T1=8

- We simulate at times t = 1 and t = 8 (Note that we can simulate at any time), The output demonstrates that the process may statistically depend on time rather than τ due to small waveform and sample sizes. However, we can still check stationarity because, as the sample size approaches infinity, the averaged statistics converge to specific values like what happened with statistical mean.

Now we will increase the number of waveforms to 5000 to check.

RESULTS(2)

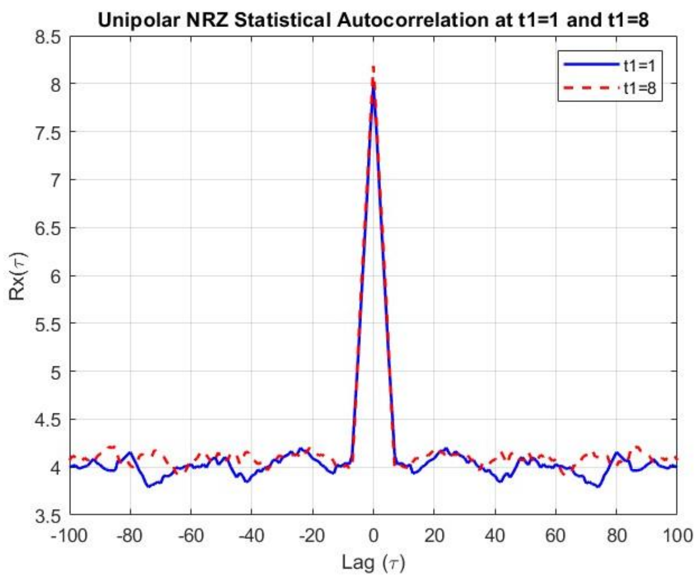


Figure 15. Unipolar NRZ autocorrelation At T1=1 and T1=8 with 5000 waveforms

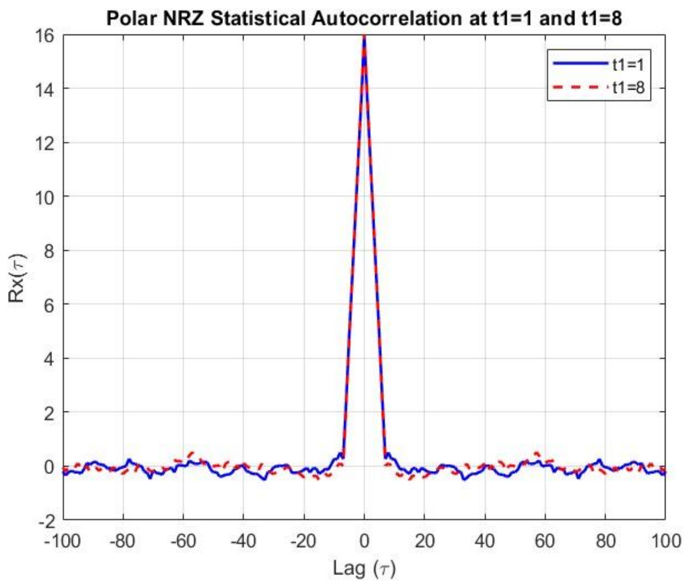


Figure 14. Polar NRZ autocorrelation At T1=1 and T1=8 with 5000 waveforms

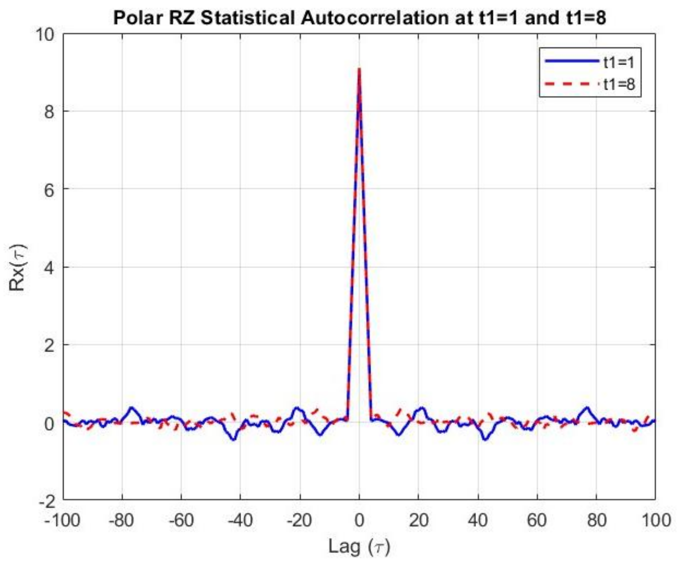


Figure 16. Polar RZ autocorrelation At T1=1 and T1=8 with 5000 waveforms

- We note that after increasing the number of realizations to 5000 the autocorrelation function at t=1, t=8 get closer, Proving the number of realizations affects the results. However, if we could generate an infinite number of realizations, the process would no longer be dependent on time. **So, we can conclude that this process is wide sense stationary.**

Time Mean:

The time mean of a waveform represents the average voltage value over time for a single waveform, assuming a consistent bit duration and amplitude level. Below, the time mean for the first waveform and average time mean of all waveforms is calculated using the following code.

```
% Function to calculate time mean for all waveforms
function time_means = calc_time_means(ensemble, Samples_length)
  % for 1 waveform
  first_waveform = ensemble(1, :);
  time_mean = sum(first_waveform) / Samples_length;
  disp("The time mean of the 1st first waveform " + time_mean);

  %for average
  time_means = sum(ensemble, 2)/Samples_length;
end
```

 Sum 1 waveform: It takes the first waveform from the ensemble, sums all its sample values, then divides by 700 (the number of samples) to get the average voltage over time for it.

 Sum average waveforms: It sums each waveform in the ensemble along time (row-wise), then divides by 700 to get the time mean for every waveform in the ensemble.

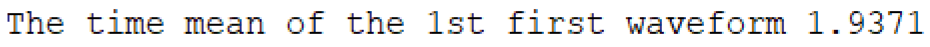
Unipolar NRZ (One waveform)

- **Theoretical Mean:** Since the data bits are equally likely [P(0) = P(1) = 0.5] across all samples, the expected voltage per bit is the average of the possible levels:

$$E[Unipolar\ NRZ] = P(0) \times 0 + P(1) \times A = 0.5 \times 0 + 0.5 \times 4 = 2V$$

- **Simulated Mean:**

For first waveform the MATLAB output was:



```
The time mean of the 1st first waveform 1.9371
```

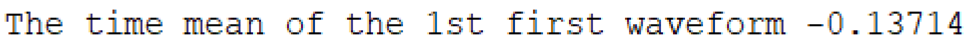
Polar NRZ (One waveform)

- **Theoretical Mean:** With equal probabilities [P(0) = P(1) = 0.5] across all samples in a waveform, the expected voltage per bit balances the positive and negative levels:

$$E[Polar\ NRZ] = P(0) \times (-A) + P(1) \times A = 0.5 \times (-4) + 0.5 \times 4 = -2 + 2 = 0V$$

- **Simulated Mean:**

For first waveform the MATLAB output was:



```
The time mean of the 1st first waveform -0.13714
```

Polar RZ (One waveform)

- **Theoretical Mean:** The mean depends on the sample position within a bit.

$$E[Polar\ RZ] = \frac{4 \times 0 + 3 \times 0}{7} = 0V$$

- **Simulated Mean:**

For first waveform the MATLAB output was

```
The time mean of the 1st first waveform -0.045714
```

- It is clear from the results that the theoretical mean remains the same whether we compute it across columns (statistical mean) or across rows (time mean).
- The simulation output for a single waveform showed that the time mean is not exactly equal to the theoretical value, but it is very close, similar to the statistical mean which has its value fluctuating around the theoretical value. This slight difference is due to the limited number of samples in the waveform.

Now that we have examined the output for a single waveform, we will move on to viewing the output for all 500 waveforms.

Unipolar NRZ (All waveforms)

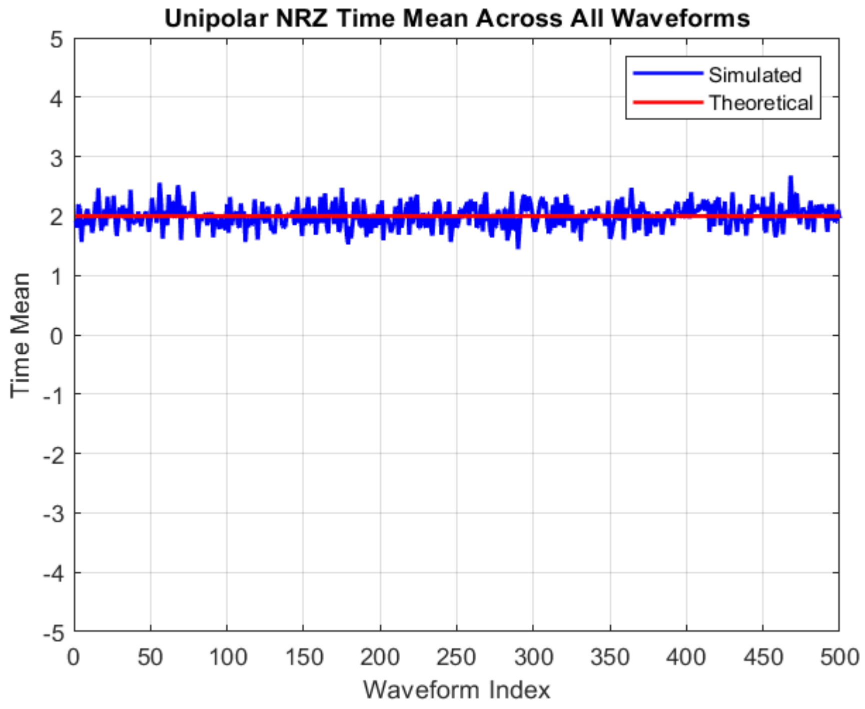


Figure 17. Unipolar NRZ time mean

- The time mean of unipolar NRZ is computed using both simulated and theoretical approaches, showing a strong correlation. The mean at each point remains fluctuating around 2 V, as random data evenly distribute active and zero periods across the 500 waveforms.

Polar NRZ (All waveforms)

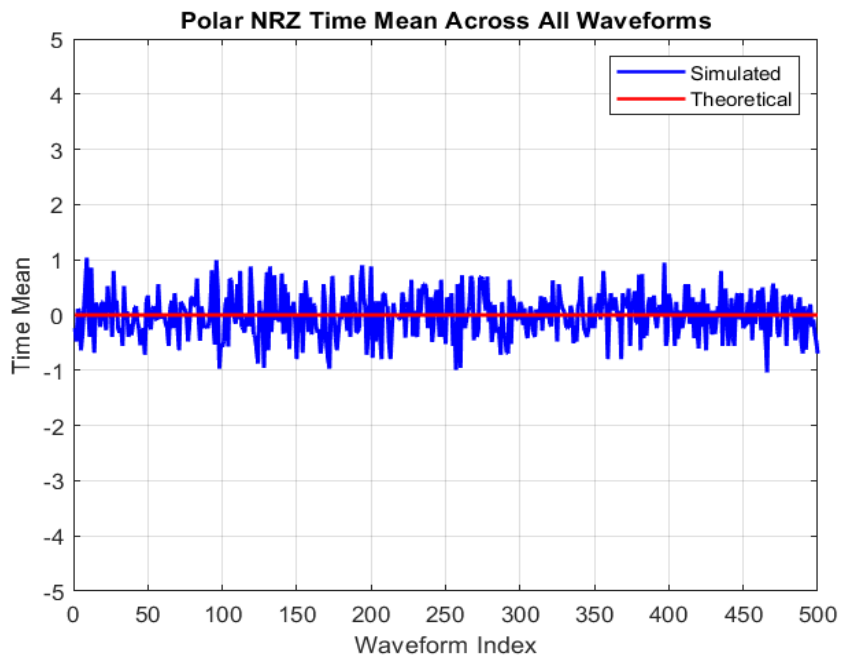


Figure 18. Polar NRZ time mean

- The time mean of Polar NRZ is computed using both simulated and theoretical approaches, showing a strong correlation. The mean at each point remains fluctuating around 0 V, as random data evenly distributes +ve and -ve active periods across the 500 waveforms.

Polar RZ (All waveforms)

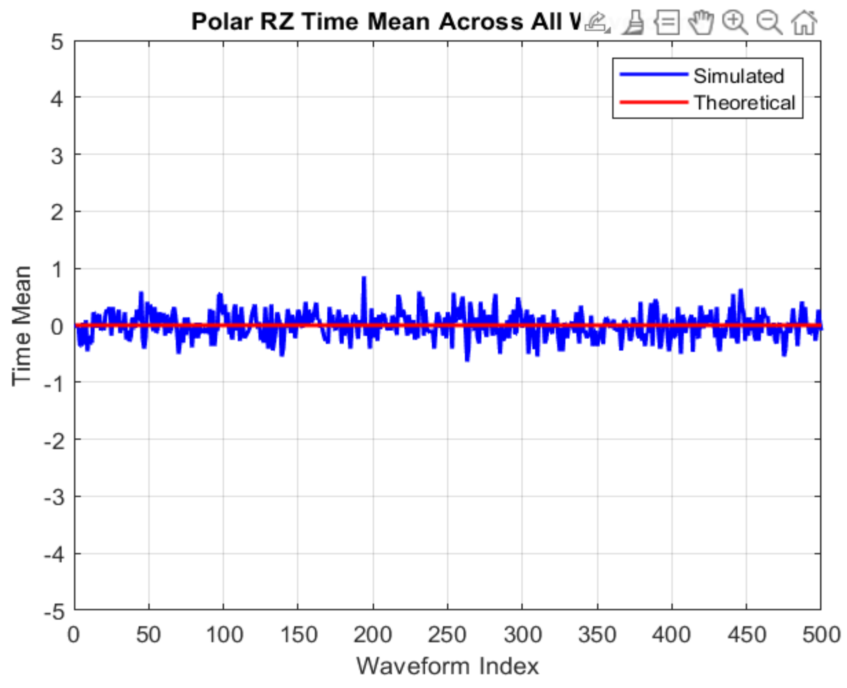


Figure 19. Polar RZ time mean

- The time mean of Polar RZ is computed using both simulated and theoretical approaches, showing a strong correlation. The mean at each point remains fluctuating around 0 V, as random data evenly distributes +ve , -ve active and zero periods across the 500 waveforms.

Time Autocorrelation:

Time autocorrelation measures how the values of a **single waveform** relate to themselves at different time delays, showing how the signal correlates with its shifted version over time.

```
% Function to calculate time autocorrelation Rx(tau) for one waveform
function Rx_time = calc_time_autocorr(signal, sample_length,max_tau)
    shifted_matrix = zeros(sample_length, sample_length);
    % Create circularly shifted versions of the input signal
    for i = 1:sample_length
        shifted_matrix(i, :) = [signal(i:end), signal(1:i-1)];
    end
    % Compute time-domain autocorrelation
    autocorr_positive = zeros(1, max_tau + 1);  % Initialize positive-lag autocorrelation
vector
    for t = 1:sample_length
        autocorr_positive(t) = sum(shifted_matrix(1,:) .* shifted_matrix(t,:)) /
sample_length;
    end

    % Mirror the autocorrelation to include negative lags (symmetric autocorrelation)
    Rx_time = [flip(autocorr_positive(2:end))'; autocorr_positive'];
end
```

 Pre-allocate shifted matrix: The first line creates a square matrix to store circularly shifted versions of the input signal, with each row holding a shifted copy for correlation calculations.

 Loop to create shifted signals: The second line iterates over indices to generate circular shifts of the signal. Each row [i] of shifted_matrix contains the signal shifted by [i-1] positions, wrapping around using [signal(i:end), signal(1:i-1)].

 Pre-allocate autocorrelation array: like before it sets up an array to store the autocorrelation for non-negative lags (0 to max_tau), with +1 for including tau = 700.

 Loop over lags for correlation: This line iterates through lags (up to max_tau). For each [t], it computes the correlation between the original signal and its [t-1] shifted version.

 Calculate correlation: This line multiplies the original signal (shifted_matrix(1,:)) with its [t-1] shifted version, sums the products, and normalizes to get the autocorrelation at lag t-1.

 Mirror for negative lags: Lastly, we create the full autocorrelation by taking positive lags (excluding tau = 0), flipping them for negative lags, and concatenating with positive lags to form a symmetric output Rx_time.

Unipolar NRZ (One waveform)

- **Theoretical Autocorrelation:** Just as the statistical the time autocorrelation will be:

$$R_x(\tau) = \frac{A^2}{4} + \frac{A^2}{4}\Delta(\frac{t}{T_b})$$

- **Simulated Autocorrelation:**

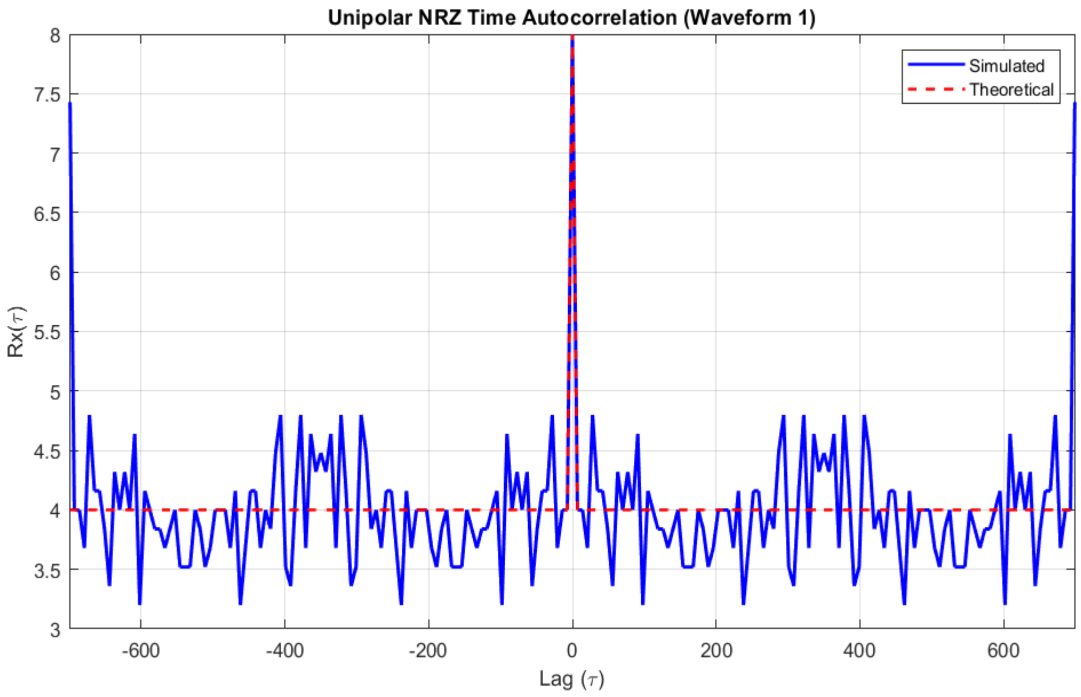


Figure 20. Unipolar NRZ Time Autocorrelation

- The Time Autocorrelation of unipolar NRZ 1st waveform is computed using both simulated and theoretical approaches. The Autocorrelation within the $-T_p \leq \tau \leq T_p$, matches exactly the theoretical part approaching the highest amplitude at $\tau = 0$ as expected, but beyond T_p point remains fluctuating around DC value due to the finite length and randomness of the bit sequence, unlike the stable DC in theory.
- Sometimes, the entire graph shifts upward or downward with an extra DC offset if the waveform has more 1s than 0s or vice versa, pushing the mean above or below 2v, which alters the expected DC level of 4 in the Autocorrelation simulation.

Polar NRZ (One waveform)

- **Theoretical Autocorrelation:** Just as the statistical the time autocorrelation will be:

$$R_x(\tau) = A^2\Delta(\frac{t}{T_b})$$

- **Simulated Autocorrelation:**

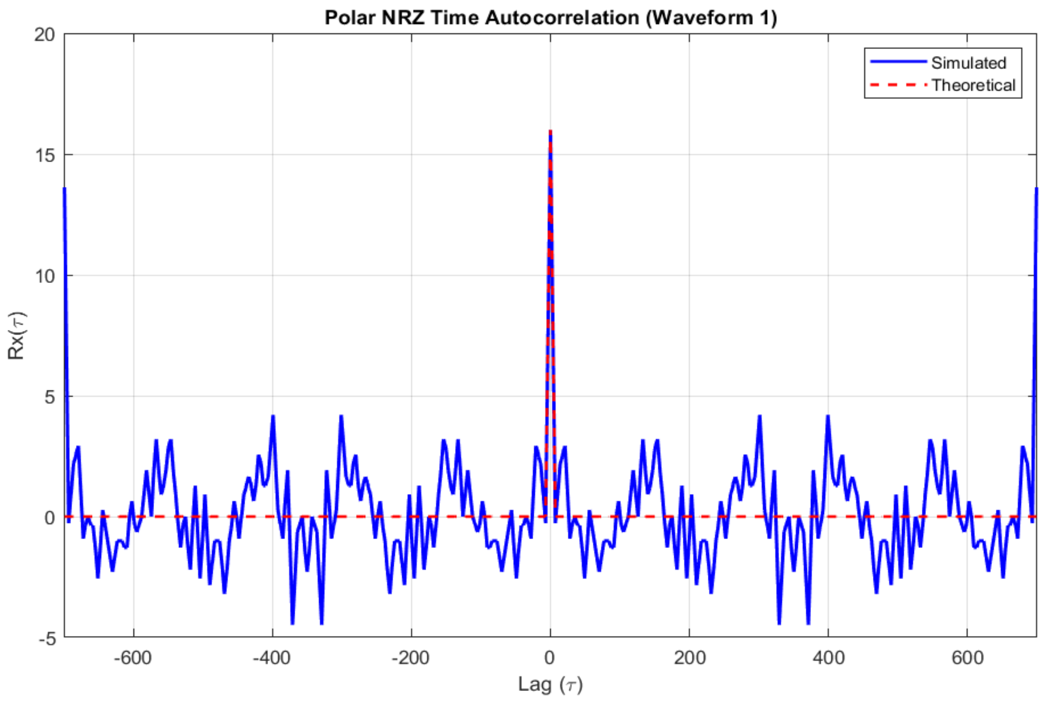


Figure 21. Polar NRZ Time Autocorrelation

Polar RZ (One waveform)

- **Theoretical Autocorrelation:** Just as the statistical the time autocorrelation will be:

$$R_x(\tau) = \frac{\mathrm{A}^2 \times 4}{7} \Delta(\frac{t}{\mathrm{D}T_b})$$

- **Simulated Autocorrelation:**

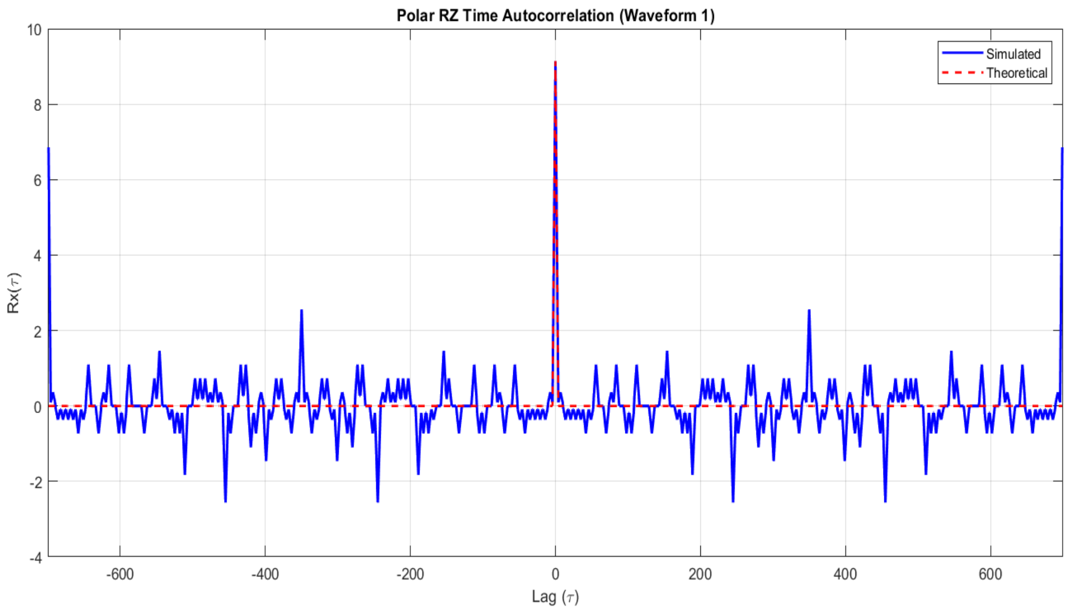


Figure 22. Polar RZ Time Autocorrelation

- For both polar NRZ and RZ 1st waveform. The Autocorrelation within the $-DT_p \leq \tau \leq DT_p$,

$$\text{where } D = \begin{cases} 1 & \text{Polar NRZ} \\ 4/7 & Polar\ RZ \end{cases}$$

  matches exactly the theoretical part approaching the highest amplitude at $\tau = 0$ as expected, but beyond T_p point remains fluctuating around [0] due to the finite length and randomness of the bit sequence.

Note: Circular convolution was used in this case to handle the missing sample values when using time delays (τ) greater than 1. So that the signal wraps around instead of padding with zeros.

Q2] Is this random process Ergodic?

For a random process to be ergodic, the time mean, and autocorrelation must match their statistical counterparts.

Based on our explanations and simulation results, we observed that the statistical and time means both vary within the same voltage range around the theoretical mean, making them almost identical.

For the autocorrelation, both statistical and time-based results have similar shapes and match the theoretical autocorrelation within the range $-D \cdot T_p \leq \tau \leq D \cdot T_p$. Outside this range, both fluctuate around a DC value. Although the statistical autocorrelation appears to fluctuate less than the time autocorrelation, it's important to note that the statistical autocorrelation is averaged over 500 waveforms, while the time autocorrelation was calculated using only one waveform.

So, we can conclude that yes, this process is ergodic.

PSD Calculation:

The PSD is the Fourier transform of the autocorrelation function.

```
% PSD from ensemble autocorrelation
Ts = 0.01;
fs = 1 / Ts; % 100 Hz
N = length(Rx); % 1399 with max_tau = 699
psd = abs(fftshift(fft(Rx))) / fs; % No padding, use full Rx
```

- **Calculate sampling frequency**: fs = 1 / Ts computes the sampling frequency(the rate at which the signal is sampled), which will be 100Hz based on given Ts of DAC.

- **Get autocorrelation length**: This line determines the length of the autocorrelation array Rx, which is 1399 (since $max\_tau = 699, so\ N = 2 \times 699 + 1$), including negative and positive lags plus zero).

- **Compute PSD**: Lastly, Taking the FFT of Rx to transform it to the frequency domain,

Unipolar NRZ

- **Theoretical PSD:**

1. Apply the Fourier Transform:

The PSD $Sx(f)$ is the Fourier Transform of the autocorrelation function Rx(τ):

$$Sx(f) = FT\{Rx(\tau)\}$$

$$Sx(f) = FT\{\frac{A^2}{4}\} + FT\{\frac{A^2}{4}\Delta(\frac{t}{T_b})\}$$

2. PSD Expression:

$$Sx(f) = \frac{A^2}{4} \cdot \delta(f) + \frac{A^2}{4} . T_b \cdot (sinc(fT_b))^2$$

> ➢ From the PSD expression we can tell that there is a delta component at f = 0 for a unipolar NRZ waveform along with the $(sinc(fT_b))^2$ term

- **Simulated PSD:**

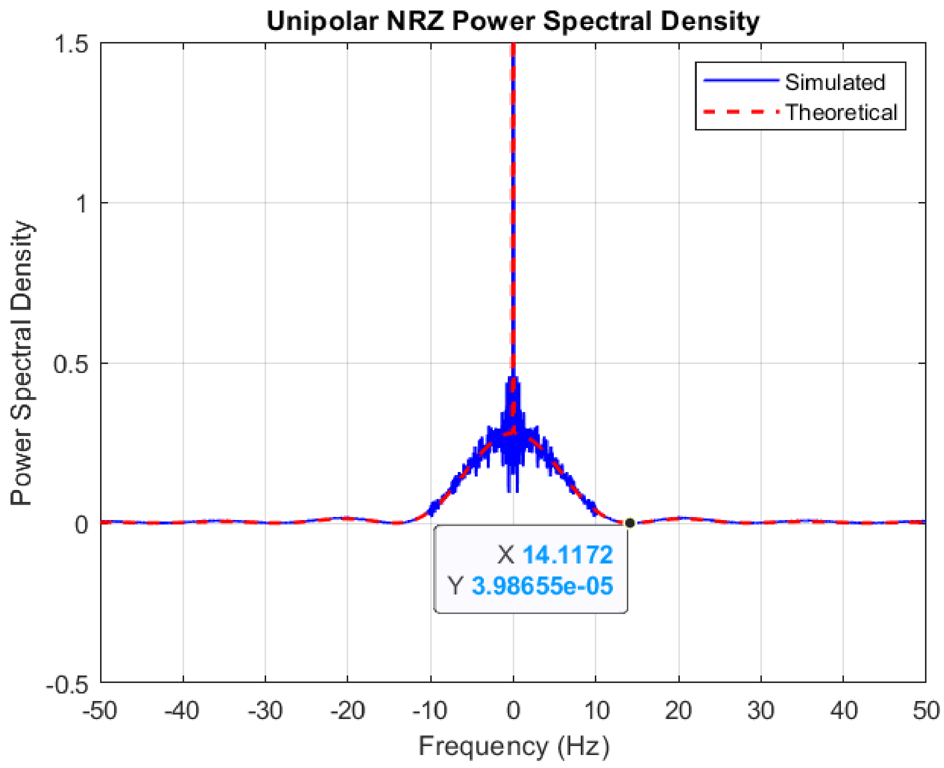


Figure 23. Unipolar NRZ PSD

- The PSD of Unipolar NRZ is computed using both simulated and theoretical approaches, As shown, the actual PSD closely matches the theoretical PSD.

Q3] What is the bandwidth of Unipolar NRZ transmitted signal?

- Bandwidth (BW) of the transmitted signal is defined as the first zero in the PSD (First zero of $sinc(x)$ function).
- From the above graph we can view that the first zero happen at 14.12 Hz.
- Theoritical bandwidth $= \frac{1}{7*T_s} = \frac{1}{T_b} = \frac{1}{0.07} = 14.258$ (Very close to simulated BW)

Polar NRZ

- **Theoretical PSD:**

1. Apply the Fourier Transform:

The PSD $Sx(f)$ is the Fourier Transform of the autocorrelation function Rx(τ):

$$Sx(f) = FT\{Rx(\tau)\}$$

$$Sx(f) = FT\{A^2 \times \Delta(\frac{t}{T_b})\}$$

2. PSD Expression:

$$Sx(f) = A^2.T_b \cdot (sinc(fT_b))^2$$

- **Simulated PSD:**

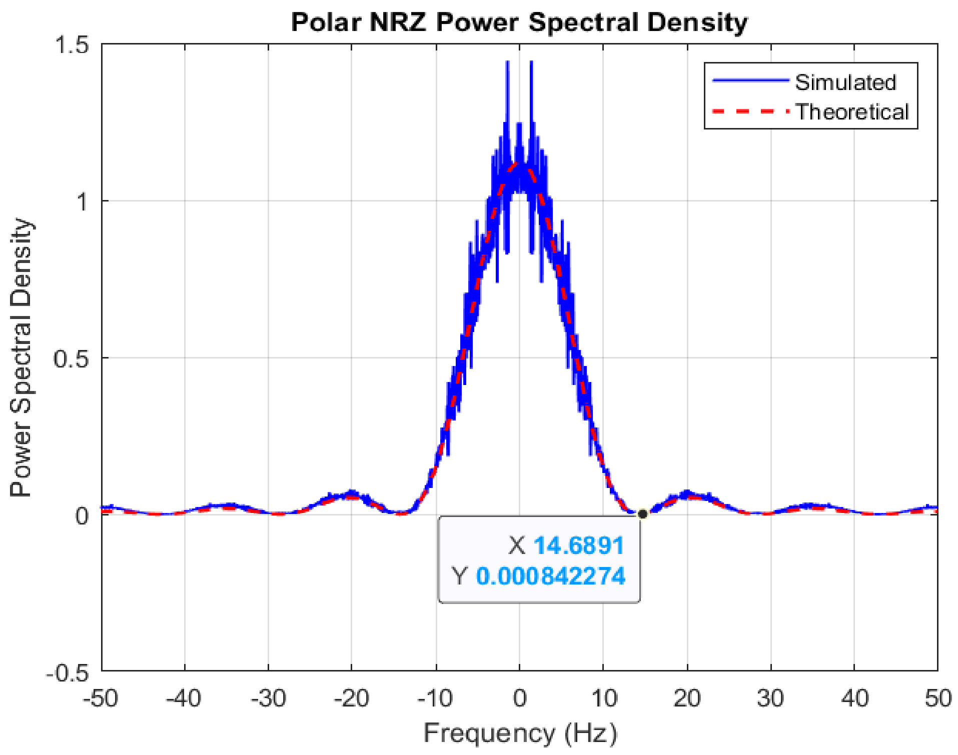


Figure 24. Polar NRZ PSD

- The PSD of Polar NRZ is computed using both simulated and theoretical approaches, As shown, the actual PSD closely matches the theoretical PSD.

Q4] What is the bandwidth of Polar NRZ transmitted signal?

- Bandwidth (BW) of the transmitted signal is defined as the first zero in the PSD (First zero of $sinc(x)$ function).
- From the above graph we can see that the first zero happens at 14.68 Hz.
- Theoritical bandwidth $= \frac{1}{7*T_s} = \frac{1}{T_b} = \frac{1}{0.07} =$ 14.258 (Very close to simulated BW)

Polar RZ

- **Theoretical PSD:**

1. Apply the Fourier Transform:

The PSD $Sx(f)$ is the Fourier Transform of the autocorrelation function Rx(τ):

$$Sx(f) = FT\{Rx(\tau)\}$$

$$Sx(f) = FT\{\frac{A^2 \times 4}{7} \times \Delta(\frac{t}{DT_b})\}$$

2. PSD Expression:

$$Sx(f) = \frac{A^2 \times 4}{7} . DT_b \cdot (sinc(f \times DT_b))^2$$

- **Simulated PSD:**

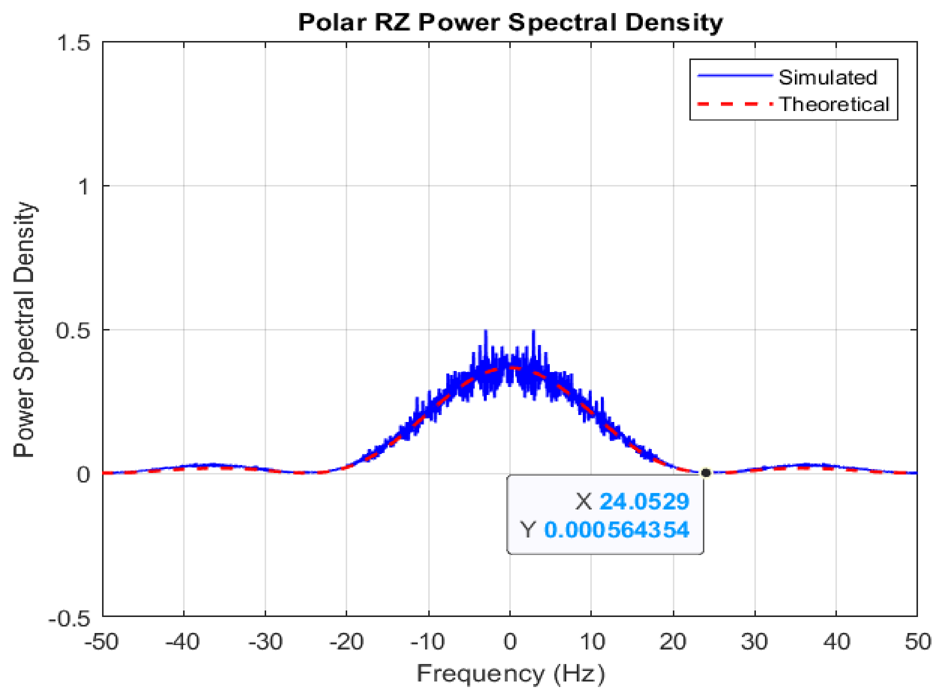


Figure 25. Polar RZ PSD

- The PSD of Polar NRZ is computed using both simulated and theoretical approaches, as shown, the actual PSD closely matches the theoretical PSD.

Q5] What is the bandwidth of Polar RZ transmitted signal?

- Bandwidth (BW) of the transmitted signal is defined as the first zero in the PSD (First zero of $sinc(x)$ function).
- From the above graph we can see that the first zero happens at 24.05 Hz.
- Theoritical bandwidth= $\frac{1}{DT_b} = \frac{7}{4\times 0.07}$ = 25 (Very close to simulated BW)

Full MATLAB Code:

```
% Initialize Parameters
num_waveforms = 500;
Amplitude = 4;
samples_per_bit = 7;
half_bit_duration = 4;
Samples_length = 700;
num_bits_with_delay = 101;

% Matrix to store waveforms
ensemble_unipolar_nrz = zeros(num_waveforms, Samples_length);
ensemble_polar_nrz = zeros(num_waveforms, Samples_length);
ensemble_polar_rz = zeros(num_waveforms, Samples_length);

% Generate Waveforms Line code and delay
for i = 1:num_waveforms
    data = randi([0 1], 1, num_bits_with_delay); % choose random number 0 or 1

    % Unipolar NRZ Signal
    unipolar_signal_nrz = data * Amplitude;
    unipolar_nrz_waveform = repmat(unipolar_signal_nrz, samples_per_bit, 1);
    unipolar_nrz_waveform = reshape(unipolar_nrz_waveform, 1, []);

    % polar NRZ Signal
    polar_signal = ((2 * data) - 1) * Amplitude;
    polar_nrz_waveform = repmat(polar_signal, samples_per_bit, 1);
    polar_nrz_waveform = reshape(polar_nrz_waveform, 1, []);

    % polar RZ Signal

    polar_rz_waveform = zeros(samples_per_bit, num_bits_with_delay); % Pre-allocation
to avoid warning and set zeros for last 3 samples
    polar_rz_waveform(1:half_bit_duration, :) = repmat(polar_signal,
half_bit_duration, 1); % first 4 samples are polar signal , last 3 samples returns to
zero
    polar_rz_waveform = reshape(polar_rz_waveform, 1, []);

    random_delay = randi([1 7]); % choose random number from 1 to 7
    % Shift waveforms to add random delay
    delayed_unipolar_nrz = unipolar_nrz_waveform(1 + random_delay : end);
    delayed_polar_nrz = polar_nrz_waveform(1 + random_delay : end);
    delayed_polar_rz = polar_rz_waveform(1 + random_delay : end);

    % Trim excess samples more than 700
    if length(delayed_unipolar_nrz) > Samples_length
       delayed_unipolar_nrz = delayed_unipolar_nrz(1:Samples_length);
    end

    if length(delayed_polar_nrz) > Samples_length
       delayed_polar_nrz = delayed_polar_nrz(1:Samples_length);
    end

    if length(delayed_polar_rz) > Samples_length
       delayed_polar_rz = delayed_polar_rz(1:Samples_length);
    end

    ensemble_unipolar_nrz(i, :) = delayed_unipolar_nrz;
    ensemble_polar_nrz(i, :) = delayed_polar_nrz;
    ensemble_polar_rz(i, :) = delayed_polar_rz;
end
% User input to choose waveform type (unchanged)
disp('Choose a linecode type to view:');
disp('1. Unipolar NRZ');
disp('2. Polar NRZ');
disp('3. Polar RZ');
choice = input('Enter the number corresponding to your choice (1-3): ');

switch choice
    case 1
        plotWaveforms(ensemble_unipolar_nrz, 'Unipolar NRZ', Amplitude,
samples_per_bit);
        plotStats(ensemble_unipolar_nrz, 'Unipolar NRZ', Amplitude, samples_per_bit,
num_waveforms, Samples_length);
    case 2
        plotWaveforms(ensemble_polar_nrz, 'Polar NRZ', Amplitude, samples_per_bit);
        plotStats(ensemble_polar_nrz, 'Polar NRZ', Amplitude, samples_per_bit,
num_waveforms, Samples_length);
    case 3
        plotWaveforms(ensemble_polar_rz, 'Polar RZ', Amplitude, samples_per_bit);
        plotStats(ensemble_polar_rz, 'Polar RZ', Amplitude, samples_per_bit,
num_waveforms, Samples_length);
    otherwise
        disp('Invalid choice! Please run again and enter a number between 1 and 3.');
end
% Function to plot waveforms
function plotWaveforms(ensemble, type, Amplitude, samples_per_bit)
    view_time = 10 * samples_per_bit + 1; % view excatly 70 samples can extend up to
700
    num_plots = 3;

    figure;
    for i = 1:num_plots
        subplot(num_plots, 1, i);
        stairs(0:view_time-1, ensemble(i, 1:view_time), 'LineWidth', 1.5);
        xlabel('Sample Index');
        ylabel('Voltage');
        title([type, ' Waveform ', num2str(i)]);
        if strcmp(type, 'Unipolar NRZ') % comapre type of line code chose by user to
"unipolar" to decided amplitude
            ylim([-1, Amplitude + 1]);
        else
            ylim([-Amplitude - 1, Amplitude + 1]);
        end
        grid on;
    end
end

% Function to calculate statistical mean (ensemble mean at each sample point)
function stat_mean = calc_statistical_mean(ensemble, num_waveforms)
    stat_mean=sum(ensemble, 1)/num_waveforms;
end

% Function to calculate time mean for one and all waveforms
function time_means = calc_time_means(ensemble, Samples_length)
  % for 1 waveform
  first_waveform = ensemble(1, :);
  time_mean = sum(first_waveform) / Samples_length;
  disp("The time mean of the 1st first waveform " + time_mean);

  %for average
  time_means = sum(ensemble, 2)/Samples_length;
end

% Function to calculate ensemble autocorrelation Rx(tau)
function Rx = calc_ensemble_autocorr(ensemble, num_waveforms, Samples_length,
max_tau)
    Rx_pos = zeros(1, max_tau + 1);
    for tau = 0:max_tau
        sum_val = 0;
        for t = 1:(Samples_length - tau)
            for i = 1:num_waveforms    % we can remove this loop and use sum_val =
sum_val + sum(ensemble(:, t) .* ensemble(:, t+tau));
                sum_val = sum_val + ensemble(i, t) * ensemble(i, t + tau);
            end
        end
        Rx_pos(tau + 1) = sum_val / (num_waveforms * (Samples_length - tau));
    end
    Rx = [flip(Rx_pos(2:end)), Rx_pos]; % Mirror for negative lags
end

% Function to calculate statistical autocorrelation at a specific time t1
function Rx_stat_t = calc_stat_autocorr_at_t(ensemble, num_waveforms, Samples_length,
max_tau, t1)
    Rx_stat_t_pos = zeros(1, max_tau + 1);
    for tau = 0:max_tau
        if t1 + tau <= Samples_length  % Check bounds for positive tau
            sum_val = 0;
            for i = 1:num_waveforms
                sum_val = sum_val + ensemble(i, t1) * ensemble(i, t1 + tau);
            end
            Rx_stat_t_pos(tau + 1) = sum_val / num_waveforms;
        else
            Rx_stat_t_pos(tau + 1) = 0; % Out of bounds
        end
    end
    Rx_stat_t = [flip(Rx_stat_t_pos(2:end)), Rx_stat_t_pos]; % Mirror for negative
lags
end

% Function to calculate time autocorrelation Rx(tau) for one waveform
function Rx_time = calc_time_autocorr(signal, sample_length,max_tau)
    shifted_matrix = zeros(sample_length, sample_length);
    % Create circularly shifted versions of the input signal
    for i = 1:sample_length
        shifted_matrix(i, :) = [signal(i:end), signal(1:i-1)];
    end
    % Compute time-domain autocorrelation
    autocorr_positive = zeros(1, max_tau + 1);  % Initialize positive-lag
autocorrelation vector
    for t = 1:sample_length
        autocorr_positive(t) = sum(shifted_matrix(1,:) .* shifted_matrix(t,:)) /
sample_length;
    end

    % Mirror the autocorrelation to include negative lags (symmetric autocorrelation)
    Rx_time = [flip(autocorr_positive(2:end))'; autocorr_positive'];
end

% Function to plot all statistics
function plotStats(ensemble, type, Amplitude, samples_per_bit, num_waveforms,
Samples_length)
    max_tau = 699;
    tau = -max_tau:max_tau;

    % Statistical mean
    stat_mean = calc_statistical_mean(ensemble, num_waveforms);

    % Time means for all waveforms
    time_means = calc_time_means(ensemble, Samples_length);

    % Ensemble autocorrelation
    Rx = calc_ensemble_autocorr(ensemble, num_waveforms, Samples_length, max_tau);

    % Time autocorrelation (single waveform)
    Rx_time = calc_time_autocorr(ensemble(1, :), Samples_length, max_tau);

    % Statistical autocorrelation at t1 = 1 and t1 = 8 (MATLAB indices start at 1)
    Rx_stat_t1 = calc_stat_autocorr_at_t(ensemble, num_waveforms, Samples_length,
max_tau, 1);
    Rx_stat_t8 = calc_stat_autocorr_at_t(ensemble, num_waveforms, Samples_length,
max_tau, 8);

    % Theoretical values
    if strcmp(type, 'Unipolar NRZ')
```

```
        theo_mean = Amplitude / 2;
        theo_Rx = (Amplitude^2 / 4) * (1 - abs(tau) / samples_per_bit) .* (abs(tau)
<= samples_per_bit) + (Amplitude/2)^2;
    elseif strcmp(type, 'Polar NRZ')
        theo_mean = 0;
        theo_Rx = Amplitude^2 * (1 - abs(tau) / samples_per_bit) .* (abs(tau) <=
samples_per_bit);
    else % Polar RZ
        theo_mean = 0;
        theo_Rx = (Amplitude^2 * 4 / samples_per_bit) * (1 - abs(tau) / 4) .*
(abs(tau) <= 4);
    end
    % PSD from ensemble autocorrelation
    Ts = 0.01;
    Tb = samples_per_bit * Ts; % 0.07 s
    fs = 1 / Ts; % 100 Hz
    N = length(Rx); % 1399 with max_tau = 699
    f = (-N/2:N/2-1) * (fs / N); % Frequency axis in Hz
    delta_f = zeros(size(f));
    [~,idx0]= min(abs(f));        % Find index where f ≈ 0
    psd = abs(fftshift(fft(Rx))) / fs; % No padding, use full Rx
    if strcmp(type, 'Unipolar NRZ')
        tho_psd = (Amplitude^2 / 4) * Tb * (sinc(f * Tb)).^2; % Main term
        delta_f(idx0) = (Amplitude^2 )/4;
        tho_psd = tho_psd + delta_f;
    elseif strcmp(type, 'Polar NRZ')
        tho_psd = Amplitude^2 * Tb * (sinc(f * Tb)).^2;
        tho_psd = tho_psd + delta_f;
    else % Polar RZ
        Tp = 4 * Ts; % 0.04 s, pulse width
        tho_psd = (Amplitude^2 *4/7) * 4/7 *Tb * (sinc(f * Tp)).^2; % Duty cycle
adjusted
        tho_psd = tho_psd + delta_f;
    end

    % Plotting
    figure('Color', 'w');
    plot(1:Samples_length, stat_mean, 'b', 'LineWidth', 1.5);
    hold on;
    plot(1:Samples_length, theo_mean * ones(1, Samples_length), 'r', 'LineWidth',
1.5);
    xlabel('Sample Index'); ylabel('Statistical Mean');
    ylim([-Amplitude - 1, Amplitude + 1]);
    title([type, ' Statistical Mean (Ensemble)']);
    legend('Simulated', 'Theoretical');
    grid on;

    figure('Color', 'w');
    plot(1:num_waveforms, time_means, 'b', 'LineWidth', 1.5);
    hold on;
    plot(1:num_waveforms, theo_mean * ones(1, num_waveforms), 'r', 'LineWidth', 1.5);
    xlabel('Waveform Index'); ylabel('Time Mean');
    ylim([-Amplitude - 1, Amplitude + 1]);
    title([type, ' Time Mean Across All Waveforms']);
    legend('Simulated', 'Theoretical');
    grid on;

    figure('Color', 'w');
    plot(tau, Rx, 'b', 'LineWidth', 1.5);
    hold on;
    plot(tau, theo_Rx, 'r--', 'LineWidth', 1.5);
    xlabel('Lag (\tau)'); ylabel('Rx(\tau)');
    title([type, ' Ensemble Autocorrelation']);
    legend('Simulated', 'Theoretical');
    grid on;
    xlim([-700, 700]);

    figure('Color', 'w');
    plot(tau, Rx_time, 'b', 'LineWidth', 1.5);
    hold on;
    plot(tau, theo_Rx, 'r--', 'LineWidth', 1.5);
    xlabel('Lag (\tau)'); ylabel('Rx(\tau)');
    title([type, ' Time Autocorrelation (Waveform 1)']);
    legend('Simulated', 'Theoretical'); grid on;
    xlim([-700, 700]);

    figure('Color', 'w');
    plot(tau, Rx_stat_t1, 'b', 'LineWidth', 1.5);
    hold on;
    plot(tau, Rx_stat_t8, 'r--', 'LineWidth', 1.5);
    xlabel('Lag (\tau)'); ylabel('Rx(\tau)');
    title([type, ' Statistical Autocorrelation at t1=1 and t1=8']);
    legend('t1=1', 't1=8'); grid on;
    xlim([-100, 100]);

    % PSD Plot
    f = (-N/2:N/2-1) * (fs / N); % Frequency axis in Hz
    figure('Color', 'w');
    plot(f, psd, 'b', 'LineWidth', 1);
    hold on;
    plot(f, tho_psd, 'r--', 'LineWidth', 1.5);
    xlabel('Frequency (Hz)');
    ylabel('Power Spectral Density');
    legend('Simulated', 'Theoretical');
    title([type, ' Power Spectral Density']);
    xlim([-50, 50]);
    ylim([-0.5,1.5])
    grid on;
end
```