Faculty of Enigeering

Cairo University

Digital Communications Project 1

Transmit Path & Line Codes

Presented for **ELC 3070** MATLAB Project

Presented to:

Dr. Mohamed Nafea

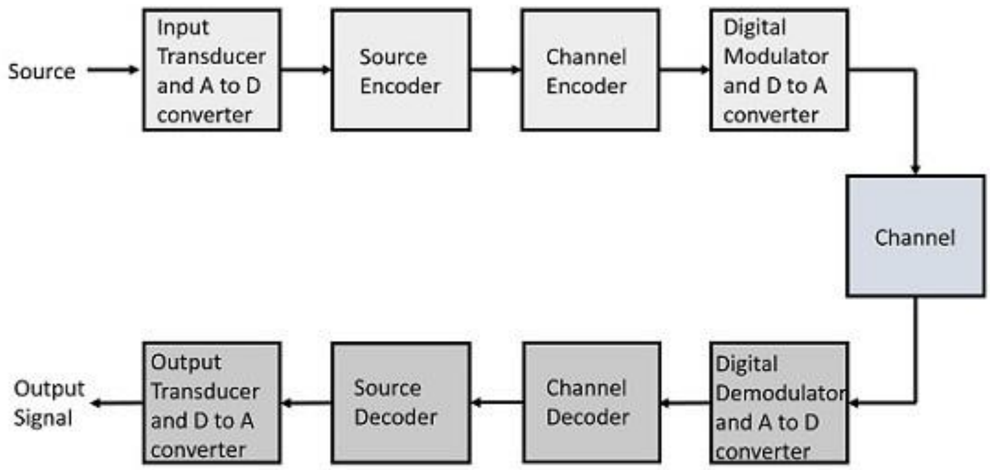
T.A: Mohamed Khaled

(3rd Year Electronics and Electrical Communication Engineers)	
مجدي أحمد عباس عبد الحميد الابرق	Sec: 3 / I.D: 9210899 / BN: 36
كريم ايمن محمد فخر الدين محمد علي عفيفي	Sec: 3 / I.D: 9210836 / BN: 26
مصطفى ابراهيم محمد ابراهيم	Sec: 4 / I.D: 9211158 / BN: 19
يحيى خالد عبد الفتاح محمد	Sec: 4 / I.D: 9211362 / BN: 40

Role of each member:

Each one of us created his own code, and in one meeting we came together on the best version by merging the 4 codes and wrote the documentation

0

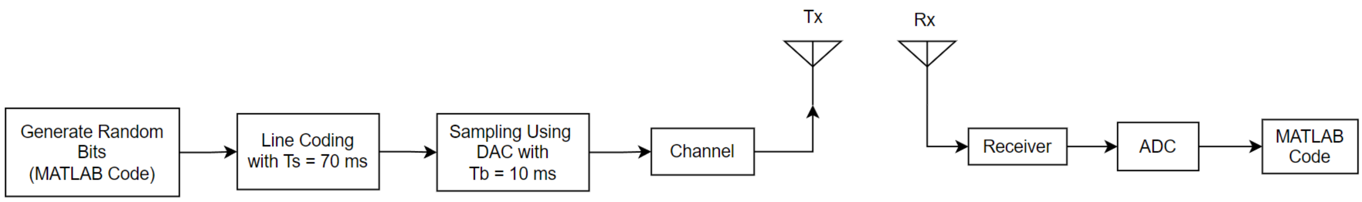


Comm Project 1

I

Problem Description

Transmitter & Receiver Path



In our system, there's a transmitter (Tx) and a receiver (Rx) connected by a channel. We create three ensembles, each using a different line code (unipolar, polar NRZ, polar RZ), where the transmitter generates random bits. These bits are turned into continuous signals, but since MATLAB works with arrays and vectors, we discretize them using a Digital to Analog Converter (DAC) with a sampling period of 10 ms. Each sample in the line code lasts 70 ms, resulting in 7 bits per sample. Our goal is to send these bits through a band-limited channel without interference. At the receiver (Rx), the transmitted bits are received, and the receiver tries to accurately estimate them with minimal error probability. Moreover, the receiver aims to find the most efficient line code in terms of bandwidth usage.

II

Introduction

The process begins by creating a sequence of random binary bits based on a specified flag indicating the number of bits per waveform (Realization). Each bit is then replicated according to another flag determining the number of samples per bit. These generated bits are then converted into corresponding line code symbols. Next, a random time shift is introduced, controlled by a time delay flag, to provide each waveform (Realization) with a random starting point. This entire process is iterated a specified number of times, governed by an ensemble size flag. The resulting ensemble is utilized to compute statistical metrics such as mean and autocorrelation. It's assessed whether each line code is stationary and ergodic, with ergodic processes being preferred as they simplify statistical analysis by requiring only a single realization rather than an entire ensemble.

III

Control Flags

Flag Name in MATLAB	Flag Value	Its Function
A	User defined	Controls the Signal Amplitude of the pulses in the line codes
Frame Length	100	Controls the Number of Bits in each Waveform (Realization)
DAC Samples	7	Controls the Number of Samples for each Bit in each Realization DAC Samples = $\frac{\text{DAC Sampling Frequency}}{\text{Bit Rate}}$
Ensemble Width	500	Controls the Number of Waveforms (Realizations) in the Ensemble
Delay period	Random value between 0 to 6	Controls Where each Realization Randomly Starts From 0 to DAC Samples - 1 ($0 \rightarrow T_b$)
Symbol Rate	$\mathrm{R_s} = \frac{100}{7}\ \mathrm{HZ}$	Is the Rate of Line Coding by Tx (User)
Bit Rate	$\mathrm{R_b} = 100\ \mathrm{HZ}$	Is the Rate of The DAC Sampling

Table 1: Control Flags Table

```
%% General Notes
frame_length = 100; %Required no. of tranmitted bits
Ensemble_width = 500; %Number of realizations
DAC_samples = 7; % Dac samples = Dac sampling freq / Bit rate = (70 ms)/(10 ms)
cond = 1;
while(cond)
A = input(" Enter the amplitude of the line code : "); % Rails
if(A > 0 && A < 10)
   cond = 0;
   else
      fprintf(" Wrong Input A in ]0,10] , please try again .....\n");
   end
end
final_frame_length = frame_length * DAC_samples; %The frame length after the DAC
Rs = (10^3)/70; %(1/Ts) = 1/70ms --> Symbol rate
Rb = (10^3)/10; %(1/Tb = n/Ts) = 1/10ms --> bit rate
```

Snippet 1: Control Flags Snippet

User Defined A = 4 in this Report for Run

```
Enter the amplitude of the line code : 4
```

IV

Generation of Data

```
%% Generation of the general random Delay & Tx stream (not mapped yet)
% Add extra bit is used to introduce the delay given by the last transmitted frame
tx = randi([0 1], Ensemble_width, frame_length + 1);
tx_DAC_out = repelem(tx, 1, DAC_samples);
```

Snippet 2: Generation of Data (Tx Stream) Snippet

This Snippet describes the process of generating random binary transmission sequences for a digital communication system. It creates a matrix named "tx" where each row represents a different transmission sequence, and each column represents a symbol in the sequence. These symbols are randomly chosen from 0 and 1, forming a random digital transmission stream. The size of "tx" is determined by parameters such as “Ensemble_width” (the number of realizations) and “frame_length + 1” (the length of each transmission frame, plus an extra bit for delay). This extra bit accounts for a delay introduced by the last transmitted frame. Subsequently, each symbol in the "tx" matrix is repeated "DAC_samples" times to simulate the conversion to analog waveforms using a digital-to-analog converter (DAC). The resulting matrix "tx_DAC_out" contains the digital transmission stream repeated multiple times, representing an analog waveform ready for further processing in the digital communication system. Overall, this process prepares random binary transmission streams for multiple realizations, ensuring they are converted to analog waveforms for subsequent system processing.

Creation of Line Codes Ensemble

```
%% Choose the required Signaling
Signaling_line_codes = {'Unipolar NRZ', 'Polar NRZ', 'Polar RZ'};
[indicies, check]
= listdlg('ListString', Signaling_line_codes, 'promptstring', 'Select the line code to continue (Unipolar
if (~check)
  indicies = 1 ;
end
switch cell2mat(Signaling_line_codes(indicies))
  case 'Unipolar NRZ'
    Signaling_mapped = tx_final * A;  % Unipolar NRZ Signaling Line Code
  case 'Polar NRZ'
    Signaling_mapped = (2 * tx_final − 1) * A;  % polar NRZ Signaling Line Code
  case 'Polar RZ'
    Signaling_mapped
              = (2 * tx_final − 1) * A; % polar RZ (Take NRZ and replace some bits with Zeros)
    for k = 5: 7: final_frame_length − 2
      for l = k: k + 2
        Signaling_mapped(: , l) = 0;
      end
    end
end
```

Snippet 3: Line Codes Signaling Snippet

This MATLAB snippet allows the user to choose a signaling line code for further processing. Here's a breakdown of how it works:


```
Signaling_line_codes = {'Unipolar NRZ','Polar NRZ','Polar RZ'};
```

- This line creates a cell array named Signaling_line_codes containing the names of three different signaling line codes: 'Unipolar NRZ', 'Polar NRZ', and 'Polar RZ'.

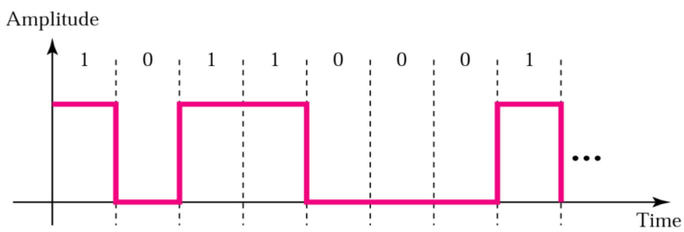
```
[indicies, check] = listdlg('ListString', Signaling_line_codes, 'promptstring',
'Select the line code to continue (Unipolar Signaling by default)',
'SelectionMode','single');
```

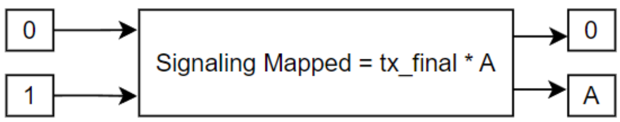
- This line opens a dialog box prompting the user to select one of the signaling line codes from the options provided in the Signaling_line_codes cell array.
- The listdlg function returns the index of the selected item in indices and a Boolean indicating whether the user made a selection in check.

```
if (~check)
  indicies = 1 ;
end
```

- This block of code checks if the user made a selection (check is true). If the user didn't make a selection (meaning check is false), it defaults the index indices to 1, which corresponds to 'Unipolar NRZ' signaling line code.

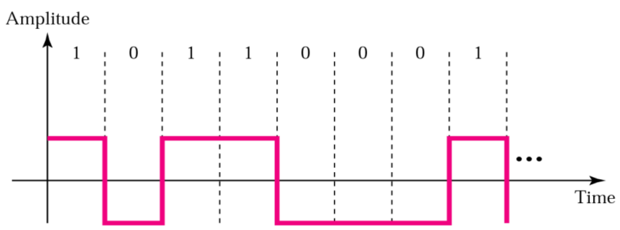
V Unipolar Signaling

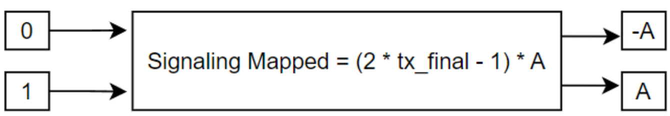




Unipolar NRZ Signaling Line Code If indices is 1

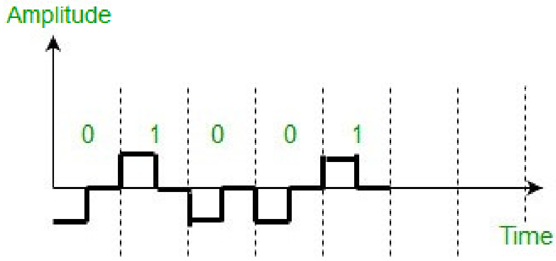
VI Polar NRZ Signaling

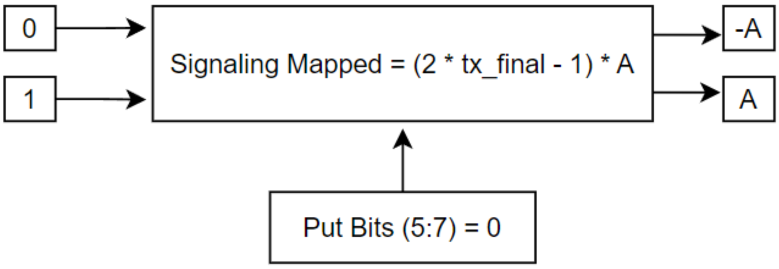




Polar NRZ Signaling Line Code If indices is 2

VII Polar RZ Signaling



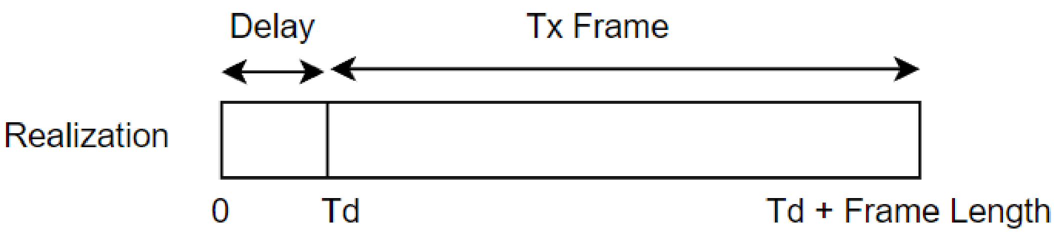


Polar RZ Signaling Line Code If indices is 3

VIII Applying Random Initial Shifts for each Waveform

```
% Delay the realizations
delay_period = randi([0 DAC_samples − 1],1, Ensemble_width);
tx_delayed = zeros(Ensemble_width, size(tx, 2) ∗ DAC_samples);
for i = 1: Ensemble_width
    tx_delayed(i, : ) = circular_shift(tx_DAC_out(i, : ), delay_period(i));
end
% pickup the shifted frame
for i = 1: Ensemble_width
   tx_final = tx_delayed(: , delay_period(i) + 1: delay_period(i) + final_frame_length);
end
%% Aiding Functions
function z = circular_shift(x, shift_val) %circular shift the realizations with the delay
len = length(x);
z = zeros(1, len);
   if(shift_val~ = 0)
      start = len − (shift_val − 1);
      i = start;
      j = 1;
      while (j <= len)
         z(j) = x(i);
         j = j + 1;
         i = i + 1;
         if(i > len)
           i = 1;
         end
      end
   else
      z = x;
   end
end
```

Snippet 4: Realizations Delay Snippet



This code section starts by generating random delay periods for each realization, sampled from a discrete uniform distribution between (0 and 1) $(\text{DAC\_samples} - 1) \equiv (0 \rightarrow T_b)$.
The delay periods are stored in a row vector. Then, a matrix is initialized to hold the delayed versions of the transmitted signals. Each signal is circularly shifted by its corresponding delay period to introduce the delays. Finally, the shifted frames are selected from the delayed signal matrix for further processing. Overall, this code snippet introduces random delays to transmission realizations and prepares the shifted frames for subsequent analysis.

IX

Getting the Cell arrays ready to calculate The Statistical Mean and Auto Correlation

```
%% Checking the sationarity using statistical (Mean/ACF)
Signaling_stat_mean = Stat_mean(Signaling_mapped, final_frame_length);
Signaling_stat_ACF_single_sided_positive = stat_acf(Signaling_mapped, final_frame_length);
Signaling_stat_ACF_single_sided_negative = fliplr(Signaling_stat_ACF_single_sided_positive);
Signaling_stat_ACF
            = [Signaling_stat_ACF_single_sided_negative(:,1:end - 1) Signaling_stat_ACF_single_sided_positive];
tau_domain_length = length(Signaling_stat_ACF);
tau_domain = (−((tau_domain_length − 1)/2):((tau_domain_length − 1)/2)).* (Rs/Rb);
time_domain = 0:final_frame_length − 1;
%{
   1 − To see if the mean is constant , we can check that :
    * Unipolar :
     Average of the mean across the time instants ~ A/2 & Std ~ 0
    * Polar NRZ:
     Average of the mean across the time instants ~ 0 & Std ~ 0
    * Polar RZ:
     Average of the mean across the time instants ~ 0 & Std ~ 0
   2 − We can check the statistic ACF by :
    * Unipolar :
      ACF @(tau = 0) ~ A^2/2 & DC ~ A^2/4
    * Polar NRZ:
      ACF @(tau = 0) ~ A^2 & DC ~ 0
    * Polar RZ :
      ACF @(tau = 0) ~ A^2 * 4/7 & DC ~ 0 (4/7 = Duty Cycle)
%}
str_stat_mean_Signaling = sprintf(" Average = %.3f \nStd
            = %.3f", mean(Signaling_stat_mean), std(Signaling_stat_mean));
str_stat_ACF_Signaling = sprintf(" DC = %.3f \nR(0)
            = %.3f", mean(Signaling_stat_ACF), Signaling_stat_ACF(tau_domain == 0));
```

```
%% Aiding Functions
function z = average(x)
len = length(x);
sum = 0;
for i = 1:len
  sum = sum + x(i);
end
z = sum/len;
end
function [y] = Stat_mean(x,frame_length)
  y = zeros(1,frame_length);
  for t = 1:frame_length
    y(t) = average(x(:,t));
  end
end

function [z] = stat_acf(x,frame_length)
%The code exercises the +ve & -ve sides of the ACF so it implicitly checks if it is even or not
y = zeros(frame_length,frame_length);
z = zeros(1,frame_length);
tau_hash = zeros(1,frame_length);
for t1 = 1 : frame_length
 tau = 0;
  while(1)
    t2 = t1 + tau;
    y(t1,abs(tau) + 1) = average(x(:,t1).* x(:,t2));
    tau_hash(abs(tau) + 1) = tau_hash(abs(tau) + 1) + 1;
    if(t1 <= (frame_length/2))
      tau = tau + 1;
    else
      tau = tau - 1;
    end
    if((t1 + tau) > frame_length || (t1 + tau) < 1)
      break;
    end
  end
end
    sum_instants = sum(y);
    for i = 1:frame_length
      z(i) = sum_instants(i)/tau_hash(i);
    end
end
```

Snippet 5: Preparation of Functions for mean and Auto Correlation Calculations

This MATLAB code segment aims to analyze the stationarity of a signal using statistical methods, particularly focusing on mean and autocorrelation function (ACF) analysis. Here's a breakdown of the code:

1. **Compute Mean and ACF:**
   - $\mathbf{Signaling\_stat\_mean = Stat\_mean(Signaling\_mapped, final\_frame\_length)}$: Calculates the mean of the signal across different time instants.
   - $\mathbf{Signaling\_stat\_ACF\_single\_sided\_positive = stat\_acf(Signaling\_mapped, final\_frame\_length)}$: Computes the autocorrelation function (ACF) for positive part.
   - $\mathbf{Signaling\_stat\_ACF\_single\_sided\_negative = fliplr(Signaling\_stat\_ACF\_single\_sided\_positive)}$: Reverses the ACF for negative part.
   - $\mathbf{Signaling\_stat\_ACF = [Signaling\_stat\_ACF\_single\_sided\_negative(:, 1: end - 1)\ Signaling\_stat\_ACF\_single\_sided\_positive]}$: Combines the positive and negative ACF to obtain the complete ACF.

2. **Generate Domains:**
   - $\mathbf{tau\_domain\_length = length(Signaling\_stat\_ACF)}$: Determines the length of the ACF.
   - $\mathbf{tau\_domain = -((tau\_domain\_length - 1)/2): ((tau\_domain\_length - 1)/2).* (Rs/Rb)}$
   - Generates the domain (parts) for the ACF.
   - $\mathbf{time\_domain = 0: final\_frame\_length - 1}$: Generates the time domain for the signal.

3. **Stationarity Criteria:**
   - In order to assess the mean and the auto correlation function for the practical frame and ensemble we need to quantify the properties of the wide sense stationary in terms of the following properties

     ① Mean → Constant with Time ∴ Standard deviation ($\sigma = 0$)
     ② Mean has Certain Theoretical value.
     ∴ Average across time instants ≈ This Theoretical value.

   - In the Comments provide guidelines to interpret the mean and ACF characteristics to determine stationarity for different signal types (unipolar and polar).

$$
\begin{aligned}
&1 - \text{To see if the mean is constant, we can check that:}\\
&* \text{ Unipolar:}\\
&\quad \text{Average of the mean across the time instants} \sim A/2\ \&\ \text{Std} \sim 0\\
&* \text{ Polar NRZ:}\\
&\quad \text{Average of the mean across the time instants} \sim 0\ \&\ \text{Std} \sim 0\\
&* \text{ Polar RZ:}\\
&\quad \text{Average of the mean across the time instants} \sim 0\ \&\ \text{Std} \sim 0\\
&2 - \text{We can check the statistic ACF by:}\\
&* \text{ Unipolar:}\\
&\quad \text{ACF @(tau = 0)} \sim A\text{\^{}}2/2\ \&\ \text{DC} \sim A\text{\^{}}2/4\\
&* \text{ Polar NRZ:}\\
&\quad \text{ACF @(tau = 0)} \sim A\text{\^{}}2\ \&\ \text{DC} \sim 0\\
&* \text{ Polar RZ:}\\
&\quad \text{ACF @(tau = 0)} \sim A\text{\^{}}2 * 4/7\ \&\ \text{DC} \sim 0\ (4/7 = \text{Duty Cycle})
\end{aligned}
$$

4. **Generate Strings for Display:**
   - **str_stat_mean_Signaling**: Formats mean statistics into a string for display.
   - **str_stat_ACF_Signaling**: Formats ACF statistics into a string for display.

5. **Helper Functions:**
   - **average(x)**: Computes the average of elements in an array.
   - **Stat_mean(x, frame_length)**:
     Computes the mean of each column of the input matrix.
   - **stat_acf(x, frame_length)**: Computes the autocorrelation function of the input matrix considering positive part.

```
%% plot the statistic mean & ACF
figure(1);
plot(time_domain, Signaling_stat_mean);
xlabel('Time (msec) ');
ylabel('E\{X(t)\}')
set(gca, 'XAxisLocation', 'origin');
title(sprintf("Stat %s Signaling Mean", cell2mat(Signaling_line_codes(indicies))));
a1 = annotation('textbox', [0.4 0.62 0.3 0.3], 'String',
                                    str_stat_mean_Signaling, 'FitBoxToText', 'on');
figure(2);
plot(tau_domain, Signaling_stat_ACF);
xlabel('\tau * (Rs/Rb) (ms)', 'FontSize', 12);
ylabel('R(\tau)', 'FontSize', 14);
set(gca, 'XAxisLocation', 'origin');
title(sprintf("Stat %s Signaling ACF", cell2mat(Signaling_line_codes(indicies))));
a2 = annotation('textbox', [0.7 0.62 0.3 0.3], 'String',
                                    str_stat_ACF_Signaling, 'FitBoxToText', 'on');
figure();
zoom = final_frame_length + [-50 50];
plot(tau_domain(zoom(1): zoom(2)), Signaling_stat_ACF(zoom(1): zoom(2)));
a2 = annotation('textbox', [0.7 0.62 0.3 0.3], 'String', str_stat_ACF_Signaling,
                                                        'FitBoxToText', 'on');
title(sprintf("Stat %s Signaling ACF (zoomed version)",
                                    cell2mat(Signaling_line_codes(indicies))));
xlabel('\tau * (Rs/Rb) (ms)', 'FontSize', 12);
ylabel('R(\tau)', 'FontSize', 14);
set(gca, 'XAxisLocation', 'origin');
```

Snippet 6: Mean and Auto Correlation Plotting

X + XI

Calculating and Plotting The Statistical Mean

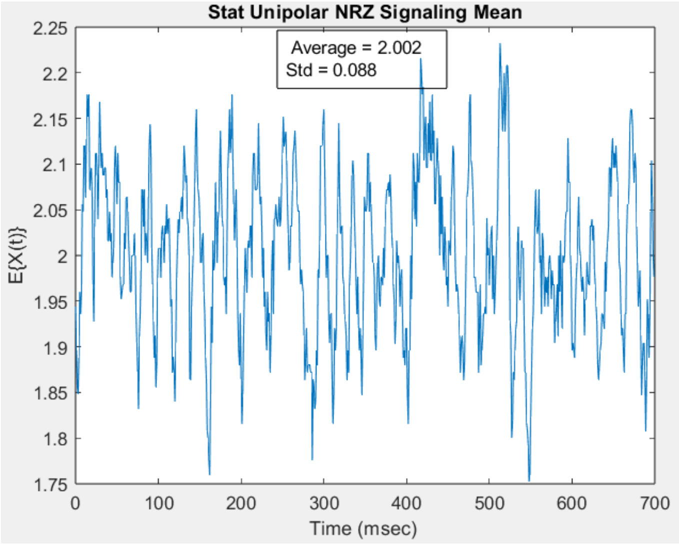
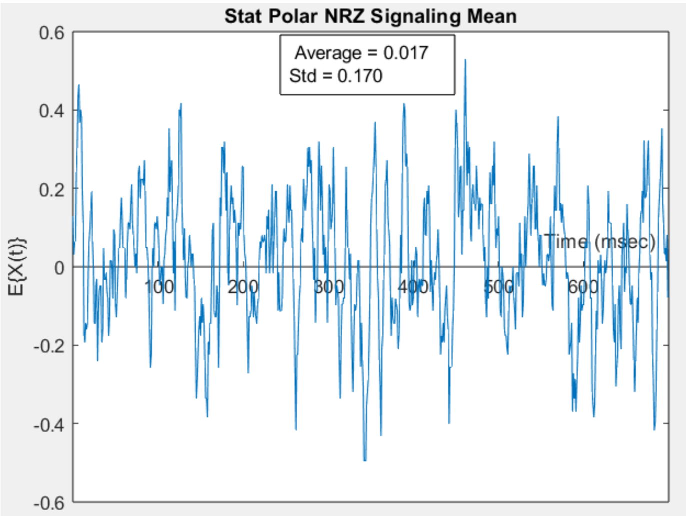


Figure 1: Stat Unipolar NRZ Signaling Mean



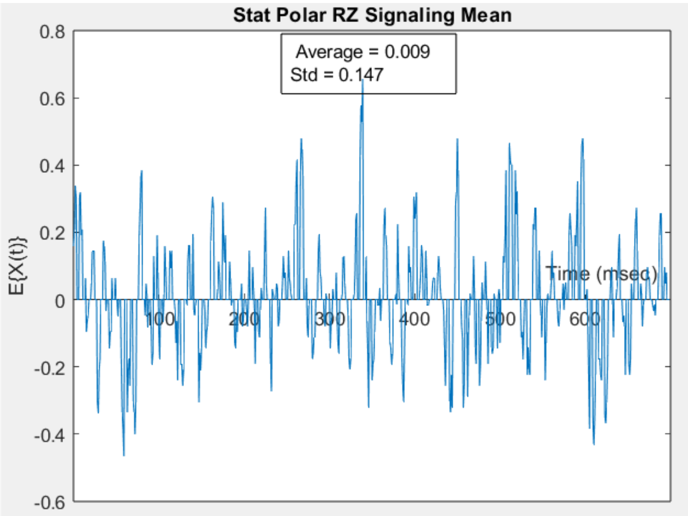


Figure 2: Stat Polar NRZ Signaling Mean

Figure 3: Stat Polar RZ Signaling Mean

ACF Calculation Concept for Statistical and Time Mean

Theoretical: There is infinite Ensemble and infinite frame (Smooth Shape for ACF).

Practical: There is a finite frame (Has less Number of Correlations for each time instant)

Disadvantages:

Ripples and distortions (results from the unequal number of correlations between time instants)
At $t_1 = 1 \rightarrow \tau_1 \in [1, \text{frame}] \rightarrow$ ACF Calculated from tau's Domain Positive part.
At $t_1 = 2 \rightarrow \tau_2 \in [1, \text{frame} - 1] \rightarrow$ and so on for which $t_1 \leq \frac{\text{frame}}{2}$, at $t_1 = 351 \rightarrow \tau_{351} \in [-351, -1]$ and we can exploit the even symmetry of the ACF by substituting the negative tau domain for the positive one this doesn't force the ACF to be even but exercises the even domain to see if the ACF will average out to the theoretical shape or not.

Expectations:

we will have an approximate waveform of the ACF with some ripples due to the practical data set.

XII + XIII

Calculating and Plotting The Statistical Auto Correlation

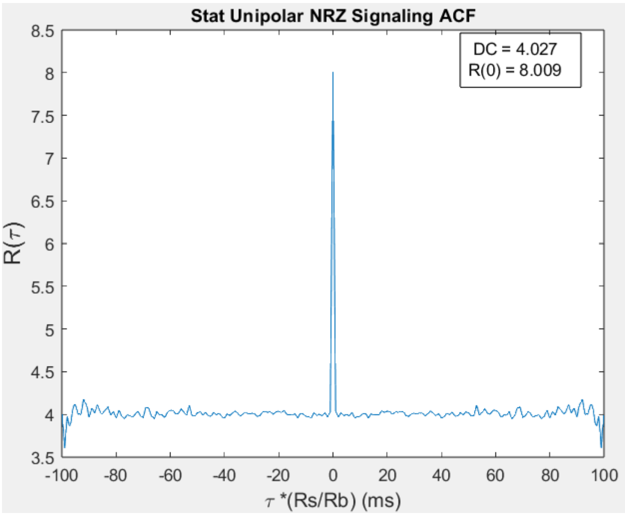


Figure 4: Stat Unipolar NRZ Signaling ACF Normalization on x - axis

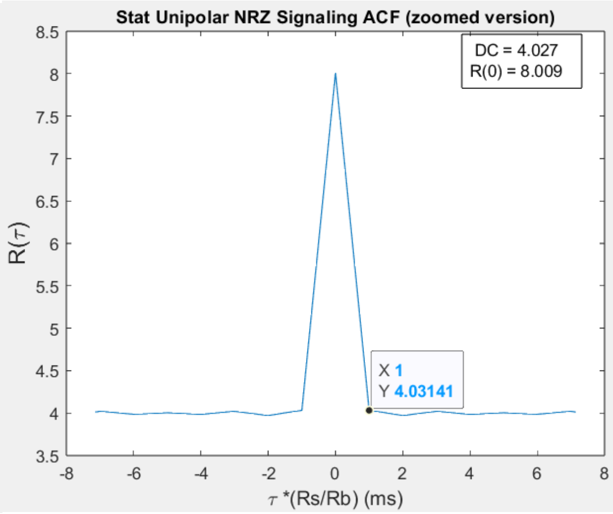


Figure 5: Stat Unipolar NRZ Signaling ACF (Zooming Version)

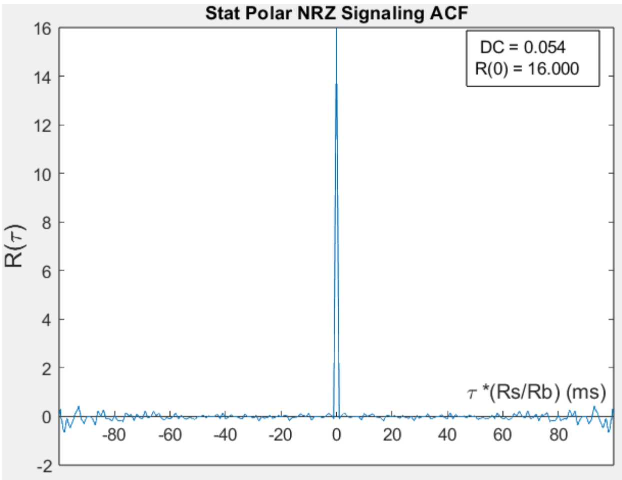


Figure 6: Stat Polar NRZ Signaling ACF Normalization on x - axis

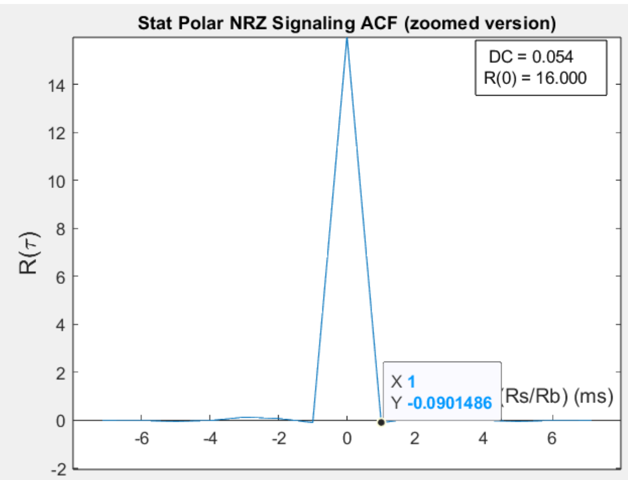


Figure 7: Stat Polar NRZ Signaling ACF (Zoomed Version)

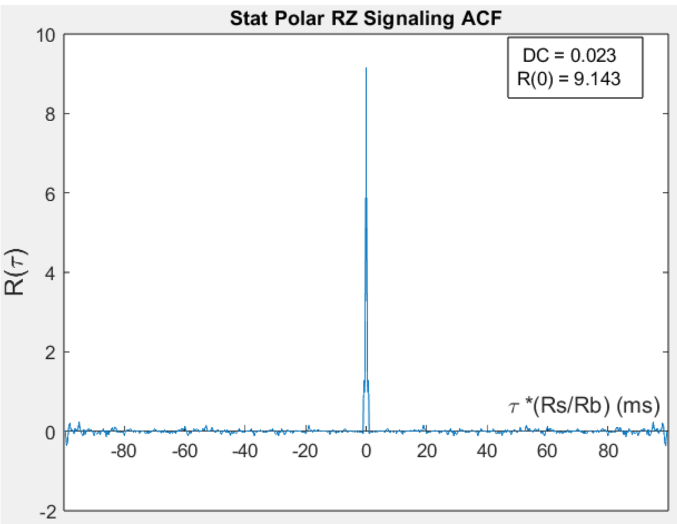


Figure 8: Stat Polar RZ Signaling ACF Normalization on x - axis

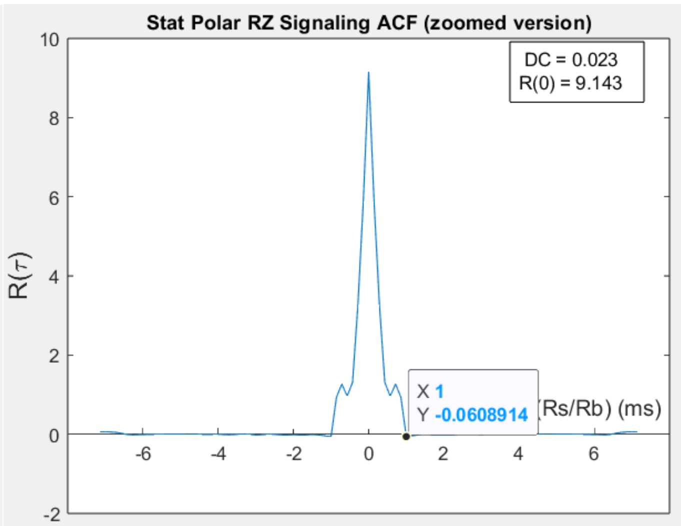


Figure 9: Stat Polar RZ Signaling ACF (Zoomed Version)

XIV

Is the Process Stationary?

P.O.C		Unipolar NRZ	Polar NRZ	Polar RZ
Mean	**Theoretical**	$\frac{A}{2} = \frac{4}{2} = 2$	0	0
	Practical	2.002	0.017	0.009
Comment: Mean is Constant with time $(\sigma = 0)$, Practical $\approx$ Theoretical				
Autocorrelation	**Theoretical**	$DC = \frac{A^2}{4} = 4$	$DC = 0$	$DC = 0$
		$R\,(0) = \frac{A^2}{2} = 8$	$R\,(0) = A^2 = 16$	$R\,(0) = A^2 \times \frac{4}{7} = 9.143$
	Practical	$DC = 4.027$	$DC = 0.054$	$DC = 0.023$
		$R\,(0) = 8.009$	$R\,(0) = 16$	$R\,(0) = 9.143$
Comment: Autocorrelation depends on time shift (τ)				
Result (WSS)		**Stationary**	**Stationary**	**Stationary**

Table 2: WSS Proof Table

XV

Computing The Time Mean & Auto Correlation of One Realization

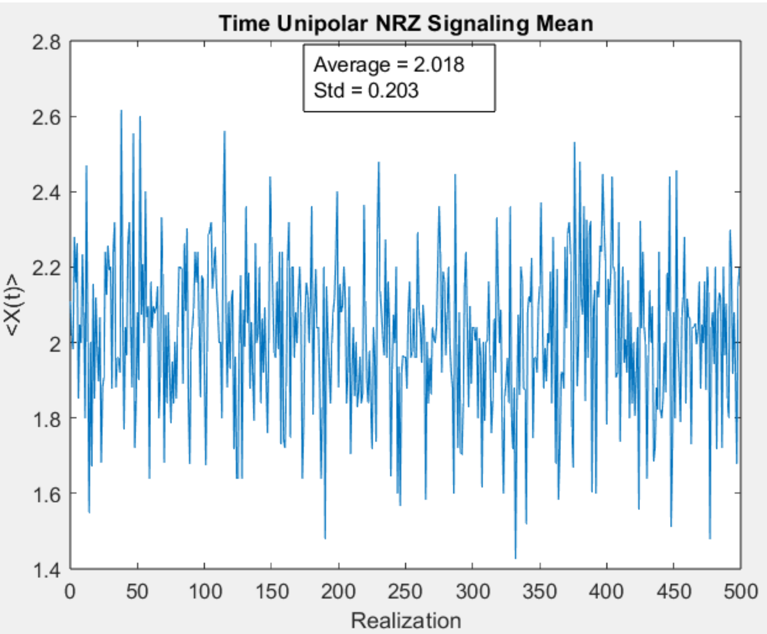


Figure 10: Time Unipolar NRZ Signaling Mean

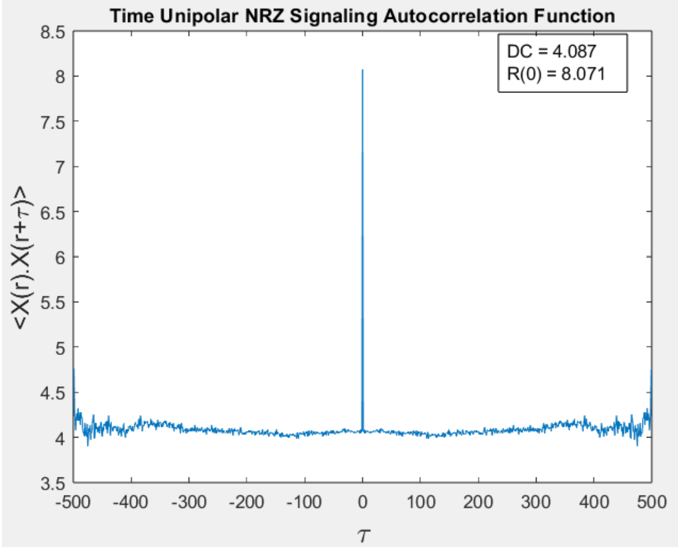


Figure 11: Time Unipolar NRZ Signaling ACF

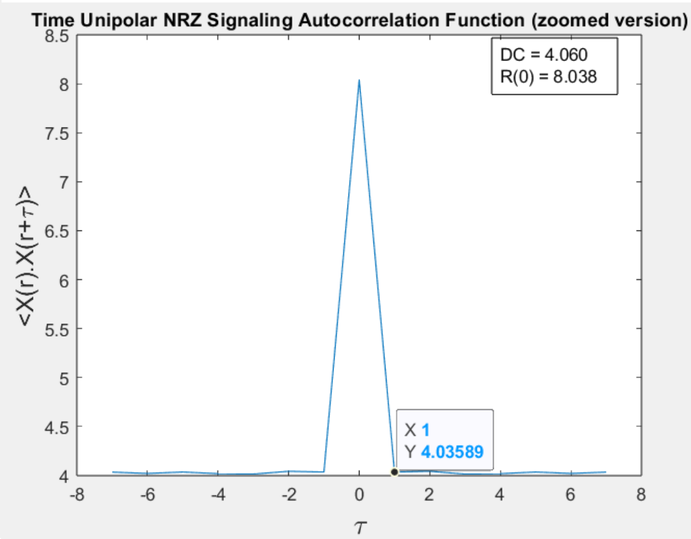


Figure 12: Time Unipolar NRZ Signaling ACF (Zoomed Version)

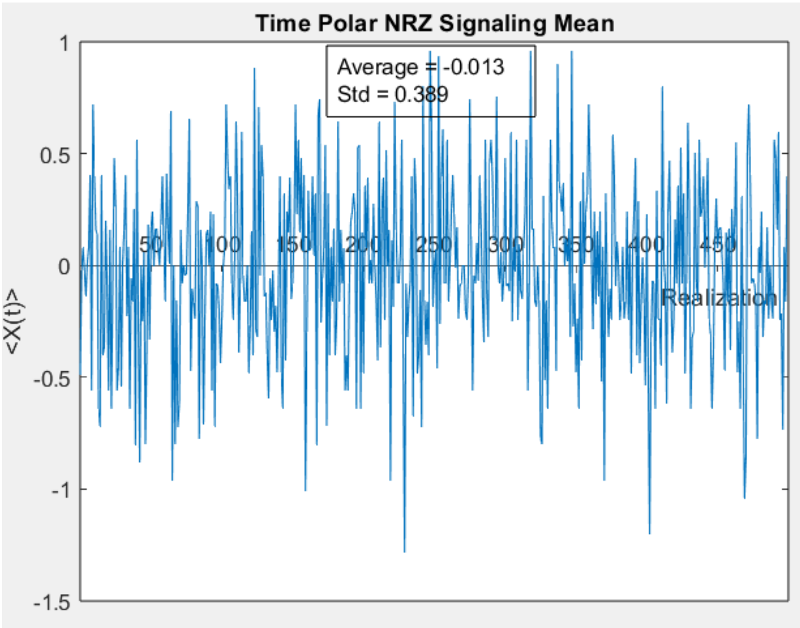


Figure 13: Time Polar NRZ Signaling Mean

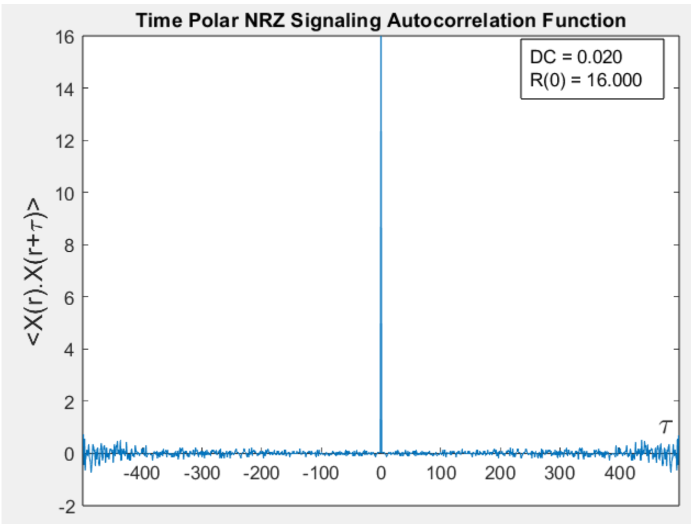


Figure 14: Time Polar NRZ Signaling ACF

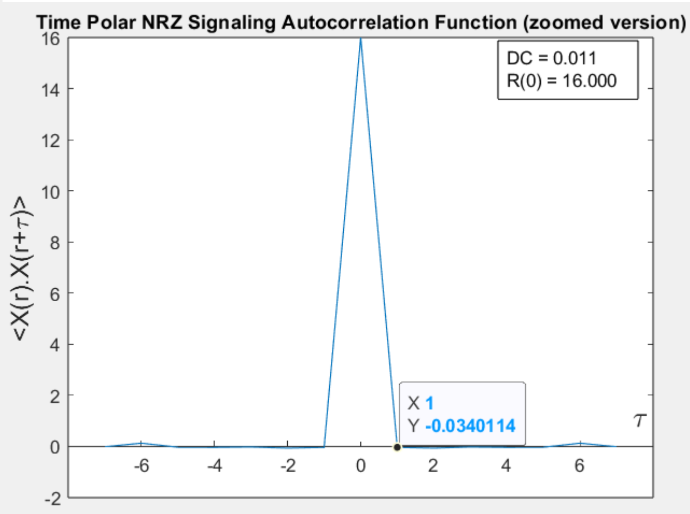


Figure 15: Time Polar NRZ Signaling ACF (Zoomed Version)

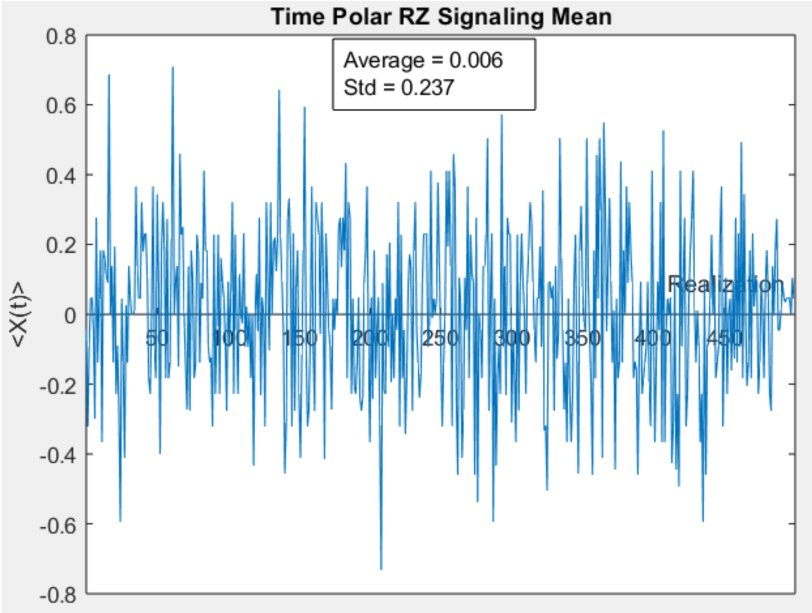


Figure 16: Time Polar RZ Signaling Mean

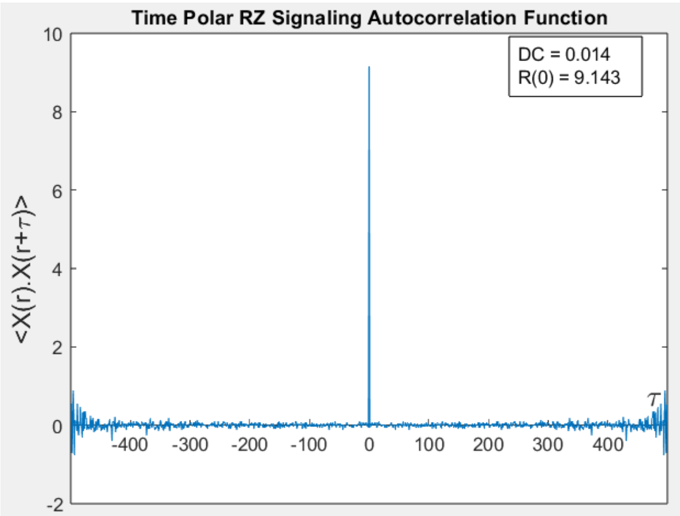


Figure 17: Time Polar RZ Signaling ACF

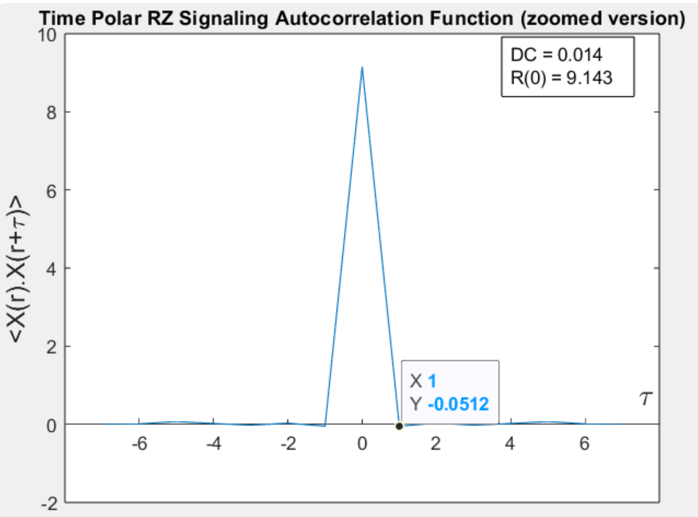


Figure 18: Time Polar RZ Signaling ACF (Zoomed Version)

```
%% Aiding Functions
function [y]  = time_mean(x, realization_count)
y  = zeros(realization_count, 1);
for r = 1: realization_count
y(r) = average(x(r, : ));
end
end
function [z]  = time_acf(x, realization_count)
y = zeros(realization_count, realization_count);
z  = zeros(1, realization_count);
tau_hash  = zeros(1, realization_count);
for n1  = 1 :  realization_count
 tau = 0;
```

```
  while(1)
    n2 = n1 + tau;
    y(n1, abs(tau) + 1) = average(x(n1, :).* x(n2, :));
    tau_hash(abs(tau) + 1) = tau_hash(abs(tau) + 1) + 1;
    if(n1 <= (realization_count/2))
      tau = tau + 1;
    else
      tau = tau - 1;
    end
    if((n1 + tau) > realization_count || (n1 + tau) < 1)
      break;
    end
  end
end
    sum_instants = sum(y);
    for i = 1: realization_count
    z(i) = sum_instants(i)/tau_hash(i);
    end
end
%% plot the time mean & ACF
figure();
plot(R_domain, Signaling_time_mean, 'LineWidth', 0.25);
xlabel('Realization');
ylabel(' < X(t) > ')
set(gca, 'XAxisLocation', 'origin');
title(sprintf("Time %s Signaling Mean", cell2mat(Signaling_line_codes(indicies))));
a3 = annotation('textbox', [0.4 0.62 0.3 0.3], 'String', str_time_mean_Signaling,
                                                              'FitBoxToText', 'on');
figure();
plot(r_domain, Signaling_time_ACF);
xlabel('\tau', 'FontSize', 16);
ylabel(' < X(r). X(r +\tau) > ', 'FontSize', 14);
set(gca, 'XAxisLocation', 'origin');
title(sprintf("Time %s Signaling Autocorrelation Function",
                                           cell2mat(Signaling_line_codes(indicies))));
a4 = annotation('textbox', [0.7 0.62 0.3 0.3], 'String', str_time_ACF_Signaling,
                                                              'FitBoxToText', 'on');
figure();
zoom = round(Ensemble_width + ([-50 50] * (Rs/Rb)));
plot(r_domain(1, zoom(1): zoom(2)), Signaling_time_ACF(1, zoom(1): zoom(2)));
xlabel('\tau', 'FontSize', 16);
ylabel(' < X(r). X(r +\tau) > ', 'FontSize', 14);
set(gca, 'XAxisLocation', 'origin');
a4 = annotation('textbox', [0.7 0.62 0.3 0.3], 'String', str_time_ACF_Signaling,
                                                              'FitBoxToText', 'on');
title(sprintf("Time %s Signaling Autocorrelation Function (zoomed version)",
                                           cell2mat(Signaling_line_codes(indicies))));
```

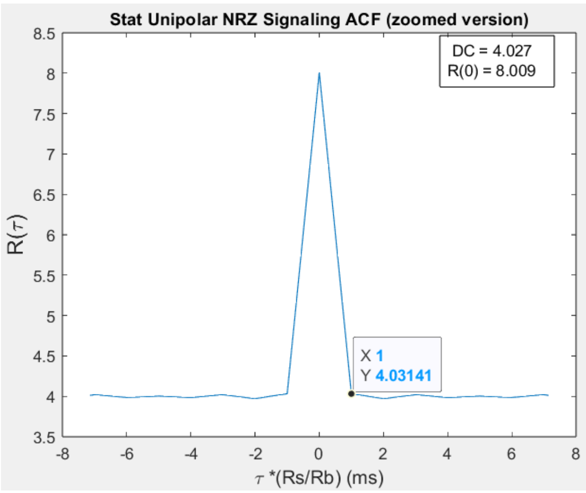
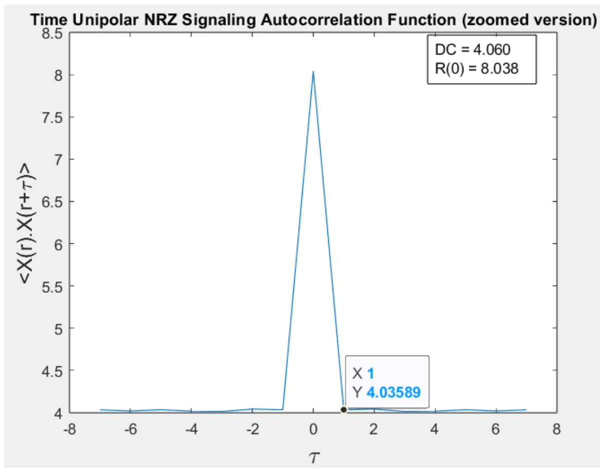
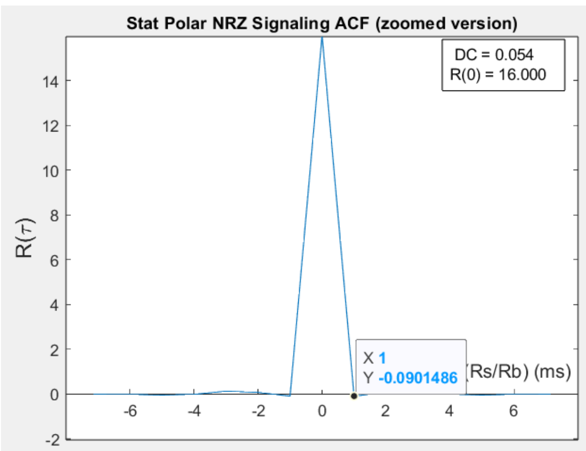
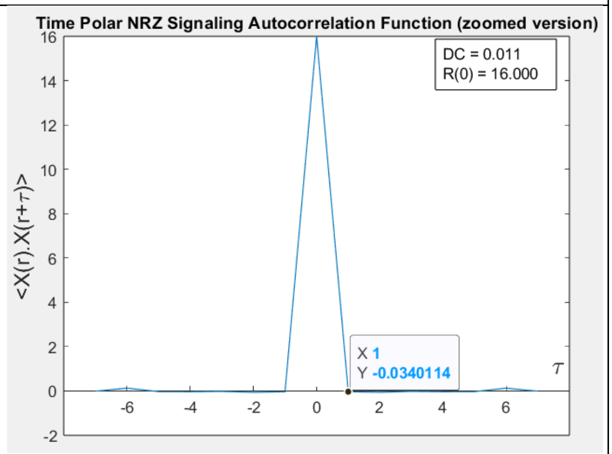
Snippet 7: Plotting Time Mean and Autocorrelation

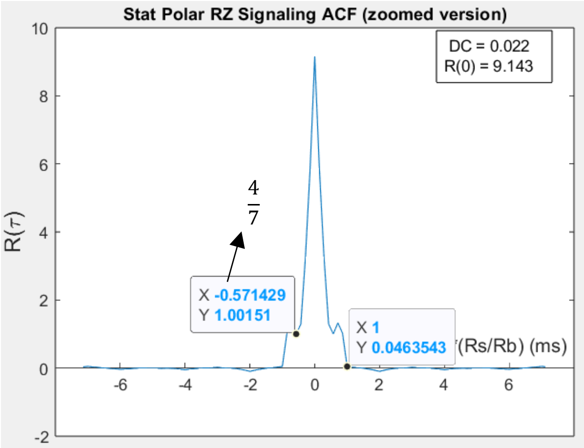
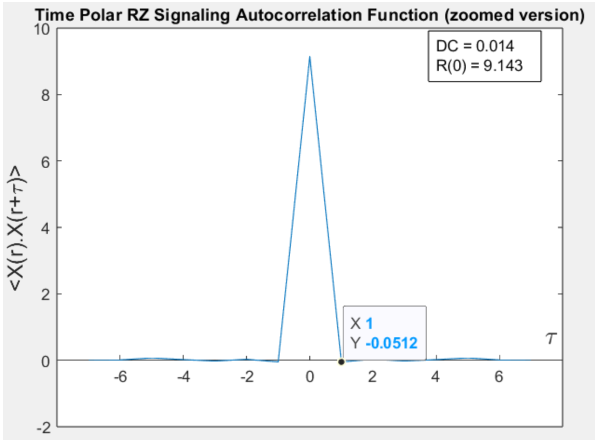
XVI

Is the RP Ergodic?

```
%% Checking the ergodicity using time(mean/ACF):
R_domain = 0:Ensemble_width - 1;
Signaling_time_mean = time_mean(Signaling_mapped, Ensemble_width);
Signaling_time_ACF_singlesided_positive = time_acf(Signaling_mapped, Ensemble_width);
Signaling_time_ACF_singlesided_negative = fliplr(Signaling_time_ACF_singlesided_positive);
Signaling_time_ACF
        = [Signaling_time_ACF_singlesided_negative(:,1:end - 1) Signaling_time_ACF_singlesided_positive];
r_length = length(Signaling_time_ACF);
r_domain = -(r_length - 1)/2:(r_length - 1)/2;
str_time_mean_Signaling = sprintf("Average = %.3f\nStd = %.3f",
                                  mean(Signaling_time_mean), std(Signaling_time_mean));
str_time_ACF_Signaling = sprintf("DC = %.3f\nR(0) = %.3f",
                                 mean(Signaling_time_ACF), Signaling_time_ACF(r_domain == 0));
```

Snippet 8: Plotting Time Mean and Autocorrelation

Line Code	Statistical ACF	Time ACF
Unipolar NRZ		
Polar NRZ		

<table>
<tr>
<td>Polar RZ</td>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>Result</td>
<td colspan="2">Since the value of the zero crossing in the Statistical ACF = Time ACF
Therefore, the Random Process is Ergodic.</td>
</tr>
</table>

XVII

Plotting PSD of the Ensemble

```
%% PSD
freq_domain = (-((tau_domain_length - 1)/2): ((tau_domain_length - 1)/2)).* (Rb/Rs/(tau_domain_length - 1));
% Normalized frequency scale (k) wrt to the sampling
switch cell2mat(Signaling_line_codes(indices))
  case 'Unipolar NRZ'
    Signaling_psd = abs(fftshift(fft(Signaling_stat_ACF)))/tau_domain_length;
    zero_crossing = psd_zero_cross(Signaling_psd, Rs * freq_domain, cell2mat(Signaling_line_codes(indicies)), Rs);
    str_psd_Signaling = sprintf('S(0) = %.3f \nZero crossing @ f = %.3f',
                                              Signaling_psd(freq_domain == 0), zero_crossing);
    figure();
    plot(freq_domain, Signaling_psd, 'LineWidth', 0.2);
    xlabel('Normalized frequency scale : k = (Rs/Rb) * F ');
    ylabel('|S(f)|');
    set(gca, 'YAxisLocation', 'origin');
    a5 = annotation('textbox', [0.55 0.52 0.3 0.3], 'String', str_psd_Signaling, 'FitBoxToText', 'on');
  case {'Polar NRZ', 'Polar RZ'}
      Signaling_psd = abs(fftshift(fft(Signaling_stat_ACF)))/final_frame_length;
      Mag_Norm = [1 Rb/Rs];
      zero_crossing = psd_zero_cross(Signaling_psd, Rs * freq_domain, cell2mat(Signaling_line_codes(indicies)), Rs);
    for p = 1: 2
      if(p == 1)
       psd_title = sprintf('%s Signaling PSD Un - normalized magnitude', cell2mat(Signaling_line_codes(indicies)));
       psd_ylabel = sprintf('|S(f)|');
      else
       psd_title = sprintf('%s Signaling PSD Normalized magnitude ', cell2mat(Signaling_line_codes(indicies)));
       psd_ylabel = sprintf('|S(f)| * (Rb/Rs)|');
      end
      figure();
      plot(freq_domain, Signaling_psd * Mag_Norm(p), 'LineWidth', 0.2);
      xlabel('Normalized frequency scale : k = (Rs/Rb) * F ');
      ylabel(psd_ylabel);
      set(gca, 'YAxisLocation', 'origin');
      title(psd_title);
      str_psd_Signaling = sprintf('S(0) = %.3f \nZero crossing @ f = %.3f',
                                       Signaling_psd(freq_domain == 0) * Mag_Norm(p), zero_crossing);
      a5 = annotation('textbox', [0.55 0.52 0.3 0.3], 'String', str_psd_Signaling, 'FitBoxToText', 'on');
```

```
            if(indicies == 2)
              a6 = annotation('ellipse', 'Position', [0.59 0.1 0.05 0.05], 'Color', 'r');
            else
              a6 = annotation('ellipse', 'Position', [0.69 0.1 0.05 0.05], 'Color', 'r');
            end
    end
end
%% Aiding Functions
function z = psd_zero_cross(x, f_domain, linecode, threshold) % get the first zero crossing
%The threshold sets the theoritical zero cross of the psd to scope it without
%being affected by the error due to the practical data set
error = threshold/12.5; % 8% acceptable error in the zero cross
switch linecode
case {'Unipolar NRZ', 'Polar NRZ'}
    l = f_domain(round(x) == 0 & abs(f_domain - threshold) <= error);
    z = l(abs((l - threshold)) == min(sort(abs(l - threshold))));
case 'Polar RZ'
    l = f_domain(round(x) == 0 & abs(f_domain - 2 * threshold) <= error);
    z = l(abs((l - 2 * threshold)) == min(sort(abs(l - 2 * threshold))));
end
end
```

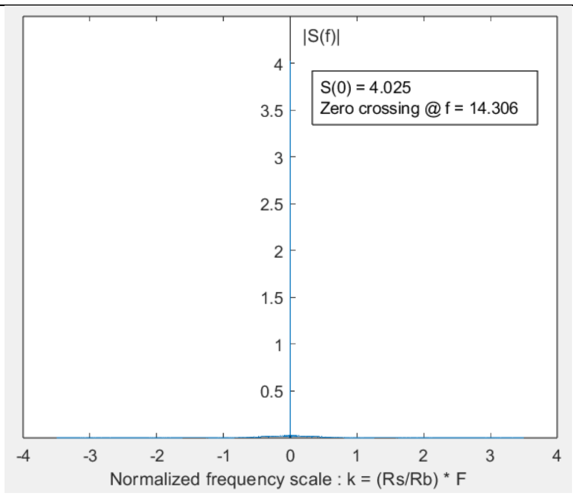


Figure 19: PSD for Unipolar NRZ Signaling Normalized Frequency

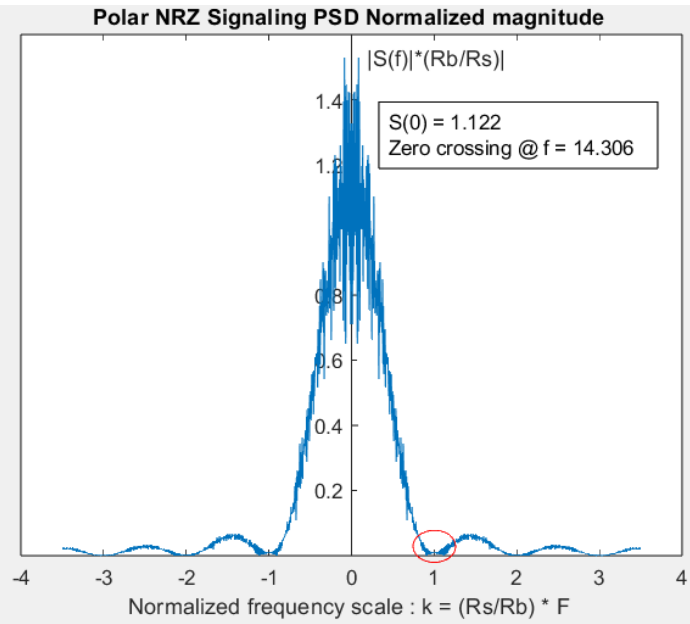


Figure 20: Normalized PSD Magnitude for Polar NRZ Signaling Normalized Frequency

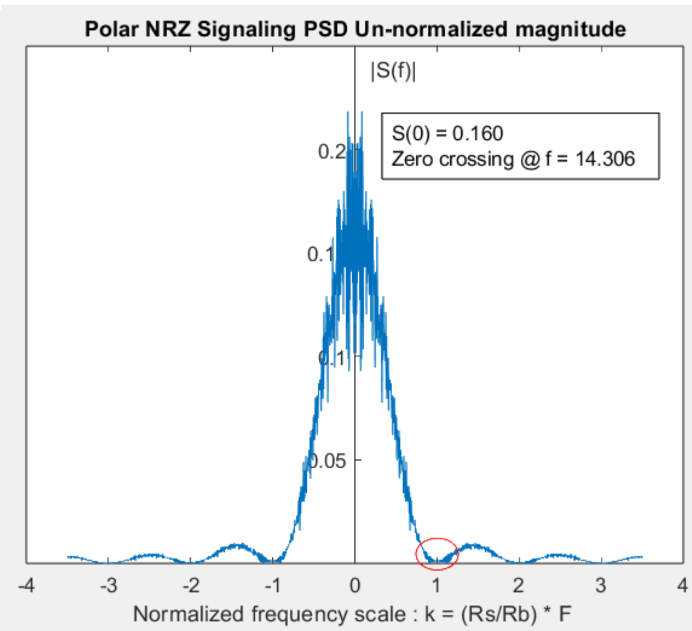


Figure 21: Un - Normalized PSD Magnitude for Polar NRZ Signaling Normalized Frequency

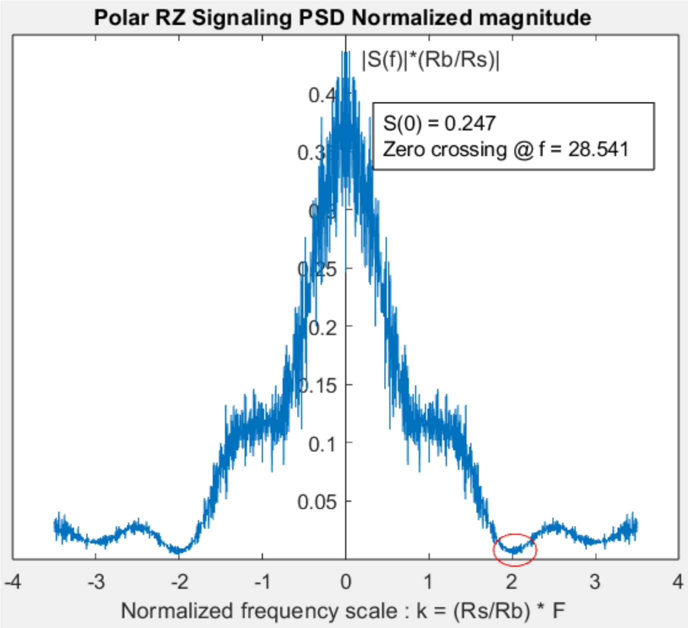


Figure 22: Normalized PSD Magnitude for Polar RZ Signaling Normalized Frequency

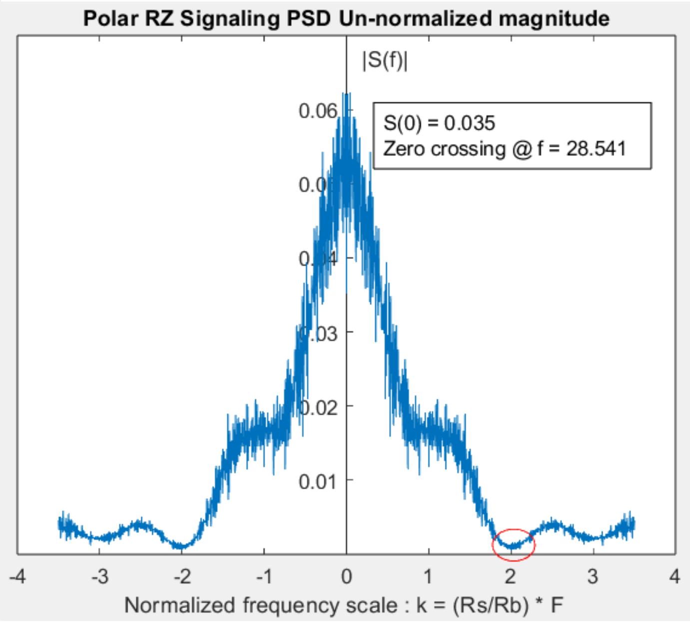


Figure 23: Un - Normalized PSD Magnitude for Polar RZ Signaling Normalized Frequency

XVIII BW of Tx Signal

P.O.C		Unipolar NRZ	Polar NRZ	Polar RZ
BW (HZ)	**Theoretical**	$R_s = 14.286$	$R_s = 14.286$	$2R_s = 28.57$
	Practical	14.306	14.306	28.541
Comments: Polar RZ gives better Synchronization but for BW Efficiency = 50 %				
PSD S (0)	**Theoretical**	$\frac{A^2}{4}(1 + T_b) = 4.04$	$A^2 \times T_b = 0.16$	$A^2 \times \left(\frac{4}{7}\right)^2 \times T_b = 0.05$
	Practical	4.025	0.160	0.035

Table 3: Bandwidth of the transmitted Signal

Comment:

- PSD for Polar NRZ / RZ is Normalized for the fs to give a better intuition for the duty cycle used for RZ line coding.
- Frequency axis is also normalized for fs to give a better intuition for BW efficiency.

XIX

Full MATLAB Code

```
clear all
close all
if(exist('a6','var'))
clf(figure(1),figure(2));
delete([a1 a2 a3 a4  a5 a6]);
end
%% General Notes
frame_length = 100; %Required no. of tranmitted bits
Ensemble_width = 500; %Number of realizations
DAC_samples = 7; % Dac samples = Dac sampling freq / Bit rate = (70 ms)/(10 ms)
cond = 1;
while(cond)
A = input(" Enter the amplitude of the line code : "); % Rails
if(A > 0 && A < 10)
   cond = 0;
   else
      fprintf(" Wrong Input A in ]0,10] , please try again .....\n");
   end
end
final_frame_length = frame_length * DAC_samples; %The frame length after the DAC
Rs = (10^3)/70; %(1/Ts) = 1/70ms --> Symbol rate
Rb = (10^3)/10; %(1/Tb = n/Ts) = 1/10ms --> bit rate
%% Generation of the general random Delay & Tx stream (not mapped yet)
% Add extra bit is used to introduce the delay given by the last transmitted frame
tx = randi([0 1], Ensemble_width, frame_length + 1);
tx_DAC_out = repelem(tx, 1, DAC_samples);
% Delay the realizations
delay_period = randi([0 DAC_samples - 1],1, Ensemble_width);
tx_delayed = zeros(Ensemble_width, size(tx, 2) * DAC_samples);
for i = 1: Ensemble_width
   tx_delayed(i, : ) = circular_shift(tx_DAC_out(i, : ), delay_period(i));
end
% pickup the shifted frame
for i = 1: Ensemble_width
   tx_final = tx_delayed(: , delay_period(i) + 1: delay_period(i) + final_frame_length);
end
%% Choose the required Signaling
Signaling_line_codes = {'Unipolar NRZ', 'Polar NRZ', 'Polar RZ'};
[indicies, check] = listdlg('ListString', Signaling_line_codes, 'promptstring',
          'Select the line code to continue (Unipolar Signaling by default)', 'SelectionMode', 'single');
if (~check)
   indicies = 1 ;
end
switch cell2mat(Signaling_line_codes(indicies))
   case 'Unipolar NRZ'
     Signaling_mapped = tx_final * A;  % Unipolar NRZ Signaling Line Code
   case 'Polar NRZ'
     Signaling_mapped = (2 * tx_final - 1) * A;  % polar NRZ Signaling Line Code
```

```
  case 'Polar RZ'
    Signaling_mapped
              = (2 * tx_final - 1) * A; % polar RZ (Take NRZ and replace some bits with Zeros)
    for k = 5: 7: final_frame_length - 2
      for l = k: k + 2
        Signaling_mapped(: , l) = 0;
      end
    end
end
%% Checking the sationarity using statistical (Mean/ACF)
Signaling_stat_mean = Stat_mean(Signaling_mapped, final_frame_length);
Signaling_stat_ACF_single_sided_positive = stat_acf(Signaling_mapped, final_frame_length);
Signaling_stat_ACF_single_sided_negative = fliplr(Signaling_stat_ACF_single_sided_positive);
Signaling_stat_ACF = [Signaling_stat_ACF_single_sided_negative
                               (: ,1: end - 1) Signaling_stat_ACF_single_sided_positive];
tau_domain_length = length(Signaling_stat_ACF);
tau_domain = (-((tau_domain_length - 1)/2): ((tau_domain_length - 1)/2)).* (Rs/Rb);
time_domain = 0: final_frame_length - 1;
%{
  1 - To see if the mean is constant , we can check that :
   * Unipolar :
    Average of the mean across the time instants ~ A/2 & Std ~ 0
   * Polar NRZ:
    Average of the mean across the time instants ~ 0 & Std ~ 0
   * Polar RZ:
    Average of the mean across the time instants ~ 0 & Std ~ 0
  2 - We can check the statistic ACF by :
   * Unipolar :
     ACF @(tau = 0) ~ A^2/2 & DC ~ A^2/4
   * Polar NRZ:
     ACF @(tau = 0) ~ A^2 & DC ~ 0
   * Polar RZ :
     ACF @(tau = 0) ~ A^2 * 4/7 & DC ~ 0 (4/7 = Duty Cycle)
%}
str_stat_mean_Signaling = sprintf(" Average = %.3f \nStd = %.3f",
                               mean(Signaling_stat_mean), std(Signaling_stat_mean));
str_stat_ACF_Signaling = sprintf(" DC = %.3f \nR(0) = %.3f",
                       mean(Signaling_stat_ACF), Signaling_stat_ACF(tau_domain == 0));
%% plot the statistic mean & ACF
figure(1);
plot(time_domain, Signaling_stat_mean);
xlabel('Time (msec) ');
ylabel('E\{X(t)\}')
set(gca, 'XAxisLocation', 'origin');
title(sprintf("Stat %s Signaling Mean", cell2mat(Signaling_line_codes(indicies))));
a1 = annotation('textbox', [0.4 0.62 0.3 0.3], 'String', str_stat_mean_Signaling, 'FitBoxToText', 'on');
figure(2);
plot(tau_domain, Signaling_stat_ACF);
xlabel('\tau * (Rs/Rb) (ms)', 'FontSize', 12);
ylabel('R(\tau)', 'FontSize', 14);
set(gca, 'XAxisLocation', 'origin');
```

```
title(sprintf("Stat %s Signaling ACF", cell2mat(Signaling_line_codes(indicies))));
a2 = annotation('textbox', [0.7 0.62 0.3 0.3], 'String', str_stat_ACF_Signaling, 'FitBoxToText', 'on');
figure();
zoom = final_frame_length + [-50 50];
plot(tau_domain(zoom(1): zoom(2)), Signaling_stat_ACF(zoom(1): zoom(2)));
a2 = annotation('textbox', [0.7 0.62 0.3 0.3], 'String', str_stat_ACF_Signaling, 'FitBoxToText', 'on');
title(sprintf("Stat %s Signaling ACF (zoomed version)", cell2mat(Signaling_line_codes(indicies))));
xlabel('\tau * (Rs/Rb) (ms)', 'FontSize', 12);
ylabel('R(\tau)', 'FontSize', 14);
set(gca, 'XAxisLocation', 'origin');
%% Checking the ergodicity using time(mean/ACF):
R_domain = 0: Ensemble_width - 1;
Signaling_time_mean = time_mean(Signaling_mapped, Ensemble_width);
Signaling_time_ACF_singlesided_positive = time_acf(Signaling_mapped, Ensemble_width);
Signaling_time_ACF_singlesided_negative = fliplr(Signaling_time_ACF_singlesided_positive);
Signaling_time_ACF = [Signaling_time_ACF_singlesided_negative
                          (:,1: end - 1) Signaling_time_ACF_singlesided_positive];
r_length = length(Signaling_time_ACF);
r_domain = -(r_length - 1)/2: (r_length - 1)/2;
str_time_mean_Signaling = sprintf("Average = %.3f \nStd = %.3f",
                          mean(Signaling_time_mean), std(Signaling_time_mean));
str_time_ACF_Signaling = sprintf("DC = %.3f \nR(0) = %.3f",
                          mean(Signaling_time_ACF), Signaling_time_ACF(r_domain == 0));
%% plot the time mean & ACF
figure();
plot(R_domain, Signaling_time_mean, 'LineWidth', 0.25);
xlabel('Realization');
ylabel(' < X(t) > ')
set(gca, 'XAxisLocation', 'origin');
title(sprintf("Time %s Signaling Mean", cell2mat(Signaling_line_codes(indicies))));
a3 = annotation('textbox', [0.4 0.62 0.3 0.3], 'String', str_time_mean_Signaling, 'FitBoxToText', 'on');
figure();
plot(r_domain, Signaling_time_ACF);
xlabel('\tau', 'FontSize', 16);
ylabel(' < X(r).X(r +\tau) > ', 'FontSize', 14);
set(gca, 'XAxisLocation', 'origin');
title(sprintf("Time %s Signaling Autocorrelation Function", cell2mat(Signaling_line_codes(indicies)))
a4 = annotation('textbox', [0.7 0.62 0.3 0.3], 'String', str_time_ACF_Signaling, 'FitBoxToText', 'on');
figure();
zoom = round(Ensemble_width + ([-50 50] * (Rs/Rb)));
plot(r_domain(1, zoom(1): zoom(2)), Signaling_time_ACF(1, zoom(1): zoom(2)));
xlabel('\tau', 'FontSize', 16);
ylabel(' < X(r).X(r +\tau) > ', 'FontSize', 14);
set(gca, 'XAxisLocation', 'origin');
a4 = annotation('textbox', [0.7 0.62 0.3 0.3], 'String', str_time_ACF_Signaling, 'FitBoxToText', 'on');
title(sprintf("Time %s Signaling Autocorrelation Function (zoomed version)",
                          cell2mat(Signaling_line_codes(indicies))));
%% PSD
freq_domain = (-((tau_domain_length - 1)/2): ((tau_domain_length - 1)/2)).* (Rb/Rs/(tau_domain_length - 1));
```

```
% Normalized frequency scale (k) wrt to the sampling
switch cell2mat(Signaling_line_codes(indicies))
  case 'Unipolar NRZ'
    Signaling_psd = abs(fftshift(fft(Signaling_stat_ACF)))/tau_domain_length;
    zero_crossing
              = psd_zero_cross(Signaling_psd, Rs
              * freq_domain, cell2mat(Signaling_line_codes(indicies)), Rs);
    str_psd_Signaling = sprintf('S(0) = %.3f \nZero crossing @ f
              = %.3f', Signaling_psd(freq_domain == 0), zero_crossing);
    figure();
    plot(freq_domain, Signaling_psd, 'LineWidth', 0.2);
    xlabel('Normalized frequency scale : k = (Rs/Rb) * F ');
    ylabel('|S(f)|');
    set(gca, 'YAxisLocation', 'origin');
    a5 = annotation('textbox', [0.55 0.52 0.3 0.3], 'String', str_psd_Signaling, 'FitBoxToText', 'on');
  case {'Polar NRZ', 'Polar RZ'}
      Signaling_psd = abs(fftshift(fft(Signaling_stat_ACF)))/final_frame_length;
      Mag_Norm = [1 Rb/Rs];
      zero_crossing = psd_zero_cross(Signaling_psd,
                             Rs * freq_domain, cell2mat(Signaling_line_codes(indicies)), Rs);
    for p = 1:2
      if(p == 1)
        psd_title = sprintf('%s Signaling PSD Un - normalized magnitude',
                                  cell2mat(Signaling_line_codes(indicies)));
        psd_ylabel = sprintf('|S(f)|');
      else
       psd_title = sprintf('%s Signaling PSD Normalized magnitude ',
                              cell2mat(Signaling_line_codes(indicies)));
       psd_ylabel = sprintf('|S(f)| * (Rb/Rs)|');
      end
      figure();
      plot(freq_domain, Signaling_psd * Mag_Norm(p), 'LineWidth', 0.2);
      xlabel('Normalized frequency scale : k = (Rs/Rb) * F ');
      ylabel(psd_ylabel);
      set(gca, 'YAxisLocation', 'origin');
      title(psd_title);
      str_psd_Signaling = sprintf('S(0) = %.3f \nZero crossing @ f = %.3f',
                           Signaling_psd(freq_domain == 0) * Mag_Norm(p), zero_crossing);
     a5 = annotation('textbox', [0.55 0.52 0.3 0.3], 'String', str_psd_Signaling, 'FitBoxToText', 'on');
        if(indicies == 2)
          a6 = annotation('ellipse', 'Position', [0.59 0.1 0.05 0.05], 'Color', 'r');
        else
          a6 = annotation('ellipse', 'Position', [0.69 0.1 0.05 0.05], 'Color', 'r');
        end
   end
end
%% Aiding Functions
function z = circular_shift(x, shift_val) %circular shift the realizations with the delay
len = length(x);
z = zeros(1, len);
  if(shift_val~ = 0)
```

```
      start  =  len − (shift_val − 1);
      i  =  start;
      j = 1;
      while (j <= len)
        z(j) = x(i);
        j = j + 1;
        i = i + 1;
        if(i > len)
          i = 1;
        end
      end
    else
      z  =  x;
    end

end
function z  =  average(x)
len  =  length(x);
sum  =  0;
for i =  1: len
    sum  =  sum  +  x(i);
end
z  =  sum/len;
end
function [y] =  Stat_mean(x, frame_length)
y  =  zeros(1, frame_length);
for t = 1: frame_length
y(t) =  average(x(: , t));
end
end
function [z]  =  stat_acf(x, frame_length)
%The code exercises the  + ve &  − ve sides of the ACF so it implicitly checks if it is even or not
y = zeros(frame_length, frame_length);
z  =  zeros(1, frame_length);
tau_hash  =  zeros(1, frame_length);
for t1  =  1 :  frame_length
  tau = 0;
    while(1)
      t2 = t1 + tau;
      y(t1, abs(tau) + 1)  =  average(x(: , t1).∗ x(: , t2));
      tau_hash(abs(tau) + 1)  =  tau_hash(abs(tau) + 1)  + 1;
      if(t1 <= (frame_length/2))
        tau = tau + 1;
      else
        tau  =  tau − 1;
      end
      if((t1 + tau) > frame_length || (t1 + tau)  < 1)
        break;
      end
    end
end
```

```
      sum_instants = sum(y);
      for i = 1: frame_length
      z(i) = sum_instants(i)/tau_hash(i);
      end
end
function [y] = time_mean(x, realization_count)
y = zeros(realization_count, 1);
for r = 1: realization_count
y(r) = average(x(r, : ));
end
end
function [z] = time_acf(x, realization_count)
y = zeros(realization_count, realization_count);
z = zeros(1, realization_count);
tau_hash = zeros(1, realization_count);
for n1 = 1 : realization_count
 tau = 0;
  while(1)
      n2 = n1 + tau;
      y(n1, abs(tau) + 1) = average(x(n1, : ).* x(n2, : ));
      tau_hash(abs(tau) + 1) = tau_hash(abs(tau) + 1) + 1;
      if(n1 <= (realization_count/2))
        tau = tau + 1;
      else
        tau = tau - 1;
      end

      if((n1 + tau) > realization_count || (n1 + tau) < 1)
        break;
      end

  end
end
  sum_instants = sum(y);
  for i = 1: realization_count
      z(i) = sum_instants(i)/tau_hash(i);
  end
end
function z = psd_zero_cross(x, f_domain, linecode, threshold) % get the first zero crossing
%The threshold sets the theoritical zero cross of the psd to scope it without
%being affected by the error due to the practical data set
error = threshold/12.5; % 8% acceptable error in the zero cross
switch linecode
case {'Unipolar NRZ', 'Polar NRZ'}
     l = f_domain(round(x) == 0 & abs(f_domain - threshold) <= error);
     z = l(abs((l - threshold)) == min(sort(abs(l - threshold))));
case 'Polar RZ'
     l = f_domain(round(x) == 0 & abs(f_domain - 2 * threshold) <= error);
     z = l(abs((l - 2 * threshold)) == min(sort(abs(l - 2 * threshold))));
end     end
```

The END Thank You