Faculty of Engineering

Cairo University

Digital Communication Course

Project (1)

Presented to:

<u>Dr. Mohamed Nafie</u>

<u>TA: Eng/ Mohamed Khaled</u>

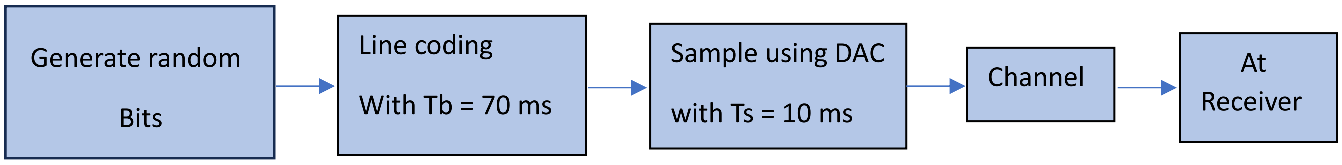
ID	B.N.	Sec.	الإسم
9213073	16	1	أحمد عادل يونس سيد
9210688	3	3	علي مختار علي الدهشوري
9210824	22	3	فيلوباتير عوني عبدالله صليب
9211295	34	4	نور الدين تميم محمد محمد
9170663	-	3	عمرو مصطفي كمال محمد

Table of Contents

0. Role of each member:

- **Each member in the team writes its own code then we compare all these codes to take the best code that provides the best functionality.**

I. Problem description:



- We have a system comprising a transmitter (T_x) and a receiver (R_x) connected via a channel. We generate three ensembles, each employing a different line code (unipolar, polar NRZ, polar RZ). Each ensemble contains multiple realizations where T_x generates random bits. These bits are then represented using continuous signals, but since MATLAB operates with arrays and vectors, we discretize these signals using a Digital to Analog Converter (DAC). The DAC samples the signal with a sampling period of 10 ms, while each bit duration in the line code is 70 ms, resulting in 7 samples per bit.
- The objective is to transmit these bits through the channel, with limited bandwidth which may interfere on the transmitted bits. At the receiver, R_x, the transmitted bits are received, and the receiver aims to estimate the correct values of the transmitted bits with minimal probability of error. Additionally, the receiver seeks to determine the best line code with the least bandwidth consumption.

II. Introduction:

Starting off by generating random uniformly distributed bits (zeros and ones) the number of the generated bits is controlled by a flag used to set the number of bits in each realization(waveform).

Each bit is repeated a set number of time controlled by a flag used to set the number of samples per bit.

Afterwards each line code is constructed by converting the generated bits to the equivalent line code symbols.

The last step is to generate a random time shift value controlled by time delay flag so that each realization has a random start.

The previous steps are repeated a set number of time controlled by a flag that sets the number of realizations in the ensemble.

The generated ensemble is used in the calculation of the statistical mean and the statistical autocorrelation function, and any realization of the ensemble can be used to calculate the time mean and autocorrelation function.

Each line code is then checked to see whether it is stationary and ergodic or not, as an ergodic process is easier to deal with as it requires the generation of one realization only instead of a whole ensemble to determine the process statistics.

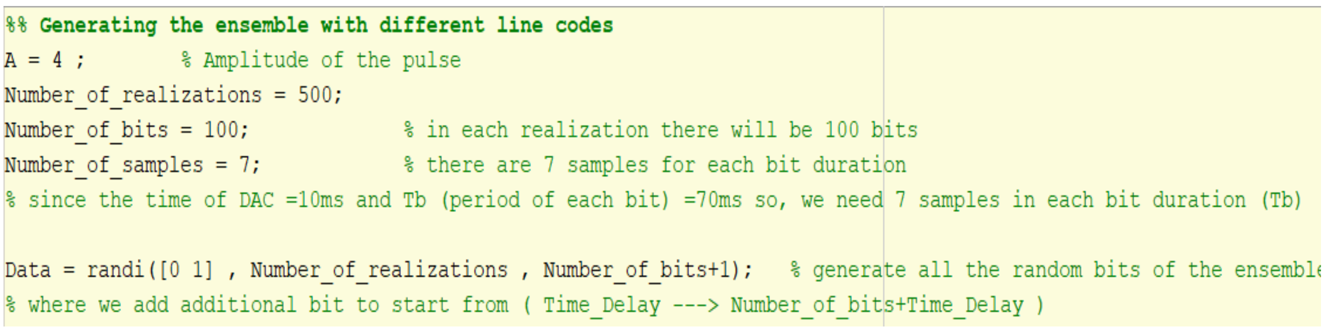
III. Control Flags:

Flag Description	Flag Name in MATLAB Code	Flag Value
Controls The Amplitude of the pulses in line code	A	**4**
Controls the Number of Bits in each realization (waveform)	Number_of_bits	**100**
Controls the Number of samples for each bit in a realization	Number_of_samples	**7**
Controls the Number of Realizations (waveforms) in the ensemble.	Number_of_realizations	**500**
Controls where each realization randomly starts from	Time_Delay	Random value between **0** and **6**

IV. Generation of Data:

Firstly, we initialize a matrix to store the processed data and after randomizing a matrix of a binary data using randi Function with a size of (Number_Of_Realizations × Number_of_bits+1), where an extra bit is added to account for the initial time delay at the start of each realization. Then we duplicate each element in the matrix to seven samples.

- **Code Snippet**

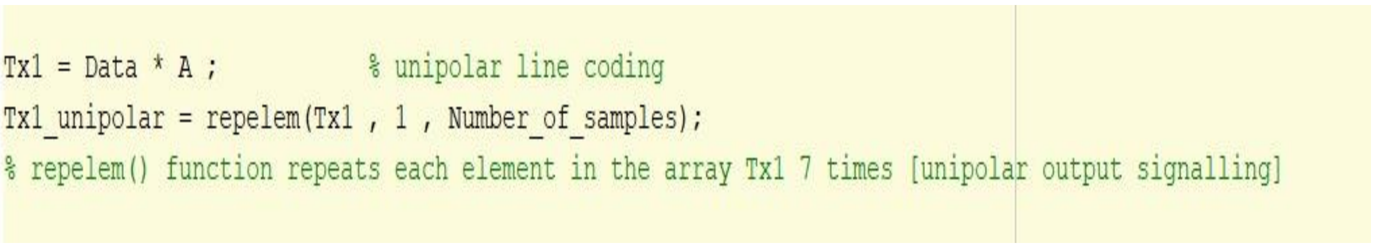


```
%% Generating the ensemble with different line codes
A = 4 ;        % Amplitude of the pulse
Number_of_realizations = 500;
Number_of_bits = 100;            % in each realization there will be 100 bits
Number_of_samples = 7;           % there are 7 samples for each bit duration
% since the time of DAC =10ms and Tb (period of each bit) =70ms so, we need 7 samples in each bit duration (Tb)

Data = randi([0 1] , Number_of_realizations , Number_of_bits+1);   % generate all the random bits of the ensembl
% where we add additional bit to start from ( Time_Delay ---> Number_of_bits+Time_Delay )
```

V. Creation of Unipolar ensemble:

- We Simply multiply the data matrix by the Amplitude (A) then we duplicate each element 7 times.

- **Code Snippet:**

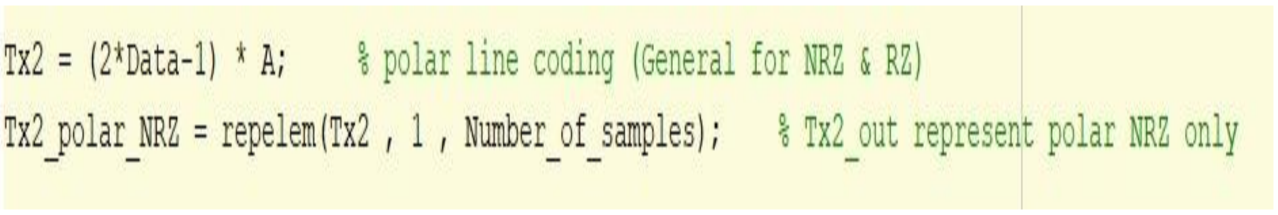


```
Tx1 = Data * A ;            % unipolar line coding
Tx1_unipolar = repelem(Tx1 , 1 , Number_of_samples);
% repelem() function repeats each element in the array Tx1 7 times [unipolar output signalling]
```

VI. Creation of Polar NRZ ensemble:

- Multiply each element in data matrix by 2 then subtract 1 after that we take the result matrix and multiply it by (A) “this step to produce values range [-A, A]”

- **Code Snippet:**

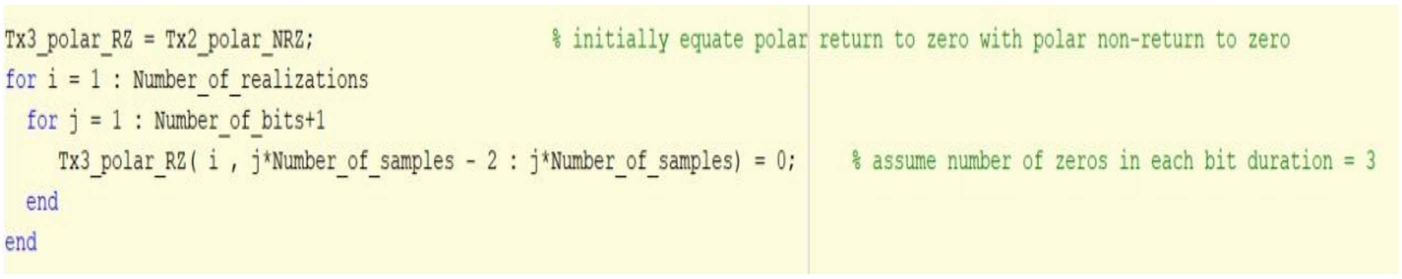


```
Tx2 = (2*Data-1) * A;     % polar line coding (General for NRZ & RZ)
Tx2_polar_NRZ = repelem(Tx2 , 1 , Number_of_samples);    % Tx2_out represent polar NRZ only
```

VIII. Creation of Polar RZ ensemble:

- The same concept as in Polar Non-return to zero (NRZ) but the difference is that during the DAC sampling we take only 4 samples out of 7 to represent the data and the other 3 samples are zeros.

➢ **Code Snippet:**

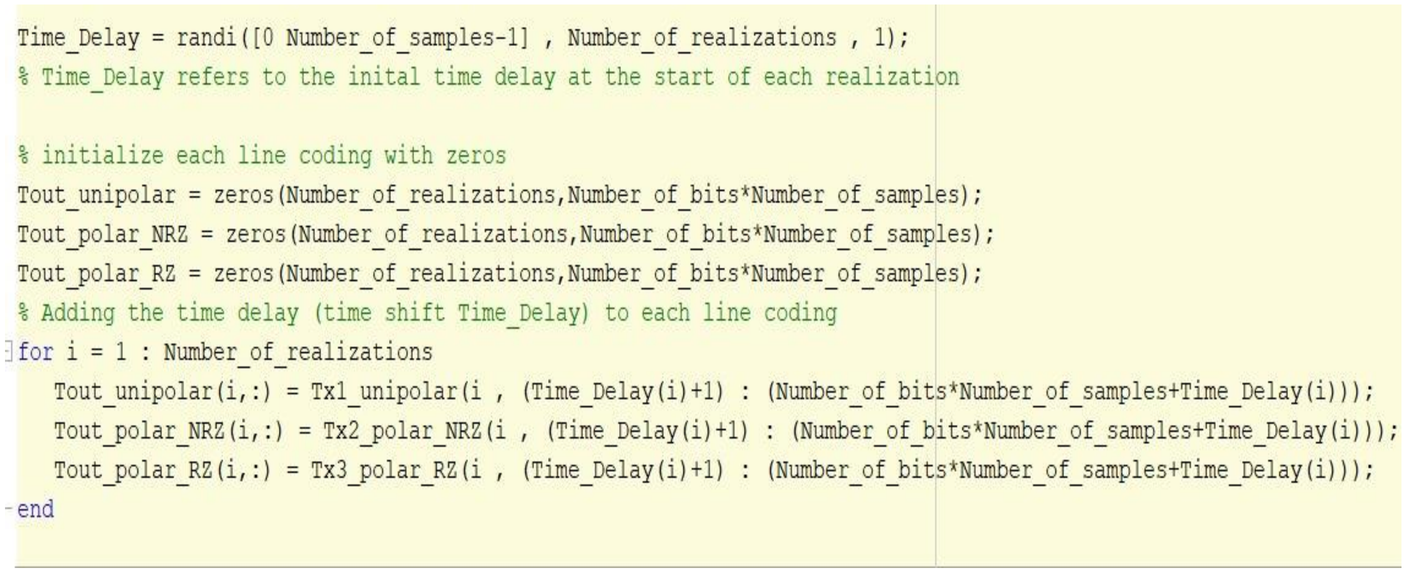


```
Tx3_polar_RZ = Tx2_polar_NRZ;                % initially equate polar return to zero with polar non-return to zero
for i = 1 : Number_of_realizations
  for j = 1 : Number_of_bits+1
    Tx3_polar_RZ( i , j*Number_of_samples - 2 : j*Number_of_samples) = 0;     % assume number of zeros in each bit duration = 3
  end
end
```

VIII. Applying random initial time shifts for each waveform:

- As we said before there is a time delay at the start of each realization due to the transmitter and the receiver not synchronized with each other. Therefore, the random bits will be received at the destination not at the start of the bit exactly but with time shift.
- Also, as mentioned previously an extra bit was added for the time delay so to maintain the original intended ensemble size, we assigned the ensemble data starting from the realization's time shift up to (700 + time shift)."

➢ **Code Snippet:**



```
Time_Delay = randi([0 Number_of_samples-1] , Number_of_realizations , 1);
% Time_Delay refers to the inital time delay at the start of each realization

% initialize each line coding with zeros
Tout_unipolar = zeros(Number_of_realizations,Number_of_bits*Number_of_samples);
Tout_polar_NRZ = zeros(Number_of_realizations,Number_of_bits*Number_of_samples);
Tout_polar_RZ = zeros(Number_of_realizations,Number_of_bits*Number_of_samples);
% Adding the time delay (time shift Time_Delay) to each line coding
for i = 1 : Number_of_realizations
   Tout_unipolar(i,:) = Tx1_unipolar(i , (Time_Delay(i)+1) : (Number_of_bits*Number_of_samples+Time_Delay(i)));
   Tout_polar_NRZ(i,:) = Tx2_polar_NRZ(i , (Time_Delay(i)+1) : (Number_of_bits*Number_of_samples+Time_Delay(i)));
   Tout_polar_RZ(i,:) = Tx3_polar_RZ(i , (Time_Delay(i)+1) : (Number_of_bits*Number_of_samples+Time_Delay(i)));
end
```

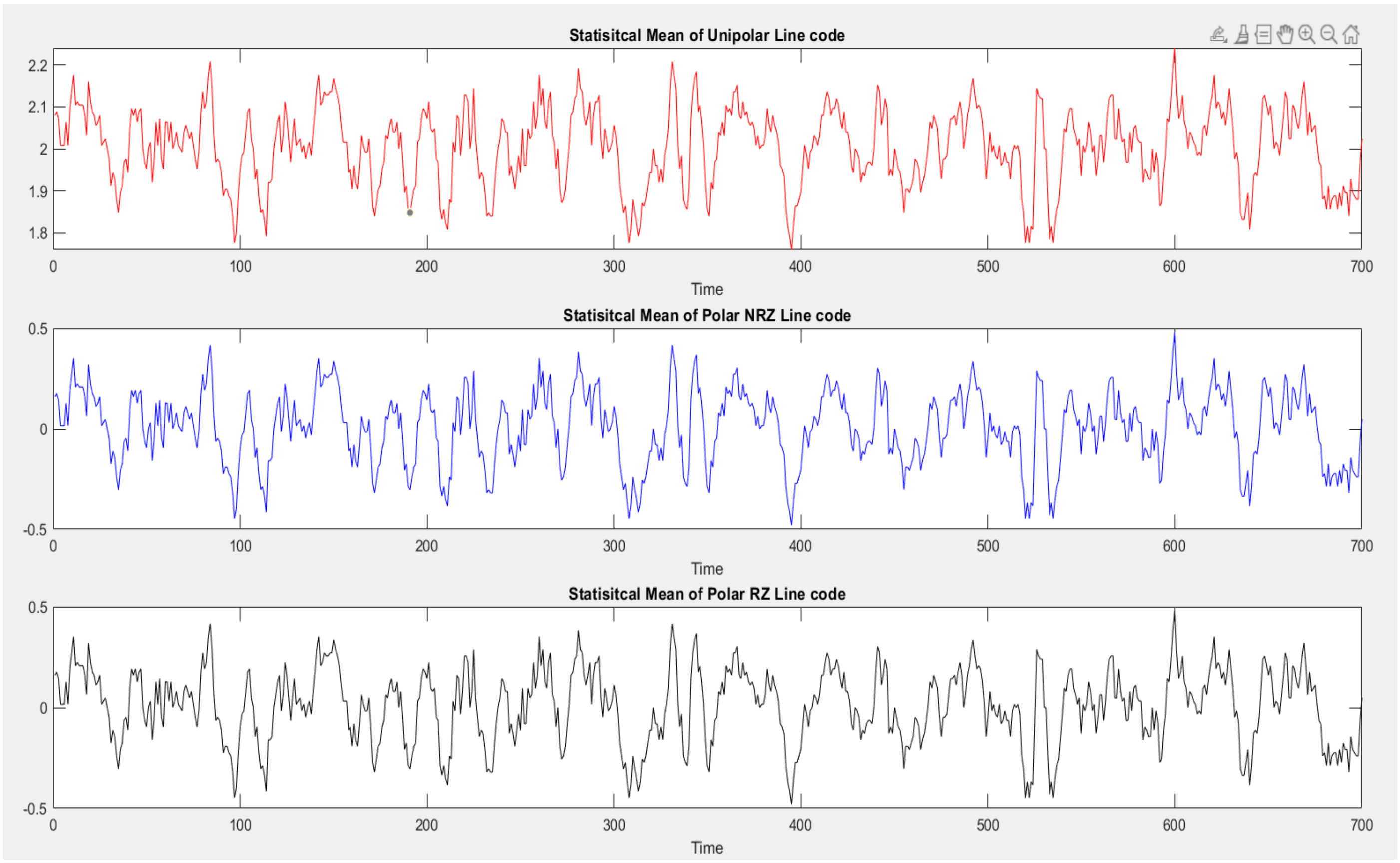
IX. Calculating the Statistical Mean:

➢ Code Snippet:

```
% for unipolar line code
Statistical_mean_unipolar = zeros(1,Number_of_bits*Number_of_samples);
for j = 1 : (Number_of_bits*Number_of_samples)
   Statistical_mean_unipolar(1,j) = Statistical_mean_unipolar(1,j) +  (sum(Tout_unipolar(:,j))/Number_of_realizations);
end
 % for polar RZ line code
Statistical_mean_polar_RZ = zeros(1,Number_of_bits*Number_of_samples);
for j = 1 : (Number_of_bits*Number_of_samples)
   Statistical_mean_polar_RZ(1,j) = Statistical_mean_polar_RZ(1,j) + (sum(Tout_polar_RZ(:,j))/Number_of_realizations);
end
 % for polar NRZ line code
Statistical_mean_polar_NRZ = zeros(1,Number_of_bits*Number_of_samples);
for j = 1 : (Number_of_bits*Number_of_samples)
   Statistical_mean_polar_NRZ(1,j) = Statistical_mean_polar_NRZ(1,j) +  (sum(Tout_polar_NRZ(:,j))/Number_of_realizations);
end
```

- The above codes represent how to calculate the statistical mean for different line codes where the concept to calculate it is to sum all the values in certain column for all realizations (**specific column represent random variable RV**)

X. Plotting the Statistical mean:



➢ Comments:

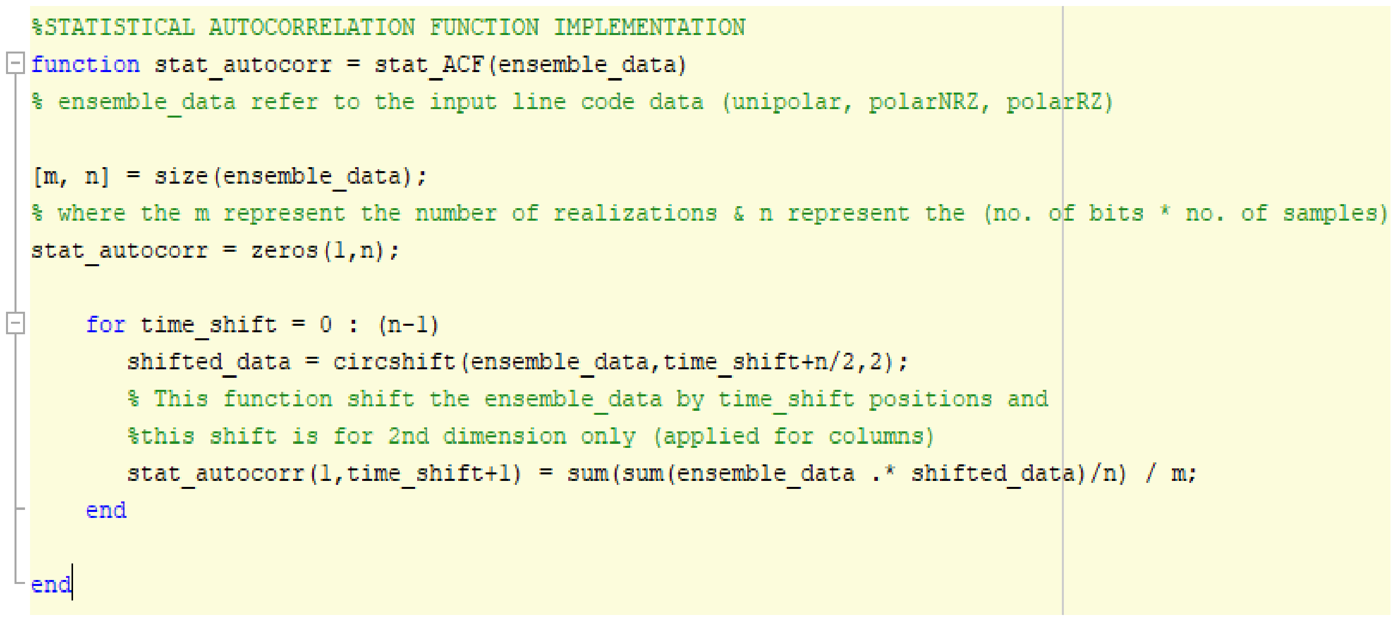
1) As expected for the two polar representations, the expected value fluctuates around zero, aligning with the theoretical expectation (μ_x= -A/2 +A/2=zero) and for the unipolar representation, the expected value oscillates around 2, consistent with the theoretical prediction (m_x= 0×1/2 +A×1/2=A/2).

2) We notice from the above figures that the statistical mean is not purely constant and there are some variations (ripples) and that is because we took a finite number of realizations = 500 only and a small number of bits in each realization the number of realizations increases the mean tends to be constant across time.

XI. Calculating the Statistical Autocorrelation:

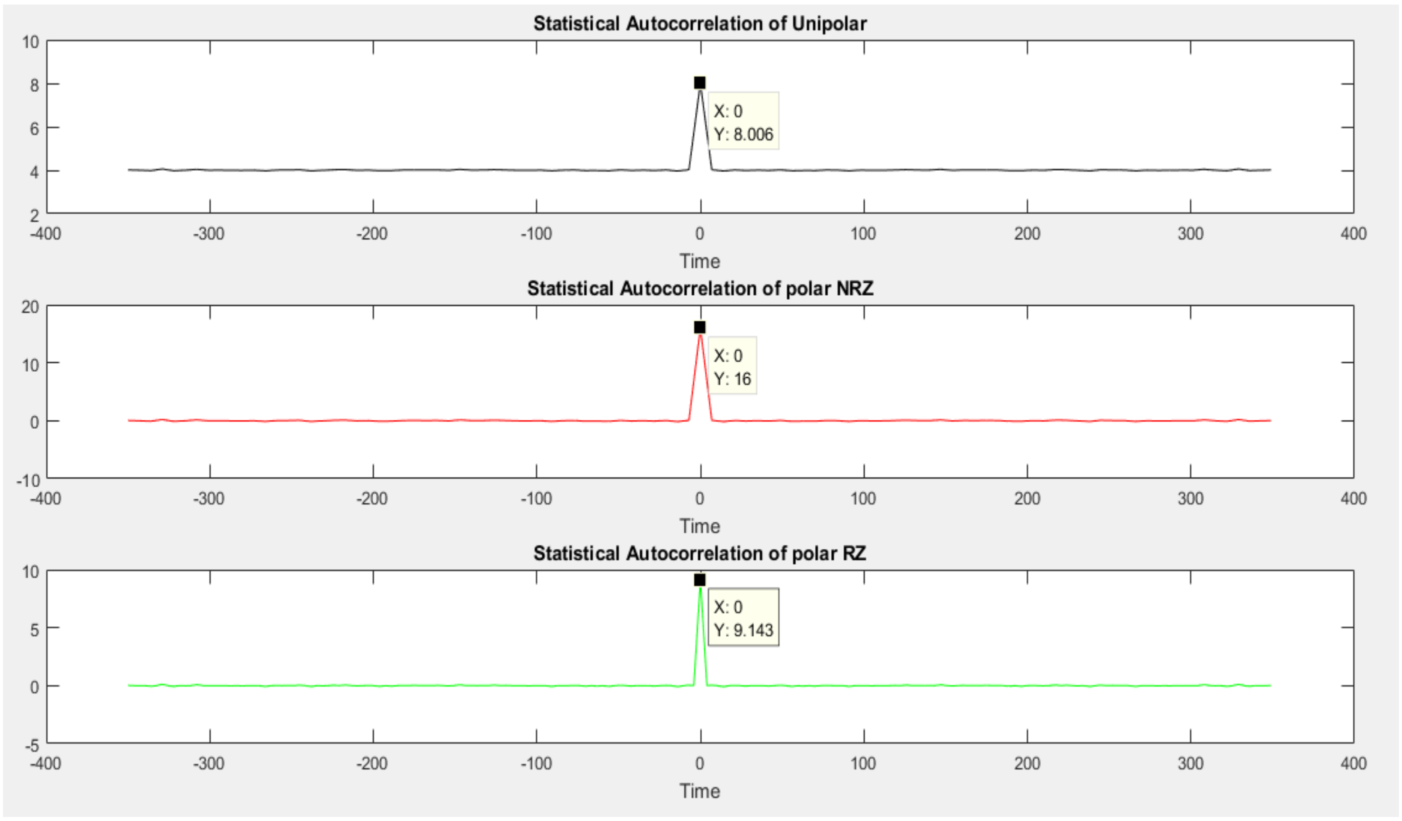
- We calculate the statistical autocorrelation by multiplying certain time instant (certain column) "t1" by all the possible combinations of the second time instant "t2" then shift the time instant "t1" and apply the same steps until the first time instant "t1" take all the possible combinations.

➢ Code Snippet:



```
%STATISTICAL AUTOCORRELATION FUNCTION IMPLEMENTATION
function stat_autocorr = stat_ACF(ensemble_data)
% ensemble_data refer to the input line code data (unipolar, polarNRZ, polarRZ)

[m, n] = size(ensemble_data);
% where the m represent the number of realizations & n represent the (no. of bits * no. of samples)
stat_autocorr = zeros(1,n);

    for time_shift = 0 : (n-1)
       shifted_data = circshift(ensemble_data,time_shift+n/2,2);
       % This function shift the ensemble_data by time_shift positions and
       %this shift is for 2nd dimension only (applied for columns)
       stat_autocorr(1,time_shift+1) = sum(sum(ensemble_data .* shifted_data)/n) / m;
    end

end
```

XII. Plotting the Statistical Autocorrelation:



➢ Comments:

- The above figure shows the statistical autocorrelation of different line codes vs the time shift (τ) where the simulated values is very close to the theoretical values.

- Initially, in the case of unipolar line coding, as anticipated $R_x(\tau)$ showcases a DC component equal to the square of the expected value ($A^2/4$) which equals 4.

- From some calculations we conclude that at time shift (τ) =0 the theoretical values for the different line codes are as follow: (where A represents the amplitude of the pulse)

$$\begin{tikzcd}
 & \text{ELC 3070} \arrow[dl] \arrow[d] \arrow[dr] & \\
\boxed{\text{Unipolar line coding}} \arrow[d, Rightarrow] & \boxed{\text{Polar NRZ line coding}} \arrow[d, Rightarrow] & \boxed{\text{Polar RZ line coding}} \arrow[d, Rightarrow] \\
\boxed{R_x(0) = \frac{A^2}{2}} & \boxed{R_x(0) = A^2} & \boxed{R_x(0) = \frac{4}{7} * A^2}
\end{tikzcd}$$

- Therefore, from the above equations we can obtain the theoretical values of statistical Autocorrelation function at τ = 0

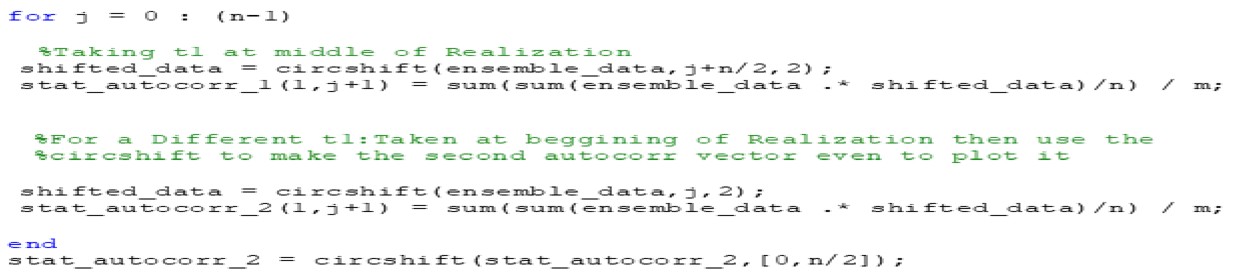
$$\begin{tikzcd}
 & 8 \quad \textbf{for unipolar} \\
R_x(0) \arrow[ur] \arrow[r] \arrow[dr] & 16 \quad \textbf{for polar NRZ} \\
 & \approx 9.14 \quad \textbf{for polar RZ}
\end{tikzcd}$$

- The simulated values above are not exactly equal to the theoretical values as we take finite number of bits in each realization whenever the number of bits increase the simulated values approach to the theoretical values.

XIII. Is the process stationary?

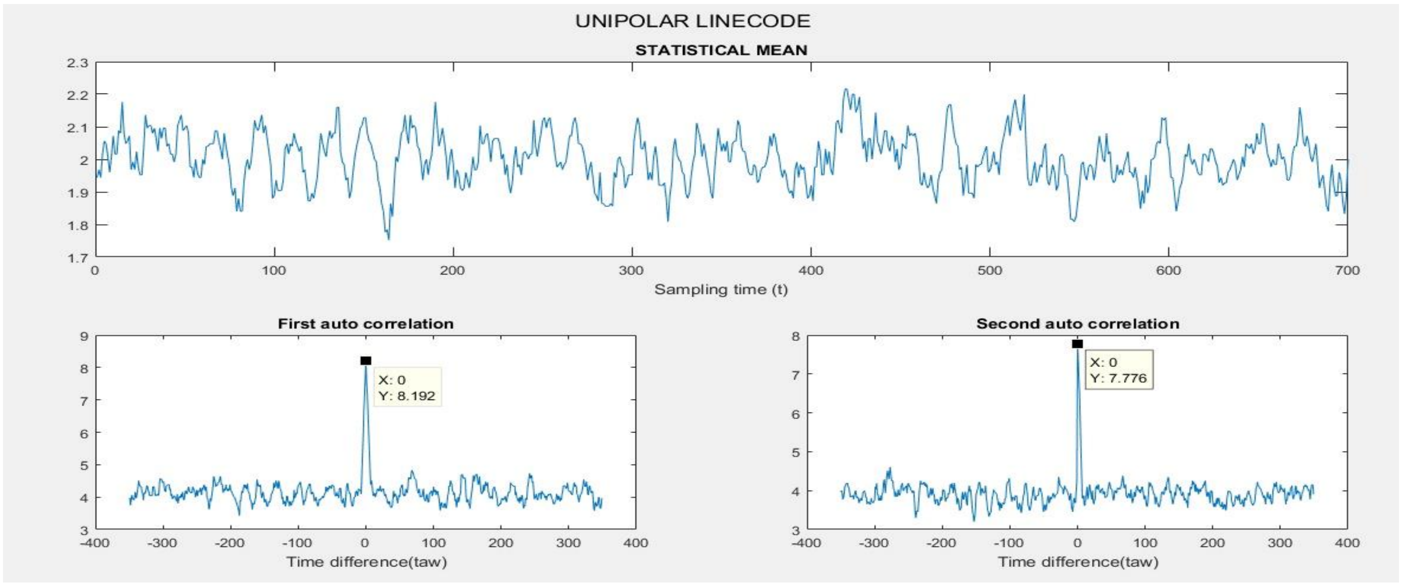
- **Above, we computed the statistical mean and autocorrelation for each line code, where Rx(τ) = (E(x(t1) x(t2) by evaluating it at every possible combination of time instants t1 and t2 for However, in this section, to validate the Wide Sense Stationary (WSS) property of the line code, we re-implemented the autocorrelation function to illustrate its dependence on (τ). Then, we recalculated the autocorrelation for different t¬1 values (e.g., t1= 1 or t1=350) and compared the results to assess stationarity.**

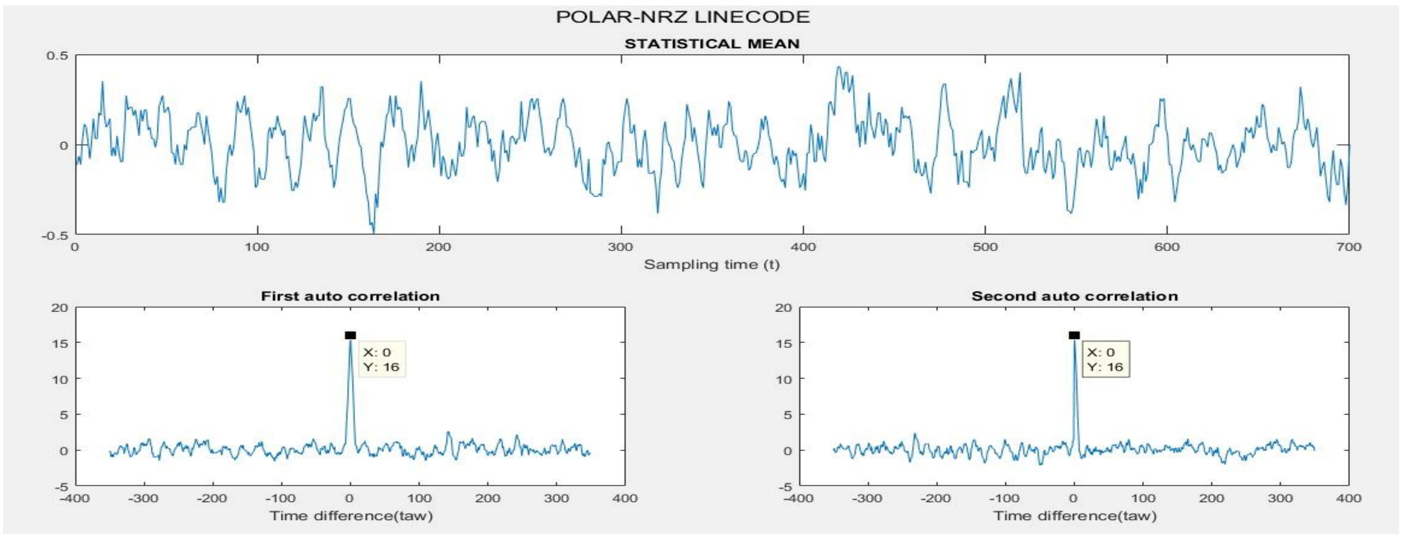
- **Code Snippet:**

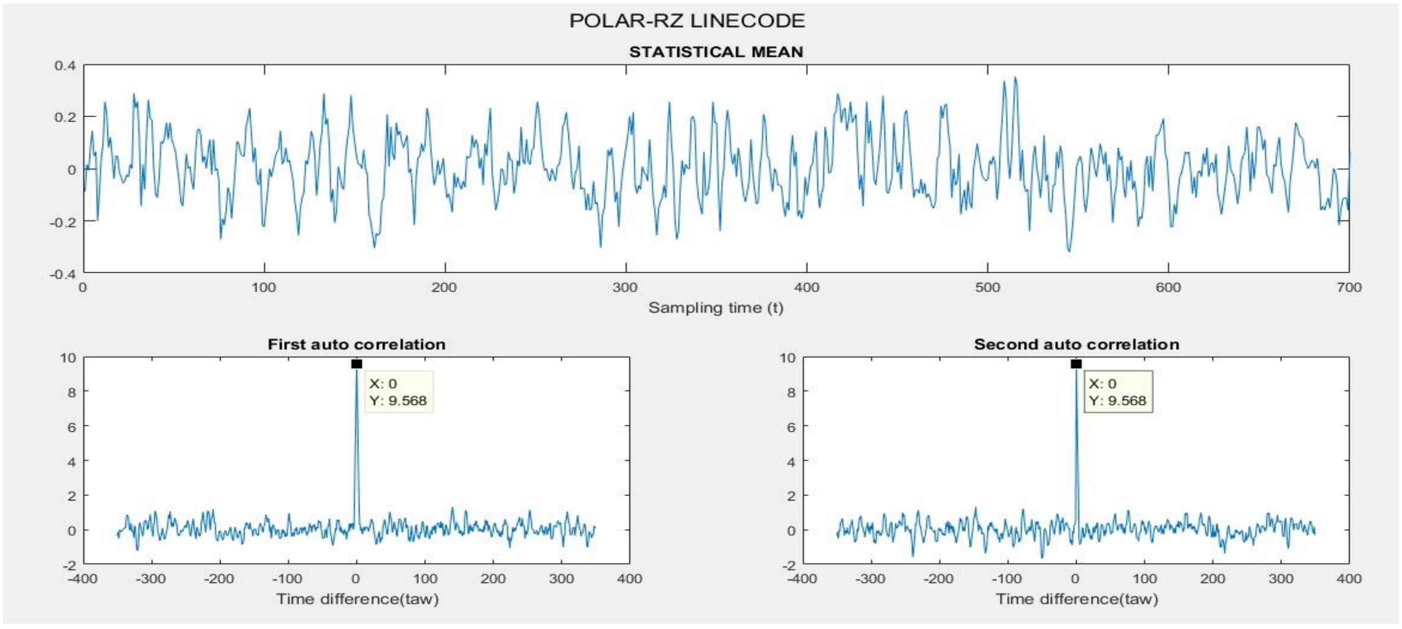


```
for j = 0 : (n-1)

  %Taking t1 at middle of Realization
 shifted_data = circshift(ensemble_data,j+n/2,2);
 stat_autocorr_1(1,j+1) = sum(sum(ensemble_data .* shifted_data)/n) / m;

  %For a Different t1:Taken at beggining of Realization then use the
  %circshift to make the second autocorr vector even to plot it

 shifted_data = circshift(ensemble_data,j,2);
 stat_autocorr_2(1,j+1) = sum(sum(ensemble_data .* shifted_data)/n) / m;

end
stat_autocorr_2 = circshift(stat_autocorr_2,[0,n/2]);
```

➢ Plotted results:







➢ Comments: Applies for all line codes

- **Regarding the statistical mean, it has a relatively small variations across time as the taken sample size of the Realizations is not a large number (=500), so the statistical mean could be considered constant across time if a larger sample was taken.**

- **For the Autocorrelation, the plots of the two vectors are similar with slight differences as shown in the above figure. when different t1 was assigned so the Rx(τ) does not depend on the absolute time value it depends only on the time difference value.**

➢ **Therefore, these three-line codes are considered a WSS processes.**

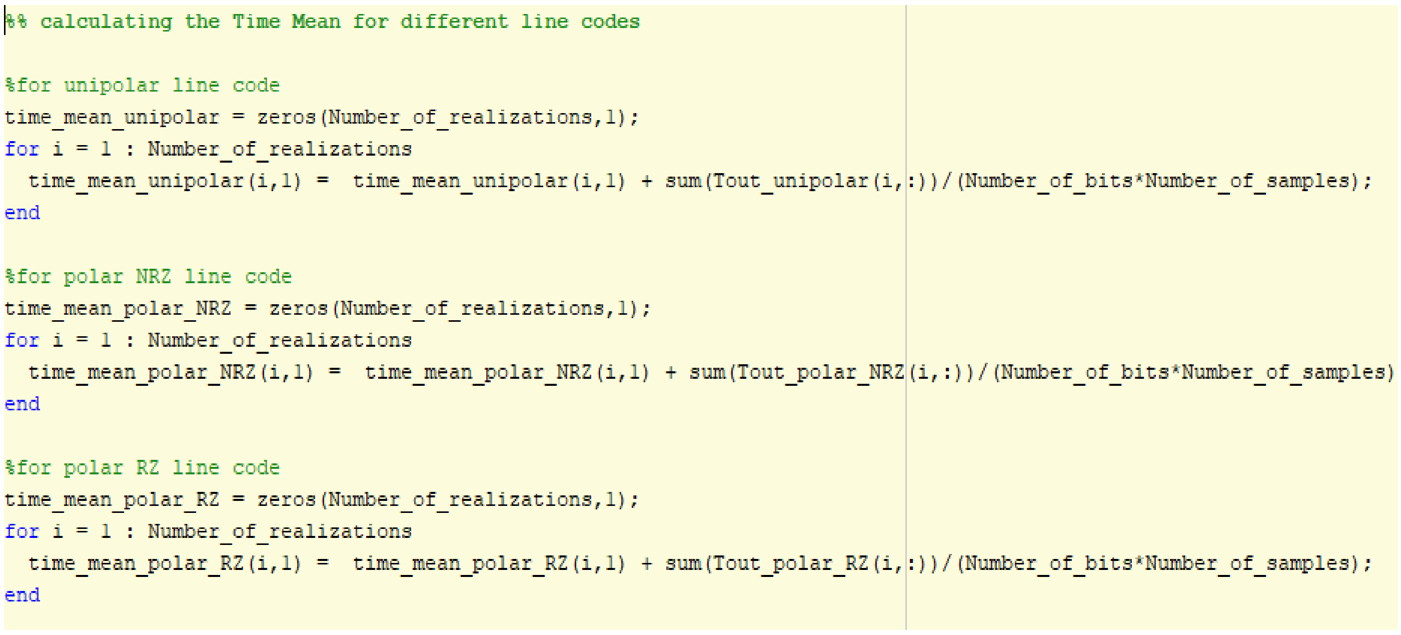
XIV. Computing the time mean & time autocorrelation for one waveform:

- Simply the time mean is calculated by taking the average of each realization by adding each sample and dividing on the total number of samples in each realization.

- For the time autocorrelation calculation occurs on only one realization, it is performed by a custom function that takes the whole Tx-ensemble extracts the specified realization and shifts it based on a time difference value. Then element-wise multiplication is performed between the shifted realization and the original one. Finally, the sum of all product elements is computed and divided by their total count.

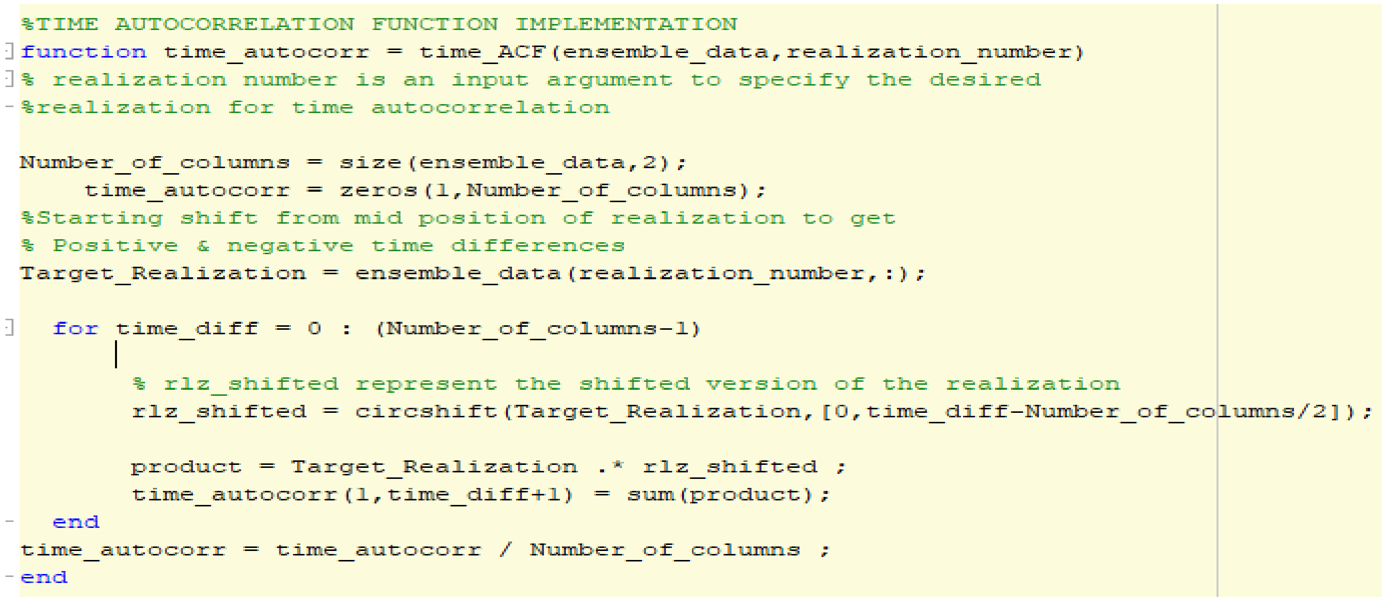
➢ Code Snippet:

a) Time mean:



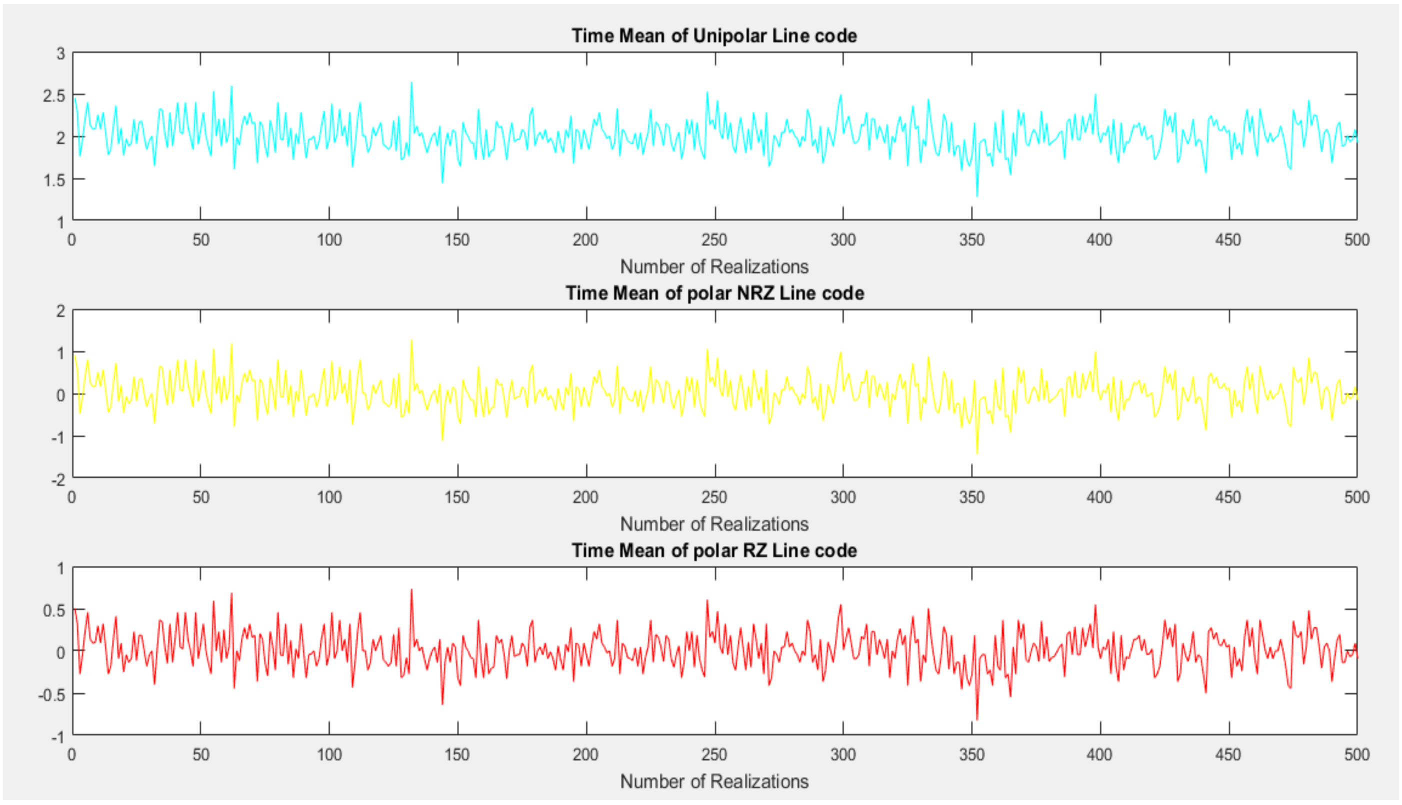
```
%% calculating the Time Mean for different line codes

%for unipolar line code
time_mean_unipolar = zeros(Number_of_realizations,1);
for i = 1 : Number_of_realizations
  time_mean_unipolar(i,1) =  time_mean_unipolar(i,1) + sum(Tout_unipolar(i,:))/(Number_of_bits*Number_of_samples);
end

%for polar NRZ line code
time_mean_polar_NRZ = zeros(Number_of_realizations,1);
for i = 1 : Number_of_realizations
  time_mean_polar_NRZ(i,1) =  time_mean_polar_NRZ(i,1) + sum(Tout_polar_NRZ(i,:))/(Number_of_bits*Number_of_samples)
end

%for polar RZ line code
time_mean_polar_RZ = zeros(Number_of_realizations,1);
for i = 1 : Number_of_realizations
  time_mean_polar_RZ(i,1) =  time_mean_polar_RZ(i,1) + sum(Tout_polar_RZ(i,:))/(Number_of_bits*Number_of_samples);
end
```

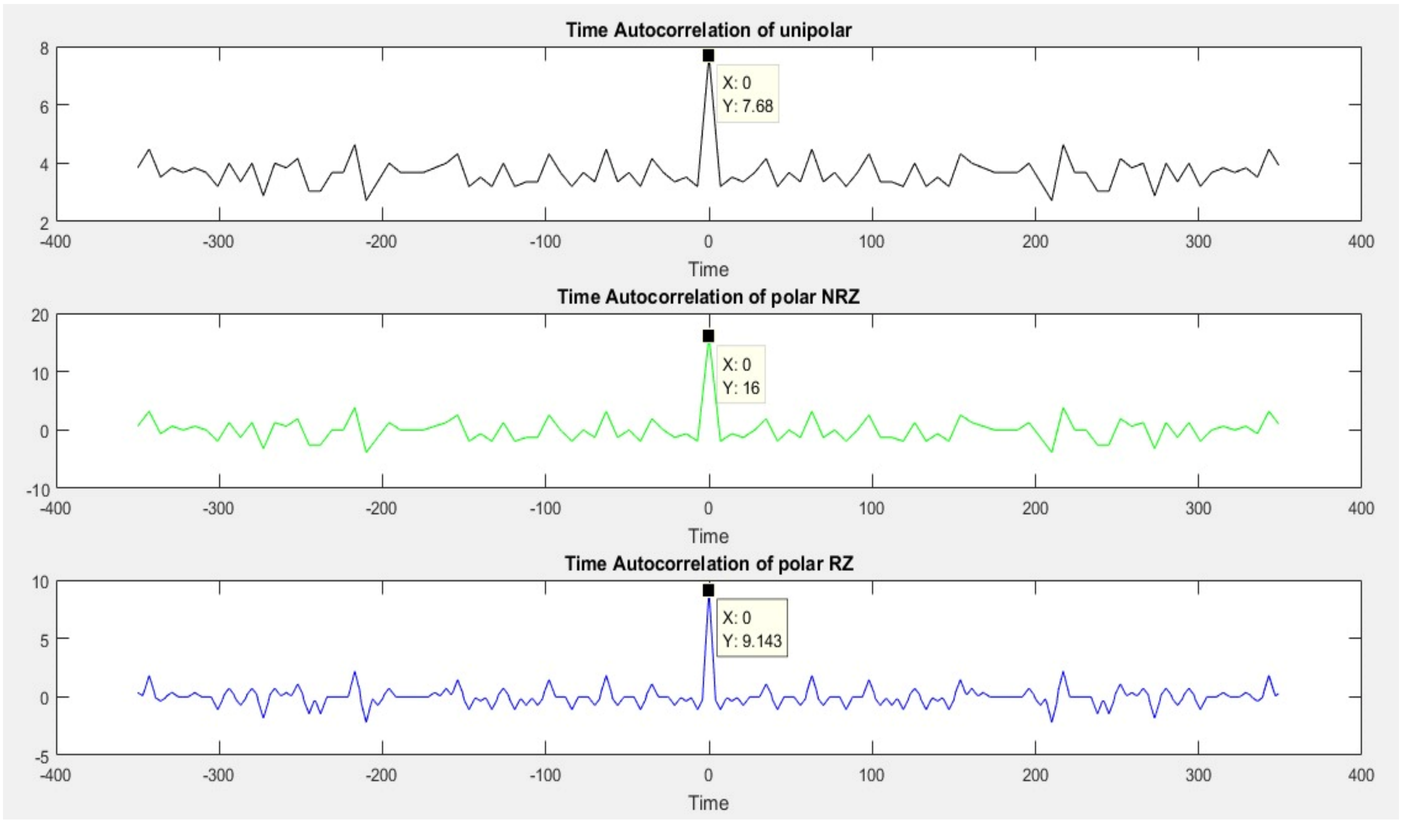
b) Time autocorrelation:



```
%TIME AUTOCORRELATION FUNCTION IMPLEMENTATION
function time_autocorr = time_ACF(ensemble_data,realization_number)
% realization number is an input argument to specify the desired
%realization for time autocorrelation

Number_of_columns = size(ensemble_data,2);
    time_autocorr = zeros(1,Number_of_columns);
%Starting shift from mid position of realization to get
% Positive & negative time differences
Target_Realization = ensemble_data(realization_number,:);

  for time_diff = 0 : (Number_of_columns-1)

      % rlz_shifted represent the shifted version of the realization
      rlz_shifted = circshift(Target_Realization,[0,time_diff-Number_of_columns/2]);

      product = Target_Realization .* rlz_shifted ;
      time_autocorr(1,time_diff+1) = sum(product);
  end
time_autocorr = time_autocorr / Number_of_columns ;
end
```

➢ Plotted results:





➢ Comments:

- **For the time mean:**

  a) As said in the statistical mean calculations for the variations across time it also applies here for the time mean as the relatively small finite number of samples (700) caused some variations around the theoretical value.
  b) In the case of unipolar-NRZ line coding, as shown in the above figure <x(t)> varies around [$\frac{1}{2T}(0 \times T + A \times T) = A/2$] which equals 2.
  c) In the case of polar-NRZ line coding, <x(t)> should vary around [$\frac{1}{2T}(-A \times T + A \times T) = zero$] , which occurs in the above figures as well
  d) In the case of polar-RZ line coding, <x(t)> should be the same as polar-NRZ which occurs also in the plot above.

- **For the time autocorrelation**, expecting ergodicity across the three different line codes, so the theoretical values should closely align with the statistical autocorrelation calculations mentioned above as:

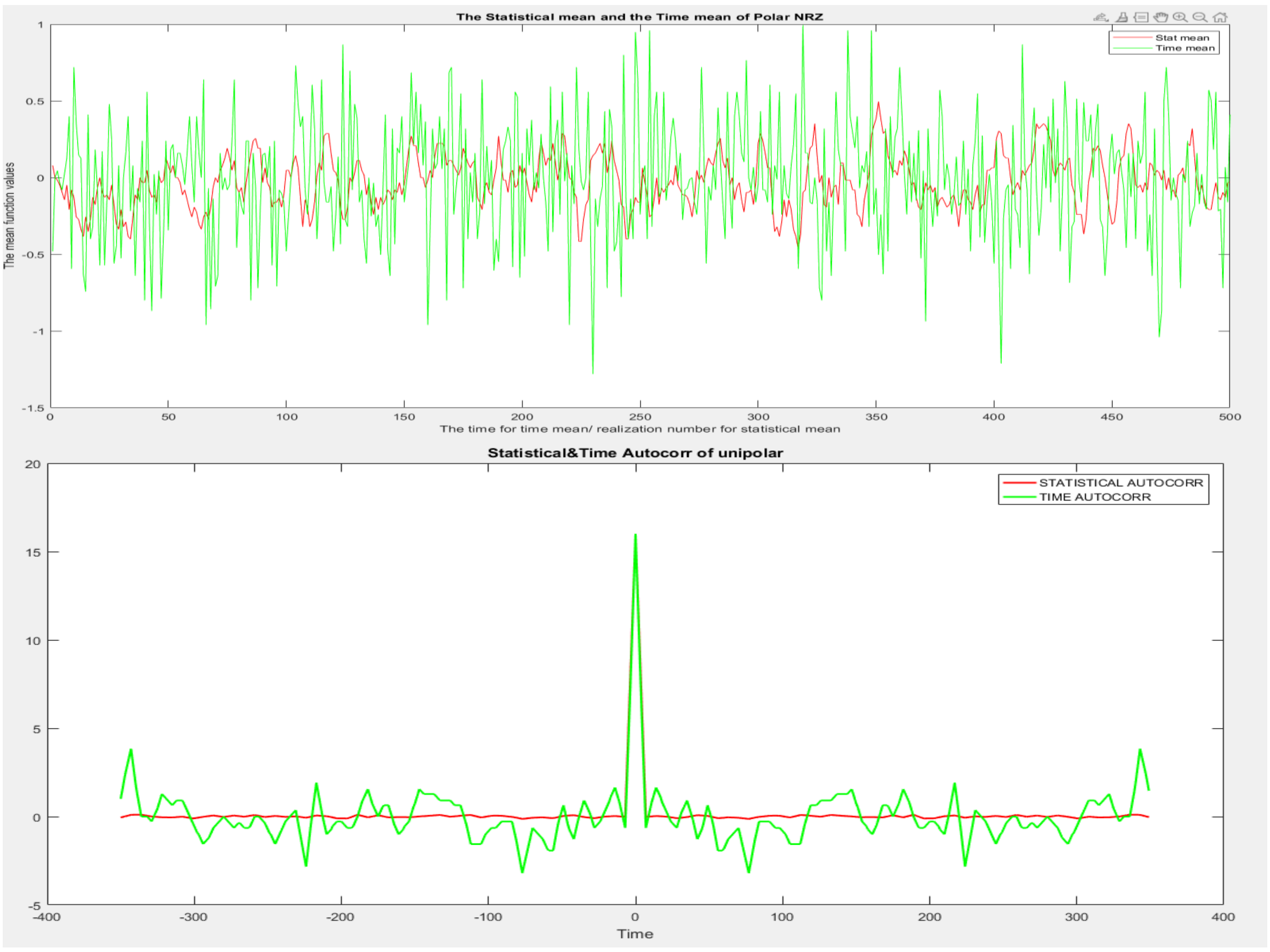
$$\begin{tikzcd} & 8 \quad \text{for unipolar} \\ \boldsymbol{R_x(0)} \arrow[ur] \arrow[r] \arrow[dr] & 16 \quad \text{for polar NRZ} \\ & \approx 9.14 \quad \text{for polar RZ} \end{tikzcd}$$

- From the Simulation , the simulated values above do not show a large difference from the expected values and the small change in the peaks amplitude is related to randomness in the signal and the small finite number of sample bits as $<x(t_1).x(t_1)>$ measures how much a signal resembles itself at a specific moment in time t_1, and on the other hand, the statistical autocorrelation R_x (0) considers how similar the signal is to itself on average, across all time instances. So, there could be a slight change in amplitudes.

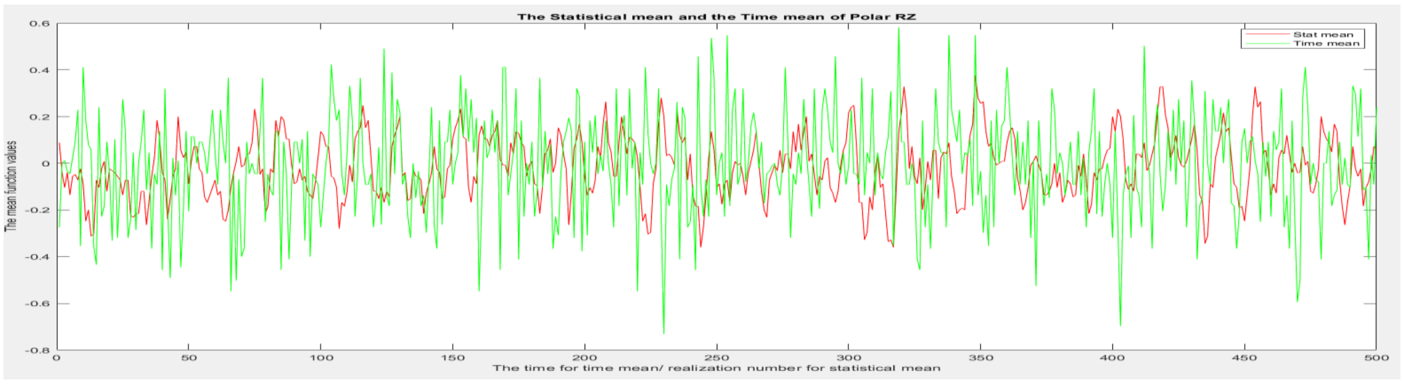
XV. Is the Random process Ergodic?

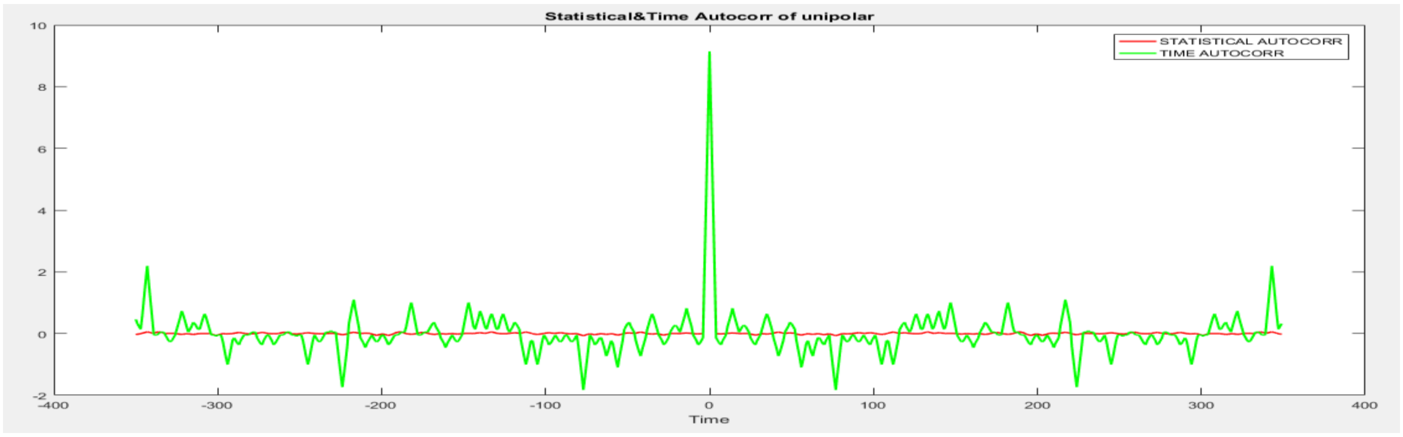
- A process is ergodic when the statistical mean is equal to time mean and the statistical autocorrelation function is equal to time ACF. So, to compare them we plotted them together.

1) <u>For Polar NRZ:</u>

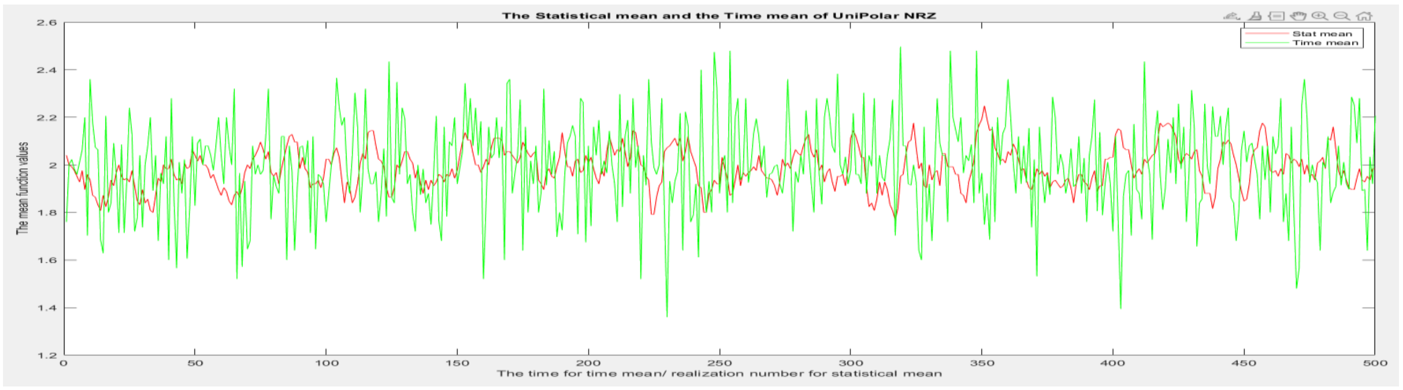


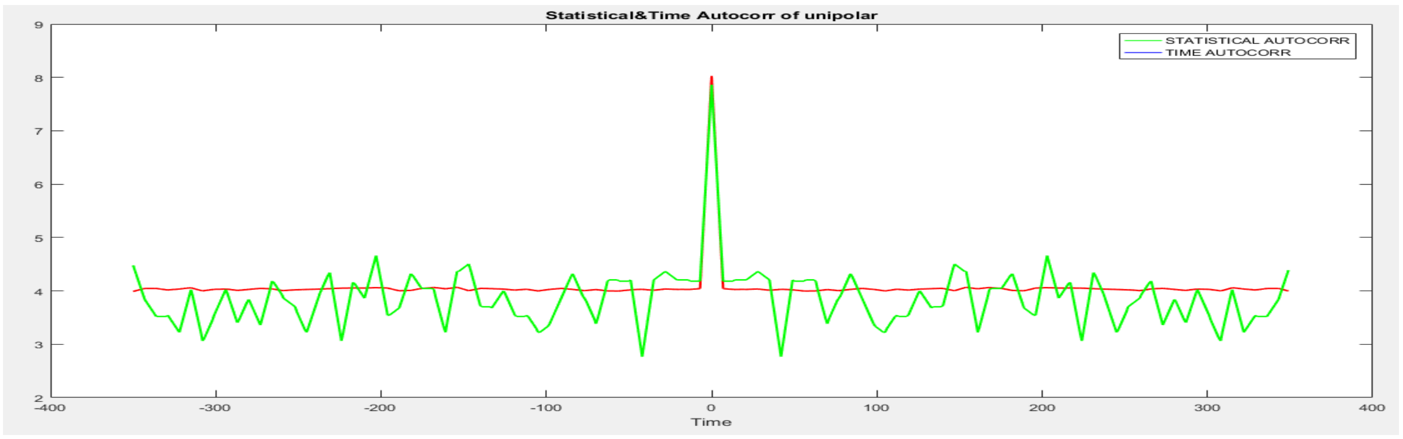
2) <u>For Polar RZ:</u>





3) <u>Unipolar non return to zero:</u>





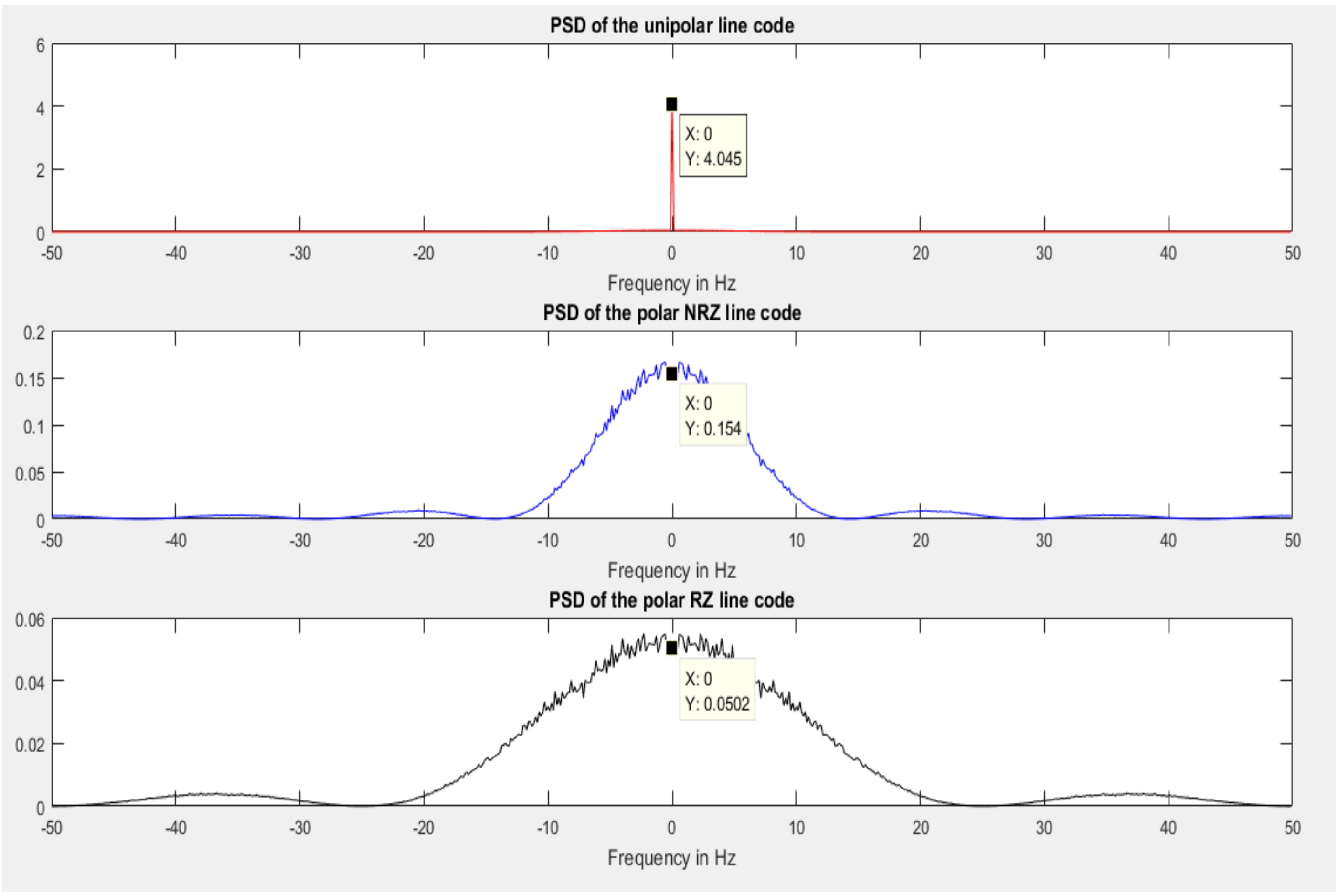
➢ Comments:

a) From the previous plots we can notice the similarity between statistical/time mean and ACF of the line codes as they have ripple around the same theoretical value having the same trends.

b) These ripples are caused by the fact that the number of samples nor the realizations are infinite and since the number of realizations used is more than the number of bits thus the resolution of statistical mean/ACF is better than that of time mean/ACF.

c) With that we can conclude that statistical mean/ACF and time mean/ACF are approximately equal thus the line codes are ergodic processes.

XVI. Plotting the PSD of the ensemble:



XVII. What is the Bandwidth of the transmitted signal?

- The above figure shows the power spectral density of different line codes vs the frequency range in Hertz where the simulated values are very close to the theoretical values.
- From some calculations we conclude that at frequency =0 the theoretical values for the different line codes are as follow

$$\begin{tikzcd}
& {} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Unipolar line coding} \arrow[d, Rightarrow] & \text{Polar NRZ line coding} \arrow[d, Rightarrow] & \text{Polar RZ line coding} \arrow[d, Rightarrow] \\
\boxed{S_x(0)=\frac{A^2}{4}(1+Ts)} & \boxed{S_x(0)=A^2*Ts} & \boxed{S_x(0)=\left(\frac{4}{7}\right)^2*A^2*Ts}
\end{tikzcd}$$

- **<u>Note:</u>** Ts refer to the sampling period of the DAC which is 10 ms.

- Therefore, from the above equations we can obtain the theoretical values of PSD function at f = 0

$$S_x(0)\begin{cases}\rightarrow \mathbf{4.04} & \textbf{for unipolar}\\ \rightarrow \mathbf{0.16} & \textbf{for polar NRZ}\\ \rightarrow \mathbf{0.04} & \textbf{for polar RZ}\end{cases}$$

- The simulated values above are not exactly equal to the theoretical values as we take finite number of bits in each realization whenever the number of bits increase the simulated values approach to the theoretical values.

a) Polar non return to zero:

The Theoretical power spectral density is $SINC^2$ function and the simulated PSD have a similar shape, also the theoretical value of the bandwidth is 1/Tb Where Tb is the time of the pulse which is $\frac{1}{70*10^{-3}} = \mathbf{14.285}\boldsymbol{Hz}$ while the simulated Bandwidth value is **14.2857 Hz** it's obvious they are very similar indicating that the simulation was successful.

b) Polar return to zero:

It’s the same as Polar no return to zero but bit duration is cut and returns to zero for 3 out of the 7 samples so the PSD is expected to have the same shape as polar NRZ, and its theoretical bandwidth will $\frac{1}{70*10^{-3}*\frac{4}{7}} = 25Hz$ while the simulated Bandwidth value is

25.2857 Hz

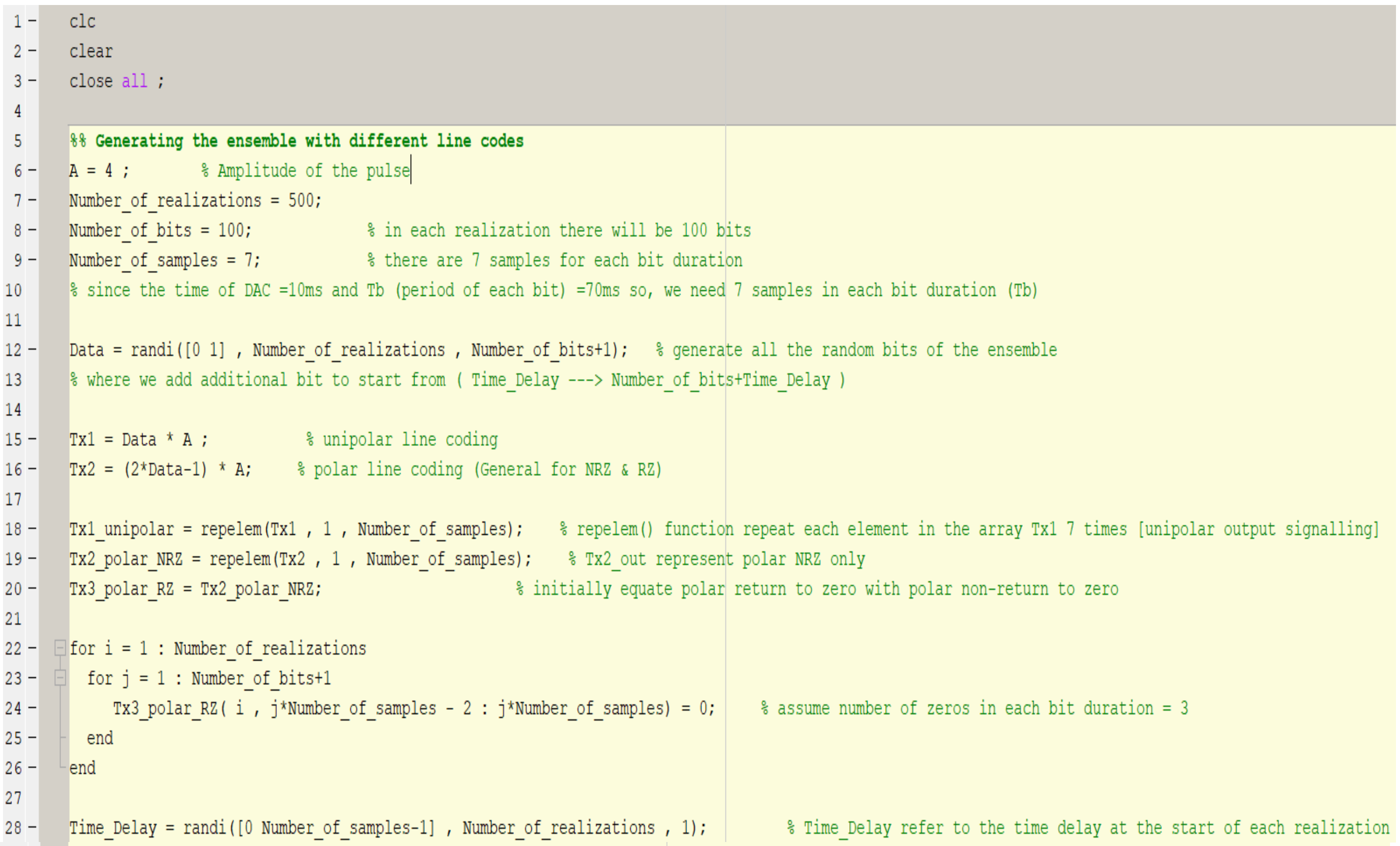
c) Unipolar non return to zero:

The theoretical PSD will be a $SINC^2$ function with a DC impulse whose theoretical value is $\frac{A^2}{4}$. The Dc component exists as the line code mean is not zero. The theoretical Bandwidth is $\frac{1}{70*10^{-3}} = 14.285Hz$ while the simulated Bandwidth value is **14.2857 Hz.**

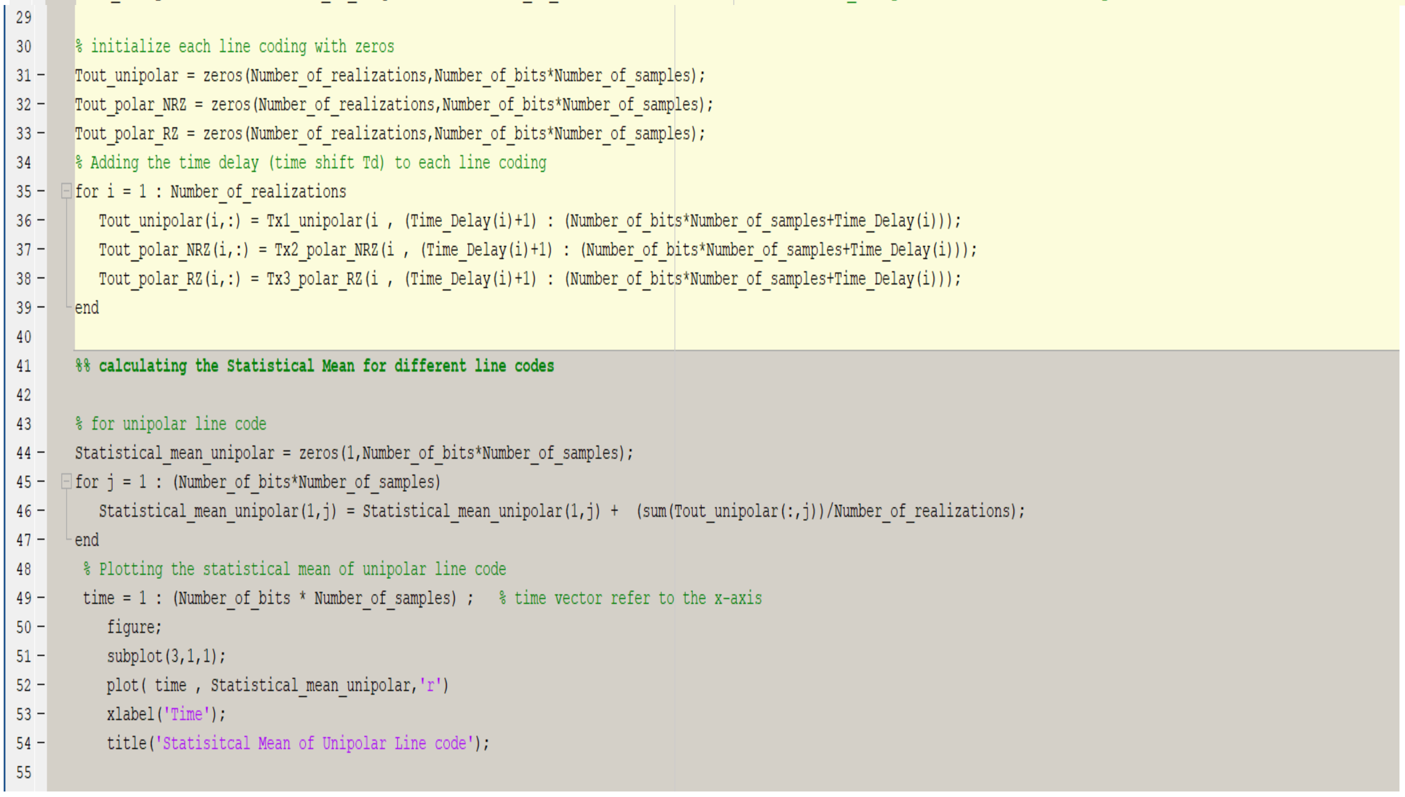
- Conclusion:

Polar NRZ Line code is better than Unipolar NRZ as they use the same bandwidth, but The Polar NRZ doesn’t have a DC component like the Unipolar. Polar RZ line code uses 1.5 more bandwidth than the other line codes due to extra bits needed for transition when returning to zero but in return it has better synchronization ability.

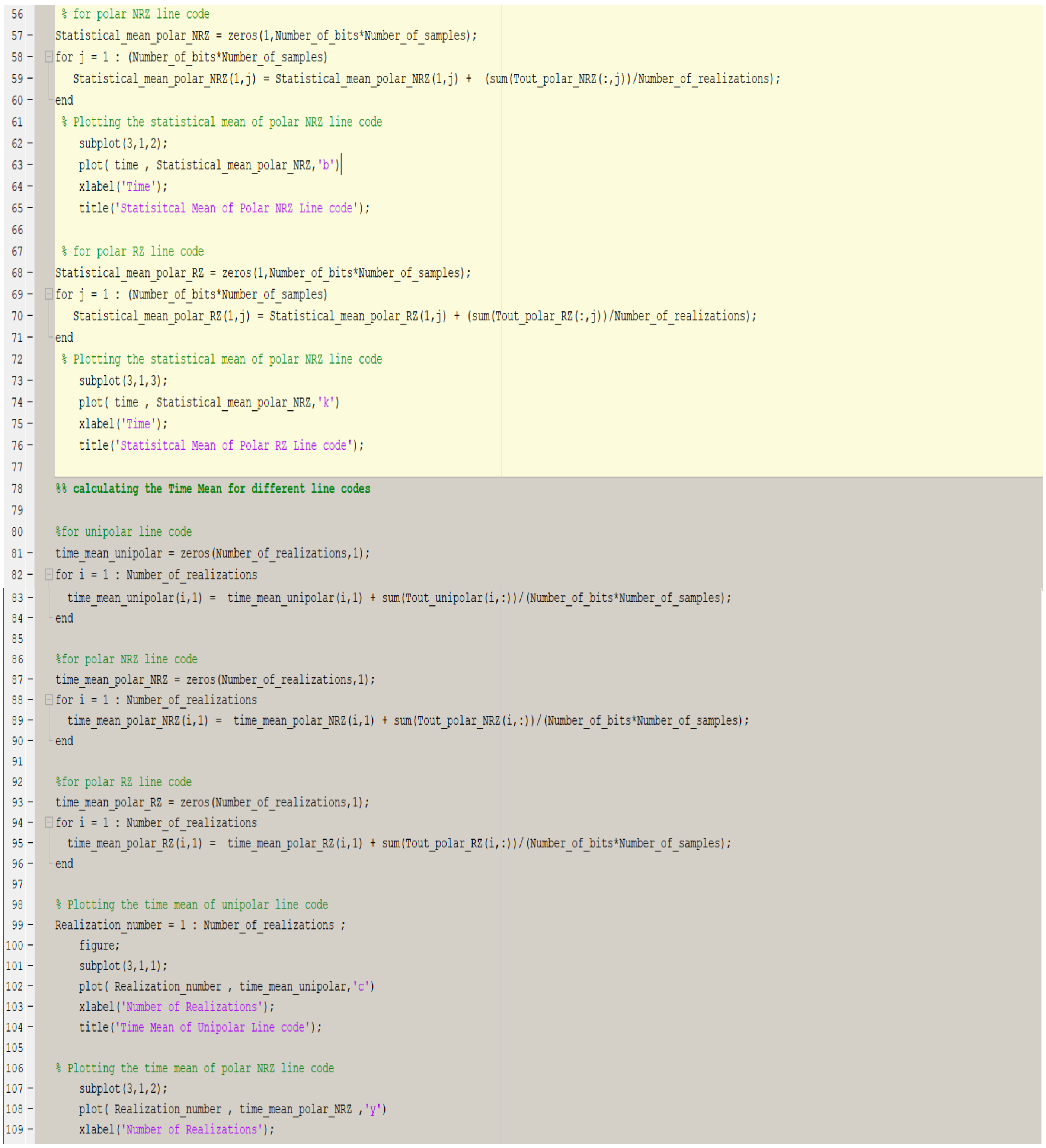
XVIII. Full MATLAB code



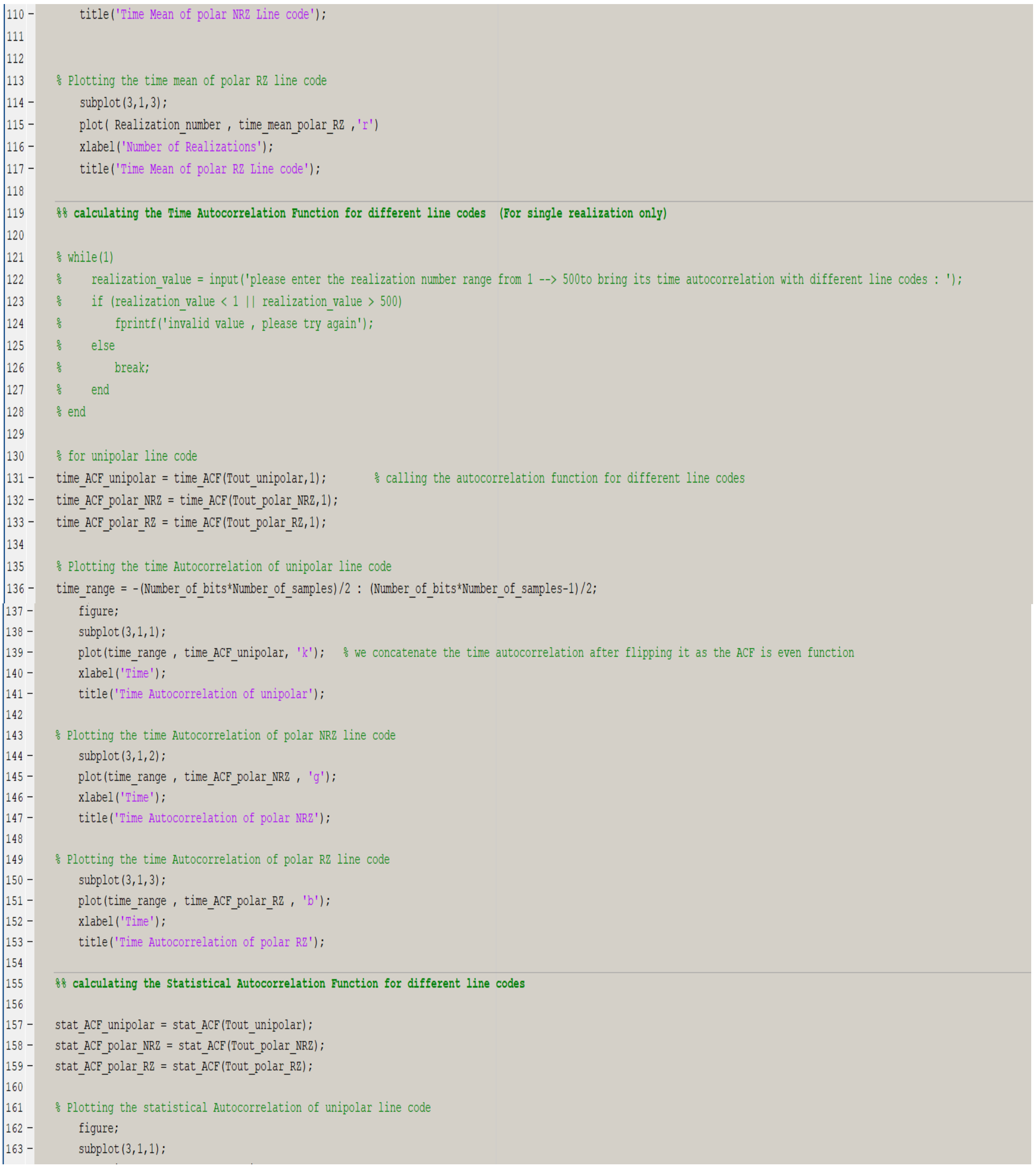
```matlab
clc
clear
close all ;

%% Generating the ensemble with different line codes
A = 4 ;        % Amplitude of the pulse
Number_of_realizations = 500;
Number_of_bits = 100;             % in each realization there will be 100 bits
Number_of_samples = 7;            % there are 7 samples for each bit duration
% since the time of DAC =10ms and Tb (period of each bit) =70ms so, we need 7 samples in each bit duration (Tb)

Data = randi([0 1] , Number_of_realizations , Number_of_bits+1);   % generate all the random bits of the ensemble
% where we add additional bit to start from ( Time_Delay ---> Number_of_bits+Time_Delay )

Tx1 = Data * A ;           % unipolar line coding
Tx2 = (2*Data-1) * A;     % polar line coding (General for NRZ & RZ)

Tx1_unipolar = repelem(Tx1 , 1 , Number_of_samples);    % repelem() function repeat each element in the array Tx1 7 times [unipolar output signalling]
Tx2_polar_NRZ = repelem(Tx2 , 1 , Number_of_samples);    % Tx2_out represent polar NRZ only
Tx3_polar_RZ = Tx2_polar_NRZ;                       % initially equate polar return to zero with polar non-return to zero

for i = 1 : Number_of_realizations
  for j = 1 : Number_of_bits+1
     Tx3_polar_RZ( i , j*Number_of_samples - 2 : j*Number_of_samples) = 0;     % assume number of zeros in each bit duration = 3
  end
end

Time_Delay = randi([0 Number_of_samples-1] , Number_of_realizations , 1);        % Time_Delay refer to the time delay at the start of each realization
```



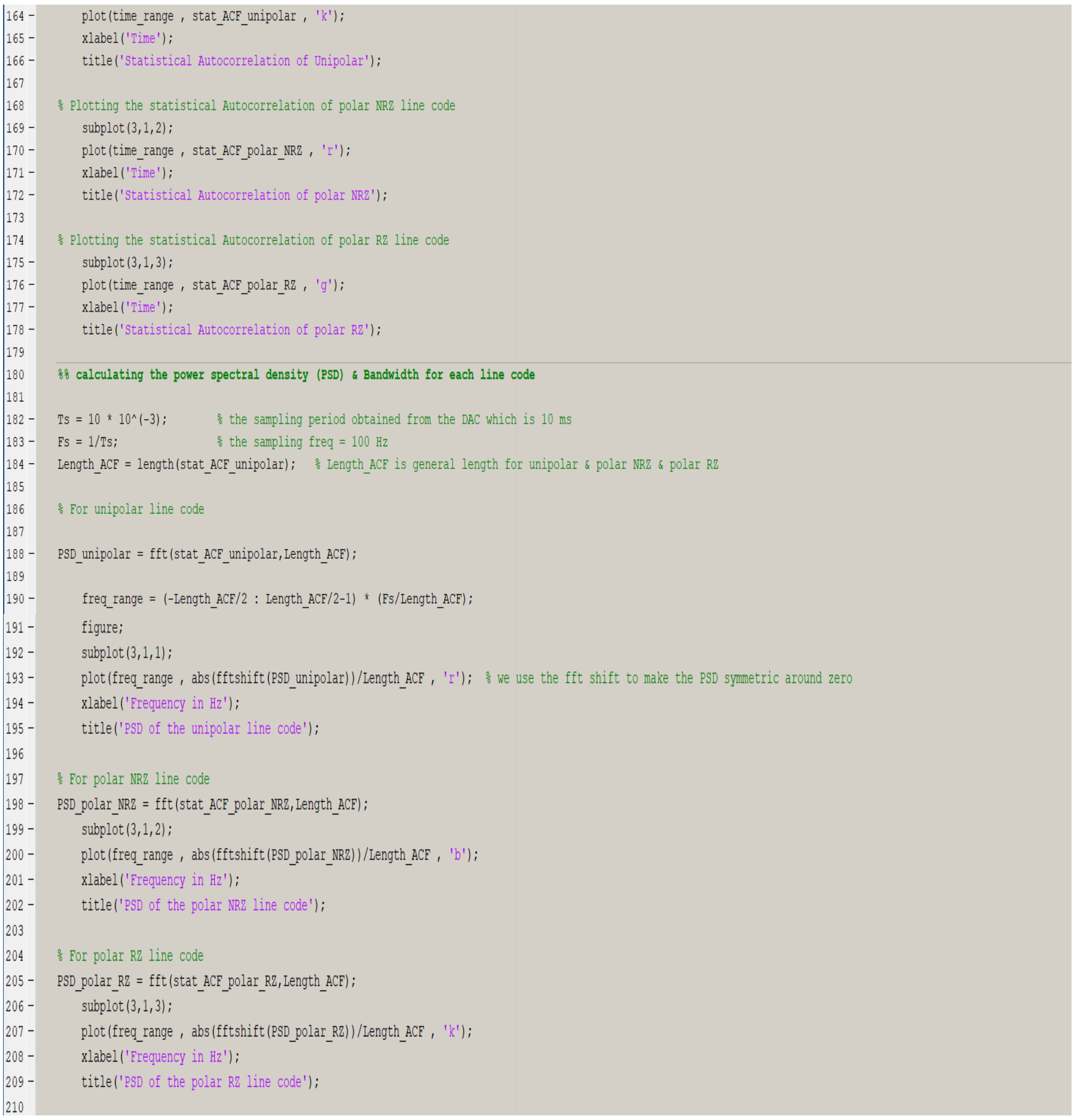
```matlab

% initialize each line coding with zeros
Tout_unipolar = zeros(Number_of_realizations,Number_of_bits*Number_of_samples);
Tout_polar_NRZ = zeros(Number_of_realizations,Number_of_bits*Number_of_samples);
Tout_polar_RZ = zeros(Number_of_realizations,Number_of_bits*Number_of_samples);
% Adding the time delay (time shift Td) to each line coding
for i = 1 : Number_of_realizations
    Tout_unipolar(i,:) = Tx1_unipolar(i , (Time_Delay(i)+1) : (Number_of_bits*Number_of_samples+Time_Delay(i)));
    Tout_polar_NRZ(i,:) = Tx2_polar_NRZ(i , (Time_Delay(i)+1) : (Number_of_bits*Number_of_samples+Time_Delay(i)));
    Tout_polar_RZ(i,:) = Tx3_polar_RZ(i , (Time_Delay(i)+1) : (Number_of_bits*Number_of_samples+Time_Delay(i)));
end

%% calculating the Statistical Mean for different line codes

% for unipolar line code
Statistical_mean_unipolar = zeros(1,Number_of_bits*Number_of_samples);
for j = 1 : (Number_of_bits*Number_of_samples)
    Statistical_mean_unipolar(1,j) = Statistical_mean_unipolar(1,j) +  (sum(Tout_unipolar(:,j))/Number_of_realizations);
end
 % Plotting the statistical mean of unipolar line code
 time = 1 : (Number_of_bits * Number_of_samples) ;  % time vector refer to the x-axis
    figure;
    subplot(3,1,1);
    plot( time , Statistical_mean_unipolar,'r')
    xlabel('Time');
    title('Statisitcal Mean of Unipolar Line code');

```



```
 % for polar NRZ line code
Statistical_mean_polar_NRZ = zeros(1,Number_of_bits*Number_of_samples);
for j = 1 : (Number_of_bits*Number_of_samples)
   Statistical_mean_polar_NRZ(1,j) = Statistical_mean_polar_NRZ(1,j) +  (sum(Tout_polar_NRZ(:,j))/Number_of_realizations);
end
 % Plotting the statistical mean of polar NRZ line code
    subplot(3,1,2);
    plot( time , Statistical_mean_polar_NRZ,'b')
    xlabel('Time');
    title('Statisitcal Mean of Polar NRZ Line code');

 % for polar RZ line code
Statistical_mean_polar_RZ = zeros(1,Number_of_bits*Number_of_samples);
for j = 1 : (Number_of_bits*Number_of_samples)
   Statistical_mean_polar_RZ(1,j) = Statistical_mean_polar_RZ(1,j) + (sum(Tout_polar_RZ(:,j))/Number_of_realizations);
end
 % Plotting the statistical mean of polar NRZ line code
    subplot(3,1,3);
    plot( time , Statistical_mean_polar_NRZ,'k')
    xlabel('Time');
    title('Statisitcal Mean of Polar RZ Line code');

%% calculating the Time Mean for different line codes

%for unipolar line code
time_mean_unipolar = zeros(Number_of_realizations,1);
for i = 1 : Number_of_realizations
  time_mean_unipolar(i,1) =  time_mean_unipolar(i,1) + sum(Tout_unipolar(i,:))/(Number_of_bits*Number_of_samples);
end

%for polar NRZ line code
time_mean_polar_NRZ = zeros(Number_of_realizations,1);
for i = 1 : Number_of_realizations
  time_mean_polar_NRZ(i,1) =  time_mean_polar_NRZ(i,1) + sum(Tout_polar_NRZ(i,:))/(Number_of_bits*Number_of_samples);
end

%for polar RZ line code
time_mean_polar_RZ = zeros(Number_of_realizations,1);
for i = 1 : Number_of_realizations
  time_mean_polar_RZ(i,1) =  time_mean_polar_RZ(i,1) + sum(Tout_polar_RZ(i,:))/(Number_of_bits*Number_of_samples);
end

% Plotting the time mean of unipolar line code
Realization_number = 1 : Number_of_realizations ;
    figure;
    subplot(3,1,1);
    plot( Realization_number , time_mean_unipolar,'c')
    xlabel('Number of Realizations');
    title('Time Mean of Unipolar Line code');

% Plotting the time mean of polar NRZ line code
    subplot(3,1,2);
    plot( Realization_number , time_mean_polar_NRZ ,'y')
    xlabel('Number of Realizations');
```

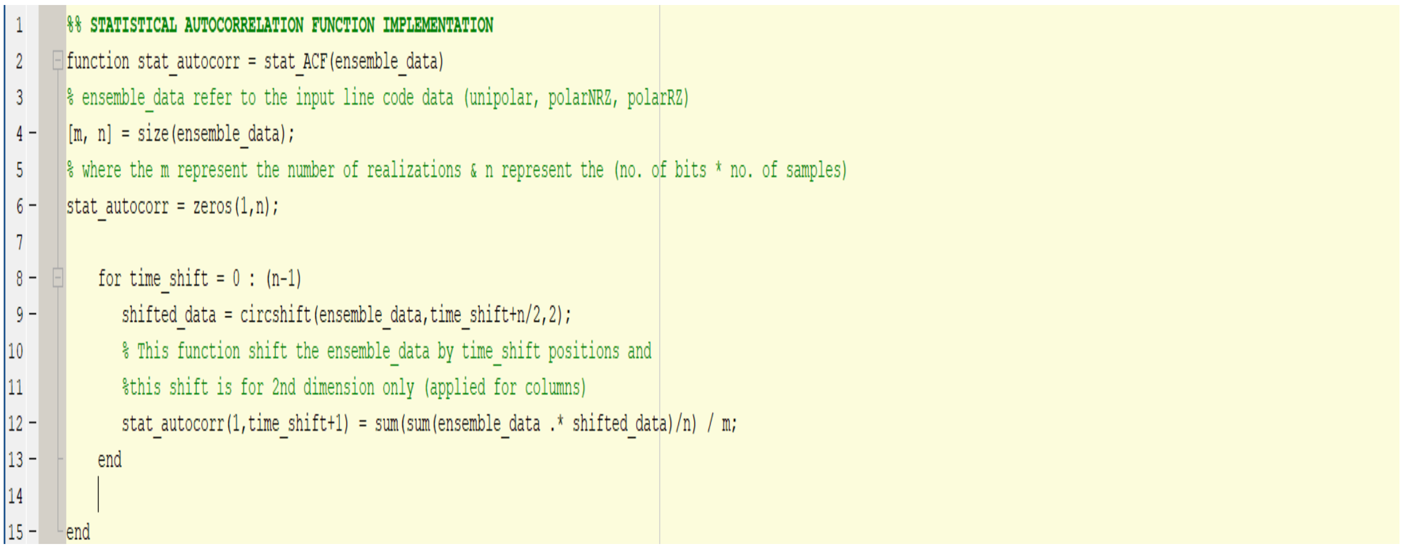


```
    title('Time Mean of polar NRZ Line code');


% Plotting the time mean of polar RZ line code
    subplot(3,1,3);
    plot( Realization_number , time_mean_polar_RZ ,'r')
    xlabel('Number of Realizations');
    title('Time Mean of polar RZ Line code');

%% calculating the Time Autocorrelation Function for different line codes  (For single realization only)

% while(1)
%     realization_value = input('please enter the realization number range from 1 --> 500to bring its time autocorrelation with different line codes : ');
%     if (realization_value < 1 || realization_value > 500)
%         fprintf('invalid value , please try again');
%     else
%         break;
%     end
% end

% for unipolar line code
time_ACF_unipolar = time_ACF(Tout_unipolar,1);        % calling the autocorrelation function for different line codes
time_ACF_polar_NRZ = time_ACF(Tout_polar_NRZ,1);
time_ACF_polar_RZ = time_ACF(Tout_polar_RZ,1);

% Plotting the time Autocorrelation of unipolar line code
time_range = -(Number_of_bits*Number_of_samples)/2 : (Number_of_bits*Number_of_samples-1)/2;
    figure;
    subplot(3,1,1);
    plot(time_range , time_ACF_unipolar, 'k');   % we concatenate the time autocorrelation after flipping it as the ACF is even function
    xlabel('Time');
    title('Time Autocorrelation of unipolar');

% Plotting the time Autocorrelation of polar NRZ line code
    subplot(3,1,2);
    plot(time_range , time_ACF_polar_NRZ , 'g');
    xlabel('Time');
    title('Time Autocorrelation of polar NRZ');

% Plotting the time Autocorrelation of polar RZ line code
    subplot(3,1,3);
    plot(time_range , time_ACF_polar_RZ , 'b');
    xlabel('Time');
    title('Time Autocorrelation of polar RZ');

%% calculating the Statistical Autocorrelation Function for different line codes

stat_ACF_unipolar = stat_ACF(Tout_unipolar);
stat_ACF_polar_NRZ = stat_ACF(Tout_polar_NRZ);
stat_ACF_polar_RZ = stat_ACF(Tout_polar_RZ);

% Plotting the statistical Autocorrelation of unipolar line code
    figure;
    subplot(3,1,1);
```



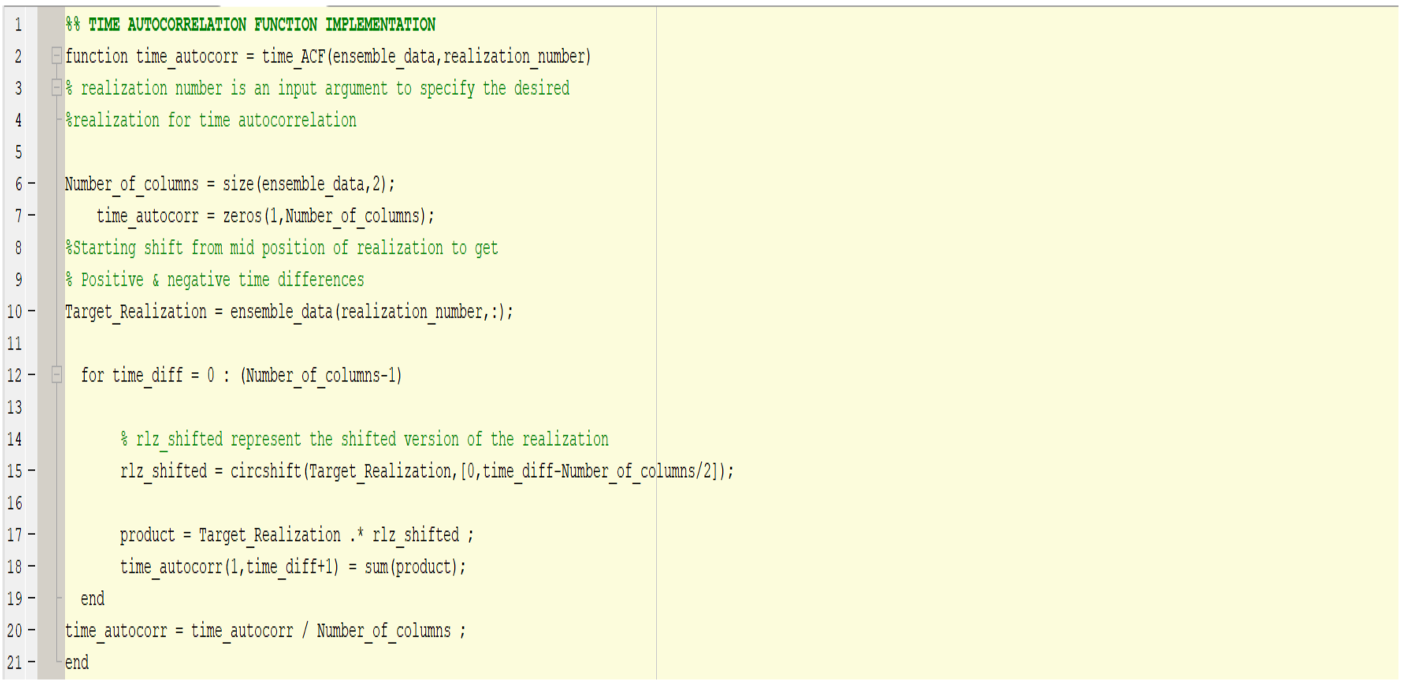
```
    plot(time_range , stat_ACF_unipolar , 'k');
    xlabel('Time');
    title('Statistical Autocorrelation of Unipolar');

% Plotting the statistical Autocorrelation of polar NRZ line code
    subplot(3,1,2);
    plot(time_range , stat_ACF_polar_NRZ , 'r');
    xlabel('Time');
    title('Statistical Autocorrelation of polar NRZ');

% Plotting the statistical Autocorrelation of polar RZ line code
    subplot(3,1,3);
    plot(time_range , stat_ACF_polar_RZ , 'g');
    xlabel('Time');
    title('Statistical Autocorrelation of polar RZ');

%% calculating the power spectral density (PSD) & Bandwidth for each line code

Ts = 10 * 10^(-3);         % the sampling period obtained from the DAC which is 10 ms
Fs = 1/Ts;                 % the sampling freq = 100 Hz
Length_ACF = length(stat_ACF_unipolar);   % Length_ACF is general length for unipolar & polar NRZ & polar RZ

% For unipolar line code

PSD_unipolar = fft(stat_ACF_unipolar,Length_ACF);

    freq_range = (-Length_ACF/2 : Length_ACF/2-1) * (Fs/Length_ACF);
    figure;
    subplot(3,1,1);
    plot(freq_range , abs(fftshift(PSD_unipolar))/Length_ACF , 'r');  % we use the fft shift to make the PSD symmetric around zero
    xlabel('Frequency in Hz');
    title('PSD of the unipolar line code');

% For polar NRZ line code
PSD_polar_NRZ = fft(stat_ACF_polar_NRZ,Length_ACF);
    subplot(3,1,2);
    plot(freq_range , abs(fftshift(PSD_polar_NRZ))/Length_ACF , 'b');
    xlabel('Frequency in Hz');
    title('PSD of the polar NRZ line code');

% For polar RZ line code
PSD_polar_RZ = fft(stat_ACF_polar_RZ,Length_ACF);
    subplot(3,1,3);
    plot(freq_range , abs(fftshift(PSD_polar_RZ))/Length_ACF , 'k');
    xlabel('Frequency in Hz');
    title('PSD of the polar RZ line code');
```

- Statistical Auto correlation function implementation:



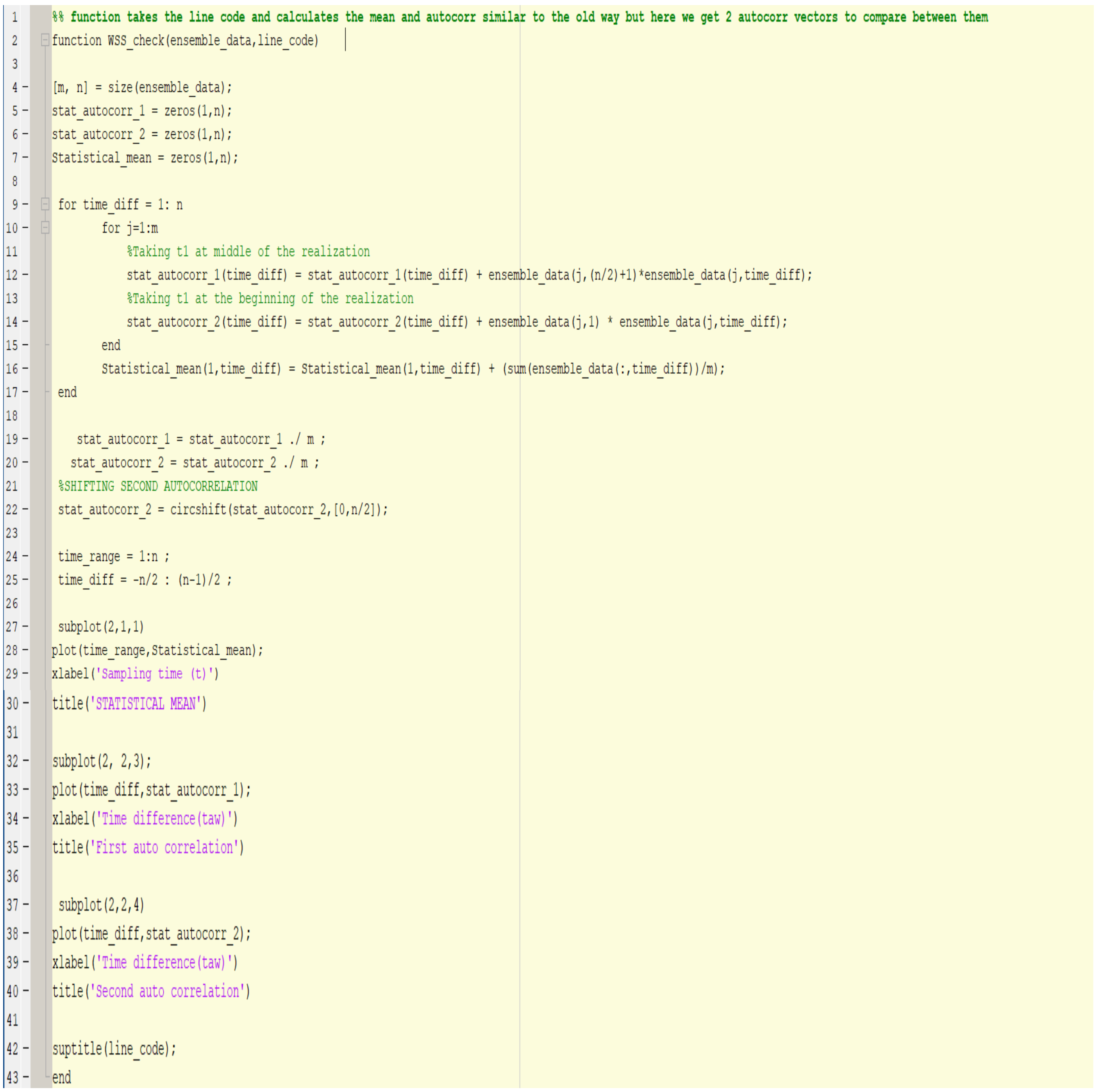
```
%% STATISTICAL AUTOCORRELATION FUNCTION IMPLEMENTATION
function stat_autocorr = stat_ACF(ensemble_data)
% ensemble_data refer to the input line code data (unipolar, polarNRZ, polarRZ)
[m, n] = size(ensemble_data);
% where the m represent the number of realizations & n represent the (no. of bits * no. of samples)
stat_autocorr = zeros(1,n);

    for time_shift = 0 : (n-1)
       shifted_data = circshift(ensemble_data,time_shift+n/2,2);
       % This function shift the ensemble_data by time_shift positions and
       %this shift is for 2nd dimension only (applied for columns)
       stat_autocorr(1,time_shift+1) = sum(sum(ensemble_data .* shifted_data)/n) / m;
    end

end
```

- Time Auto correlation function implementation:



```
%% TIME AUTOCORRELATION FUNCTION IMPLEMENTATION
function time_autocorr = time_ACF(ensemble_data,realization_number)
% realization number is an input argument to specify the desired
%realization for time autocorrelation

Number_of_columns = size(ensemble_data,2);
    time_autocorr = zeros(1,Number_of_columns);
%Starting shift from mid position of realization to get
% Positive & negative time differences
Target_Realization = ensemble_data(realization_number,:);

  for time_diff = 0 : (Number_of_columns-1)

       % rlz_shifted represent the shifted version of the realization
       rlz_shifted = circshift(Target_Realization,[0,time_diff-Number_of_columns/2]);

       product = Target_Realization .* rlz_shifted ;
       time_autocorr(1,time_diff+1) = sum(product);
  end
time_autocorr = time_autocorr / Number_of_columns ;
end
```

- **Separate Function to check that the random process is wide sense stationary:**



```
%% function takes the line code and calculates the mean and autocorr similar to the old way but here we get 2 autocorr vectors to compare between them
function WSS_check(ensemble_data,line_code)

[m, n] = size(ensemble_data);
stat_autocorr_1 = zeros(1,n);
stat_autocorr_2 = zeros(1,n);
Statistical_mean = zeros(1,n);

 for time_diff = 1: n
        for j=1:m
            %Taking t1 at middle of the realization
            stat_autocorr_1(time_diff) = stat_autocorr_1(time_diff) + ensemble_data(j,(n/2)+1)*ensemble_data(j,time_diff);
            %Taking t1 at the beginning of the realization
            stat_autocorr_2(time_diff) = stat_autocorr_2(time_diff) + ensemble_data(j,1) * ensemble_data(j,time_diff);
        end
        Statistical_mean(1,time_diff) = Statistical_mean(1,time_diff) + (sum(ensemble_data(:,time_diff))/m);
 end

    stat_autocorr_1 = stat_autocorr_1 ./ m ;
   stat_autocorr_2 = stat_autocorr_2 ./ m ;
 %SHIFTING SECOND AUTOCORRELATION
 stat_autocorr_2 = circshift(stat_autocorr_2,[0,n/2]);

 time_range = 1:n ;
 time_diff = -n/2 : (n-1)/2 ;

 subplot(2,1,1)
plot(time_range,Statistical_mean);
xlabel('Sampling time (t)')
title('STATISTICAL MEAN')

subplot(2, 2,3);
plot(time_diff,stat_autocorr_1);
xlabel('Time difference(taw)')
title('First auto correlation')

 subplot(2,2,4)
plot(time_diff,stat_autocorr_2);
xlabel('Time difference(taw)')
title('Second auto correlation')

suptitle(line_code);
end
```