**Electronics and Electrical Communications Engineering  
Department**

**Faculty of Engineering**

**Cairo University**

**Implementation and Comparative Analysis of Huffman and Fano Source Coding Algorithms**

**ELC4020 “Advanced Communication Systems “**

**4th Year**

**1st Semester - Academic Year 2025/2026**

**Prepared by:**

|  |  |  |
| --- | --- | --- |
| **NAME** | **SECTION** | **ID** |
| **Mohamed Ahmed Abd El Hakam** | **3** | **9220647** |
| **Yousef Khaled Omar Mahmoud** | **4** | **9220984** |
| **Ahmed Wagdy Mohy Ibrahim** | **dsds** | **9220120** |
|  | **dsda** | **da** |

**Submission Date:** 2 November 2025

**Instructor: Eng. Mohamed Khaled**

**Dr. Mohammed Nafie & Dr. Mohamed Khairy**

# Contents

[A. **Contents** 2](#_Toc195652116)

[**B.** **Table of Figures** 3](#_Toc195652117)

[**C.** **Role of Each Member** 4](#_Toc195652118)

[**D.** **Project Description** 5](#_Toc195652119)

[**E.** **Introduction** 5](#_Toc195652120)

[**F.** **Control Flags** 5](#_Toc195652121)

[**G.** **Generation of Data** 7](#_Toc195652122)

[**H.** **polar NRZ ensemble creation** **Error! Bookmark not defined.**](#_Toc195652123)

[**I.** **Uni polar NRZ ensemble creation** **Error! Bookmark not defined.**](#_Toc195652124)

[**J.** **polarRZ ensemble creation** **Error! Bookmark not defined.**](#_Toc195652125)

[**K.** **Random initial time shift** **Error! Bookmark not defined.**](#_Toc195652126)

[**L.** **Getting cell arrays ready to calculate the statistical mean and autocorrelation:** **Error! Bookmark not defined.**](#_Toc195652127)

[**M.** **Questions** **Error! Bookmark not defined.**](#_Toc195652128)

[**1.** **Statistical Mean** **Error! Bookmark not defined.**](#_Toc195652129)

[**1.1.** **Hand Analysis** **Error! Bookmark not defined.**](#_Toc195652130)

[**1.2.**  **Code Snippet** **Error! Bookmark not defined.**](#_Toc195652131)

[**1.3.** **Plotting the Statistical Mean:** **Error! Bookmark not defined.**](#_Toc195652132)

[**2.** **Statistical Autocorrelation** **Error! Bookmark not defined.**](#_Toc195652133)

[**2.1.** **Hand Analysis** **Error! Bookmark not defined.**](#_Toc195652134)

[**2.2.** **Code Snippet** **Error! Bookmark not defined.**](#_Toc195652135)

[**2.3.** **Plotting the statistical autocorrelation** **Error! Bookmark not defined.**](#_Toc195652136)

[**3.** **Is the Process Stationary** 23](#_Toc195652137)

[**4.** **The time mean and autocorrelation function for one waveform** 24](#_Toc195652138)

[**4.1.** **Time Mean** 24](#_Toc195652139)

[**4.3.** **Time Auto Correlation** 27](#_Toc195652140)

[**4.4.** **Time Auto Correlation for one wave form:** 28](#_Toc195652141)

[**5.** **Is The Random Process Ergodic?** 29](#_Toc195652142)

[**6.** **the PSD & Bandwidth of the Ensemble** 31](#_Toc195652143)

[**6.1.** **PSD using fft:** 31](#_Toc195652144)

[**6.2.** **Theoritical PSD:** 33](#_Toc195652145)

[**N.** **References:** 34](#_Toc195652146)

[**O.** **Appendix** 34](#_Toc195652147)

# Table of Figures

[Figure 1 Rx and Tx path 5](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652148)

[Figure 2 ADC Binary Output **Error! Bookmark not defined.**](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652149)

[Figure 3 PolarNRZ Realizations **Error! Bookmark not defined.**](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652150)

[Figure 4 Uni Polar Realizations **Error! Bookmark not defined.**](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652151)

[Figure 5 PolarRZ Realization **Error! Bookmark not defined.**](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652152)

[Figure 6 Realization Shifted **Error! Bookmark not defined.**](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652153)

[Figure 7 Plot of Statistical Mean **Error! Bookmark not defined.**](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652154)

[Figure 8 Statistical Auto Correction plot **Error! Bookmark not defined.**](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652155)

[Figure 9 Statistical Auto Correction plot zoomed **Error! Bookmark not defined.**](#_Toc195652156)

[Figure 10 autocorrelation at two different times 23](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652157)

[Figure 11 Time Mean for Uni Polar 24](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652158)

[Figure 12 Time Mean for Polar NRZ 25](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652159)

[Figure 13 Time Mean for Polar RZ 25](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652160)

[Figure 14 Time Mean Vs Realization 26](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652161)

[Figure 15 Time Auto Correction plot zoomed 28](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652162)

[Figure 16 Time Auto Correction plot 28](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652163)

[Figure 17 Time Mean vs Statistical 29](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652164)

[Figure 18 Time Auto Correlation Vs Statistical 29](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652165)

[Figure 19 PSD plot of the Ensemble 32](file:///D:\project\Digital%20Communication%20Radio\Digital_Communication_Radio\project%201\P1-T25.docx#_Toc195652166)

# Role of Each Member

|  |  |
| --- | --- |
| Role | Name |
| code the Huffman source coding | Youssef Khaled |
|  | Ahmed Wagdy |
| Report and Hand Analysis | Mohamed Ahmed |
| Report and Hand Analysis | Abdelrahman Eissa |

# Project Description

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

هنا ممكن تشرحله انه تشرحله انه  
source code implementqation using matlab

# Introduction

Software radio is a revolutionary approach that brings the programming code directly to the antenna, minimizing reliance on traditional radio hardware as shown in figure 1.

By doing so, it transforms challenges associated with radio hardware into software- related issues. Unlike conventional radios, where signal processing primarily relies on analog circuitry or a combination of analog and digital chips, software radio operates by having software dictate both the transmitted and received waveforms.

هنا ممكن تشرحله انه تشرحله انه  
what’s source coding and why it’s important

This paradigm shift allows for greater flexibility and adaptability in radio systems, as they can be easily reconfigured and optimized through software updates, rather than hardware modifications.

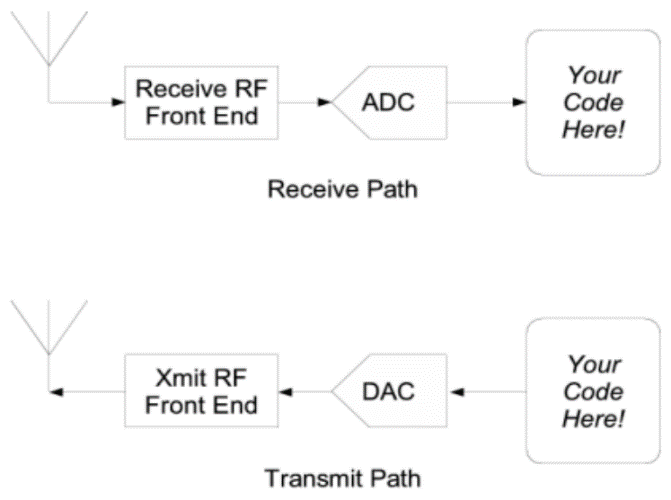


Figure 1 Rx and Tx path

# Control Flags

مفيش المرة ده ممكن تمسحها لو عاوز

|  |  |  |
| --- | --- | --- |
| Flag | Value | Description |
| A | 4 | Amplitude of line code |
| N\_realizations | 500 | Number of waveforms (ensemble size) |
| num\_bits | 101 | Bits per waveform and one extra bit for shifting |
| bit\_duration | 70e-3 | Duration of each bit |
| dac\_interval | 10e-3 | DAC update interval |

# Input Data Symbols

Figure 2 Input Symbols

As seen in figure 2, this is the input given

كمل هنا عشان معيش أفكار

# Huffman Source Coding

حط هنا شوية معلومات عنه زي مثلا انه ممكن يوصل ل 100% لو كان عددهم ما لا نهاية

## Algorithm Overview[2]

Figure 3 Huffman Algorithm

اشرحه عادي زي ما هو شرح في السيكشن

## Hand Analysis

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Symbol | P0 | C0 | P1 | C1 | P2 | C2 | P3 | C3 | P4 | C4 | P5 | C5 |
| A | 0.35 | 00 | 0.35 | 00 | 0.35 | 00 | 0.35 | 00 | 0.35 | 1 | 0.65 | 0 |
| B | 0.3 | 01 | 0.3 | 01 | 0.3 | 01 | 0.3 | 01 | 0.35 | 00 | 0.35 | 1 |
| C | 0.2 | 10 | 0.2 | 10 | 0.2 | 10 | 0.2 | 10 | 0.3 | 01 |  |  |
| D | 0.1 | 110 | 0.1 | 110 | 0.1 | 110 | 0.15 | 11 |  |  |  |  |
| E | 0.04 | 1110 | 0.04 | 1110 | 0.05 | 111 |  |  |  |  |  |  |
| F | 0.005 | 11110 | 0.01 | 1111 |  |  |  |  |  |  |  |  |
| G | 0.005 | 11111 |  |  |  |  |  |  |  |  |  |  |

So here’s the output that we are aiming for.

With the Kraft’s Sum,entropy, average length and efficiency:

So by calculating them we can find that:

And

So the overall efficiency is

احسب هنا ال  
kraft   
,اقله  
stratify

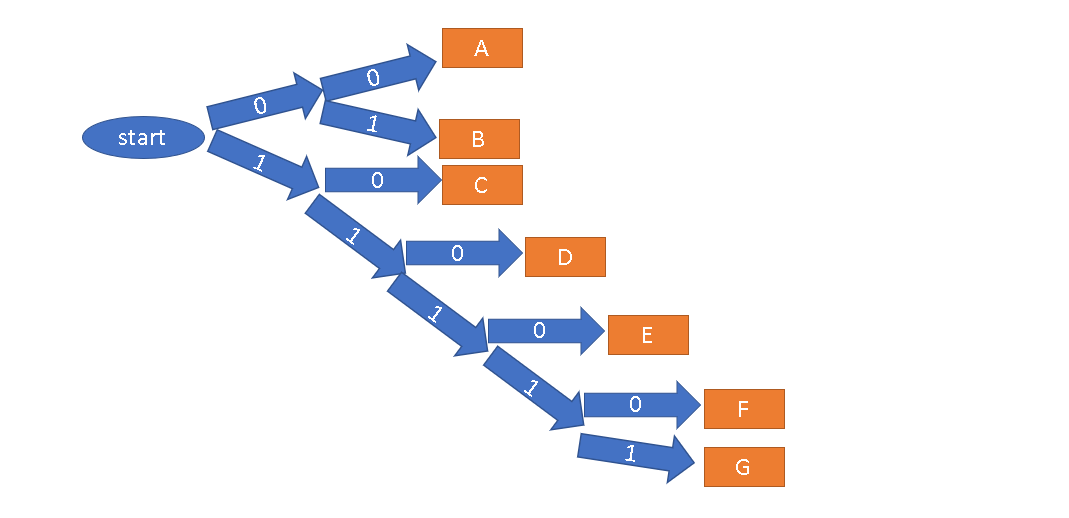


Figure 4 Huffman Output Kraft's Tree

As seen the kraft tree doesn’t have a code in the internal not so it’s Instantinous code

## MatLab Implementation[3]

### Calculations Functions

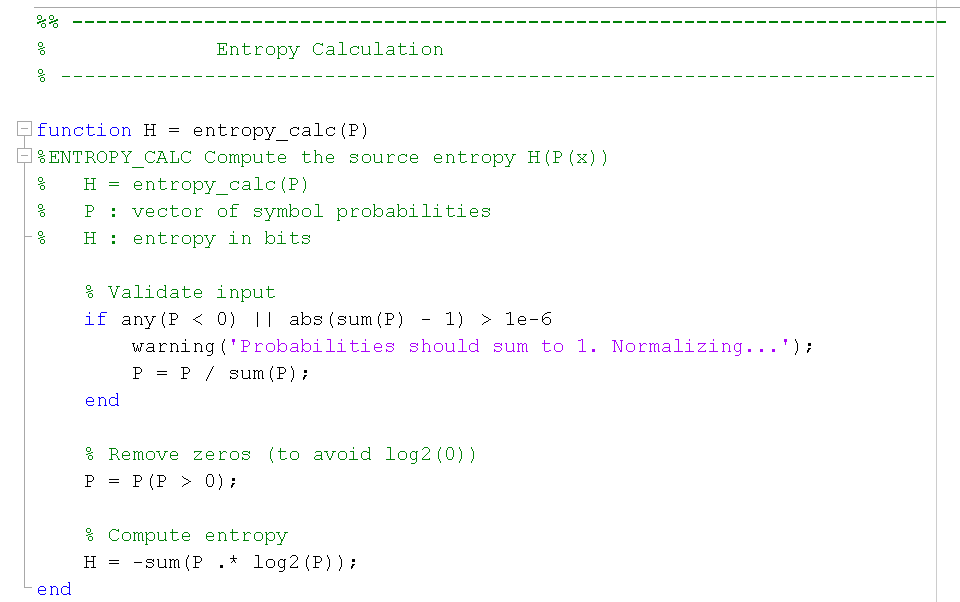
First we got the calculations Functions:

Figure 5 Entropy Calculation

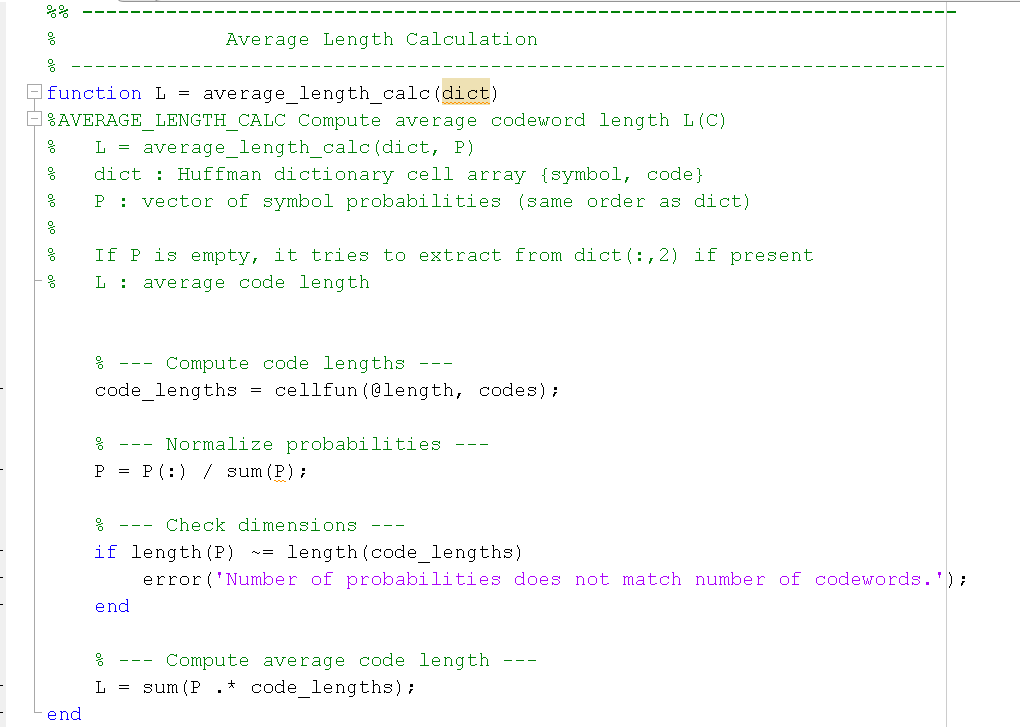
And

Figure 6 Average length Calculation

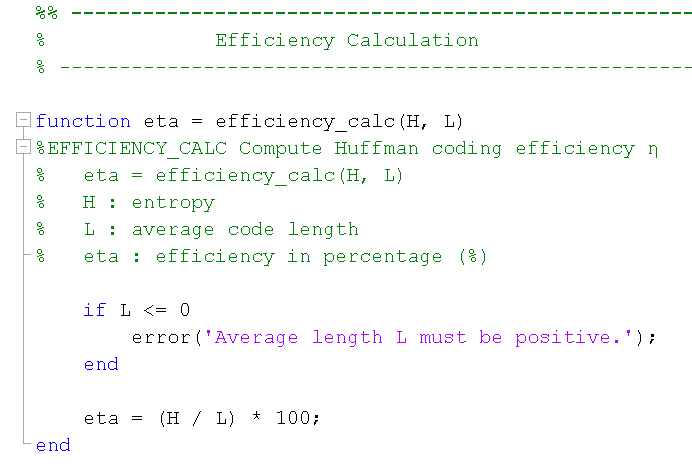
And

Figure 7 Efficiency Calculation

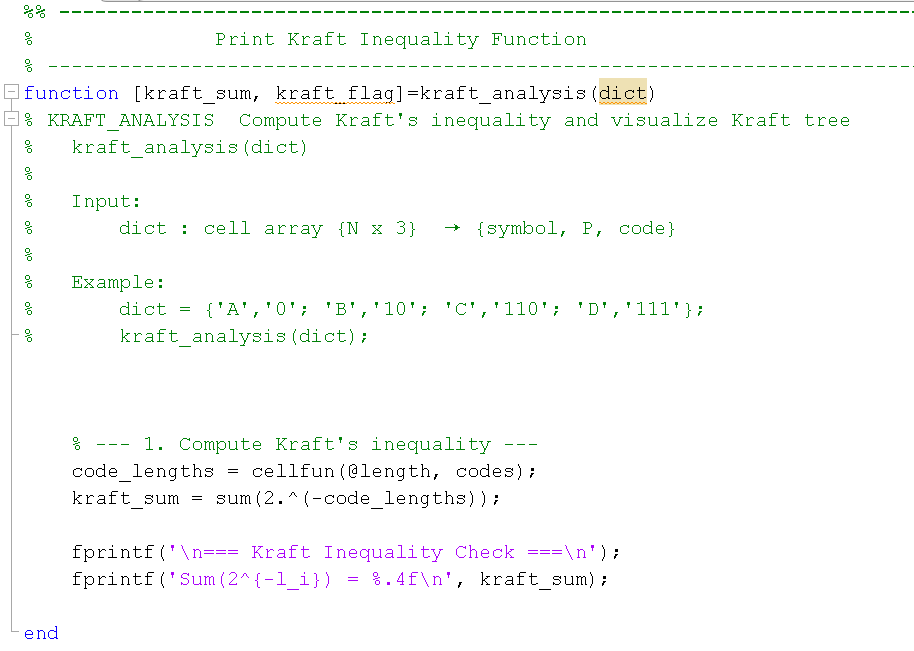
And

Figure 8 Kraft's Inequality

### Getting input data

Figure 9 Input Validation

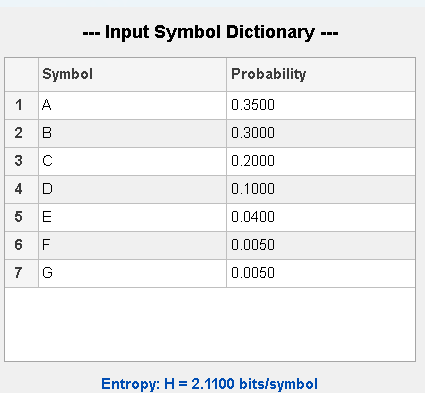
So the output is:

Figure 10 Input Symbols

### Huffman Function

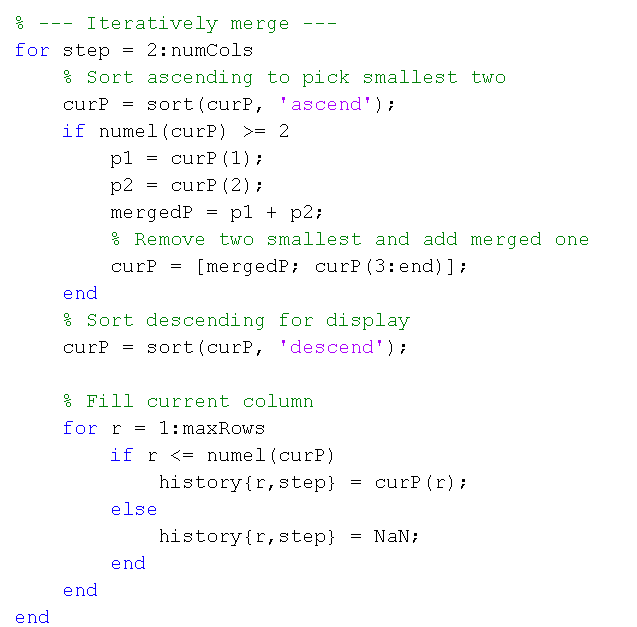
First we merged the last 2 probabilities to have in last just 2 probabilities

Figure 11 Probability Merge

So we got:

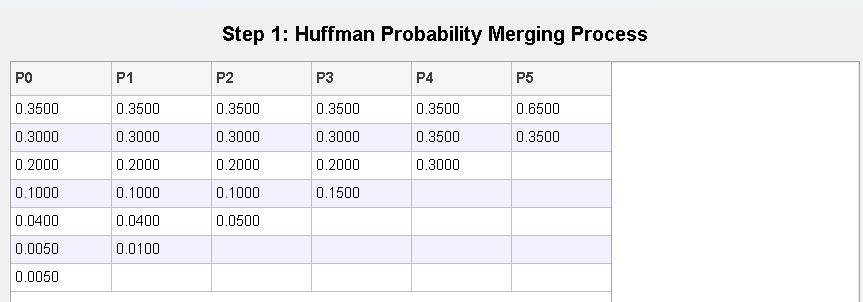


Figure 12 Merged Probabilities

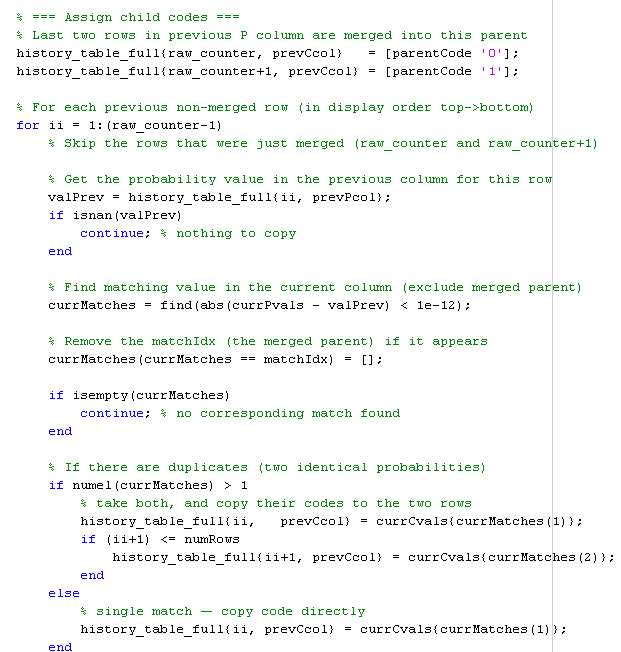
s Then we assign the codes from last column to first

Figure 13 Code Assignment

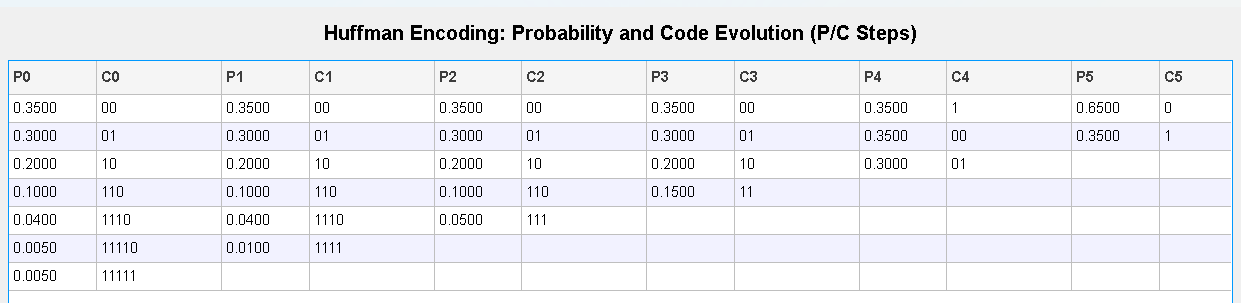
So the output is:

Figure 14 Huffman Steps

### Theoretical Vs Practical

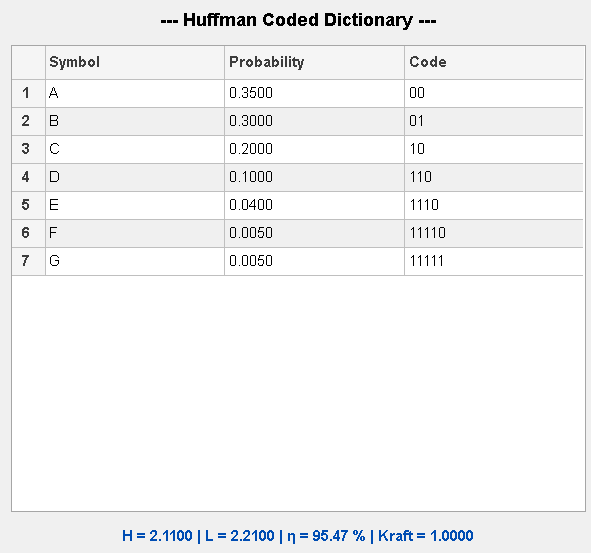
 The results are:

Figure 15 Huffman Output

As theoretical

# Fano Source Coding

حط هنا شوية معلومات عنه

## Algorithm Overview[2]

Figure 16 Fano Algorithm

اشرحه عادي زي ما هو شرح في السيكشن

## Hand Analysis

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Symbol | P0 | C0 | P1 | C1 | P2 | C2 | P3 | C3 | P4 | C4 | Final Code |
| B | 0.3 | 0 | 0.3 | 01 | 0.3 | 01 | 0.3 | 01 | 0.3 | 01 | 11 |
| C | 0.2 | 0.2 | 00 | 0.2 | 00 | 0.2 | 00 | 0.2 | 00 | 01 |
| A | 0.35 | 1 | 0.35 | 11 | 0.35 | 11 | 0.35 | 11 | 0.35 | 11 | 00 |
| D | 0.1 | 0.1 | 10 | 0.04 | 101 | 0.04 | 101 | 0.04 | 101 | 1000 |
| E | 0.04 | 0.04 | 0.01 | 100 | 0.01 | 1000 | 0.01 | 1000 | 101 |
| F | 0.005 | 0.005 | 0.005 | 0.005 | 1001 | 0.005 | 10010 | 10010 |
| G | 0.005 | 0.005 | 0.005 | 0.005 | 0.005 | 10011 | 10011 |

So here’s the output that we are aiming for.

With the Kraft’s Sum,entropy, average length and efficiency:

احسب هنا كل الحاجات اللي انا كنت حاسبها عند هافمان زي:  
length..  
kraft   
,اقله  
stratify

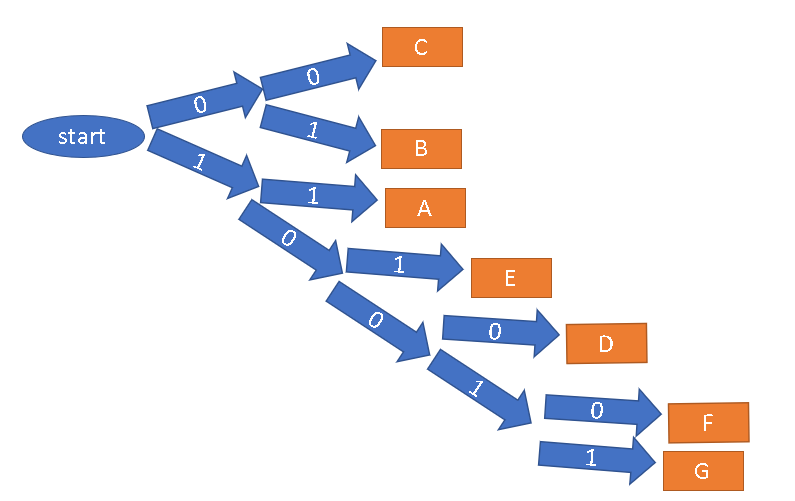
Kraft’s tree:

Figure 17 Fano Output Kraft's Tree

### Fano Function

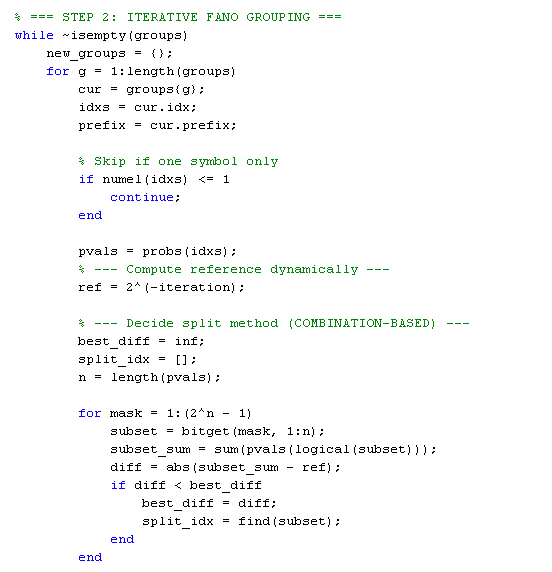
First we splitted the probabilities into groups

Figure 18 Group Splitting

Then we assign the codes to each group

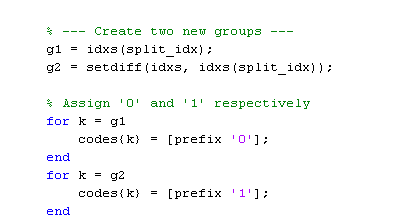


Figure 19 Group Code Assignment

So each group got a new code

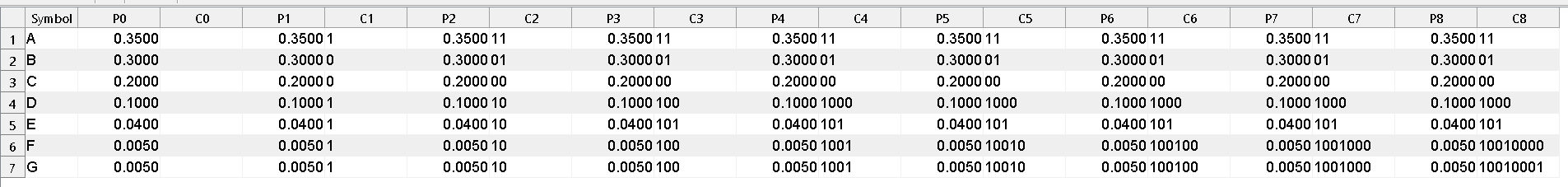
So the output steps is

Figure 20 Fano Steps

### Theoretical Vs Practical

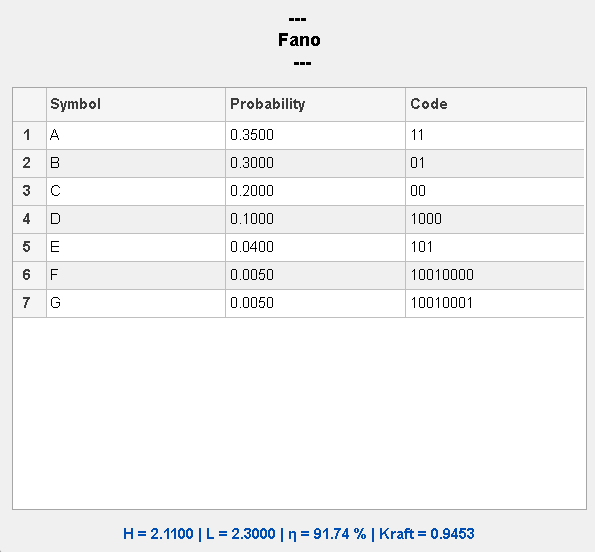
 The results are:

Figure 21 Fano Output

As theoretical

## Is the Process Stationary

Figure 22 autocorrelation at two different times

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

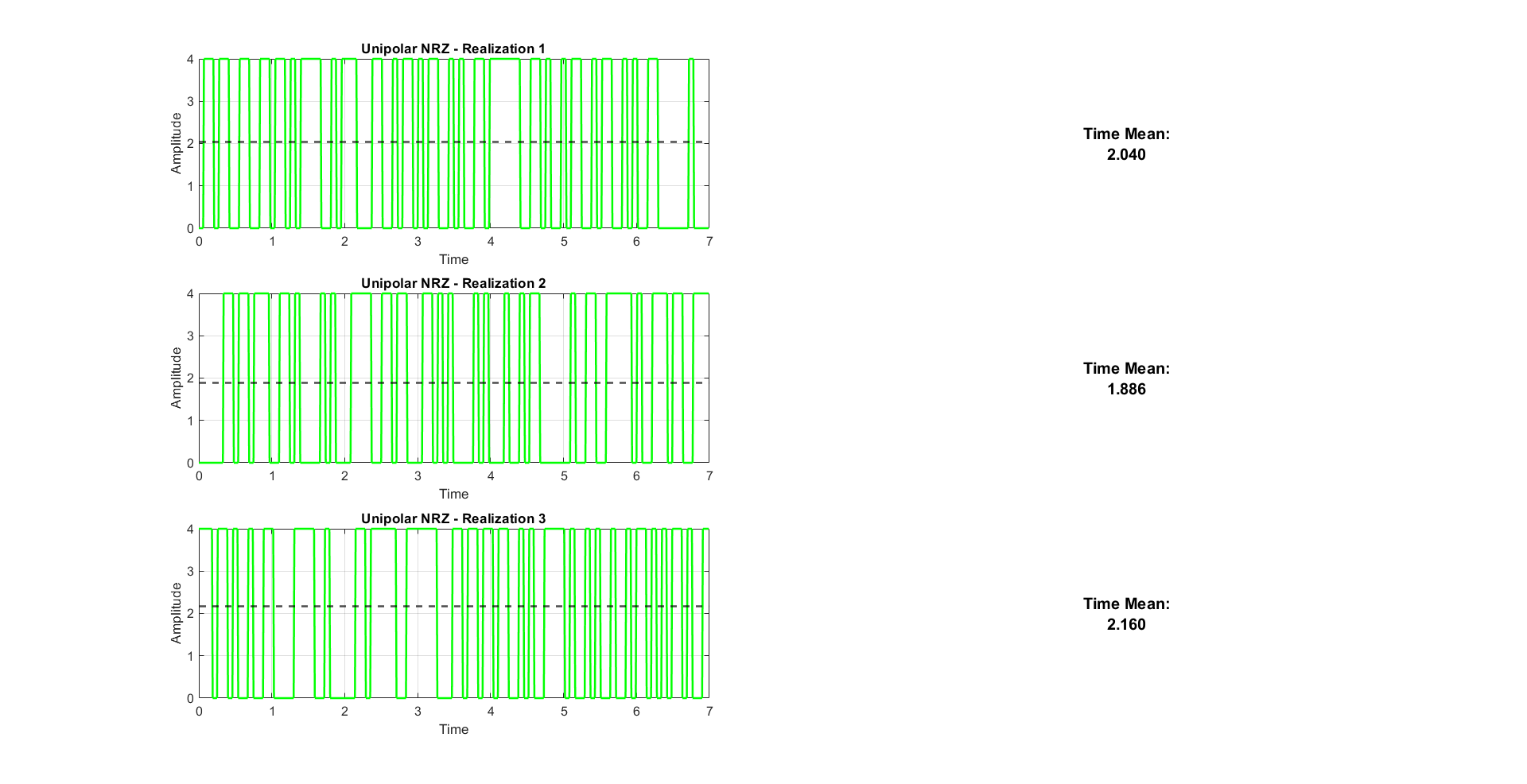
Yes, the process is stationary (WSSP) because the mean is constant function in time as shown in **Error! Reference source not found.** and the autocorrelation depends only on the time difference not the absolute time.

## The time mean and autocorrelation function for one waveform

### Time Mean

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

Figure 23 Time Mean for Uni Polar



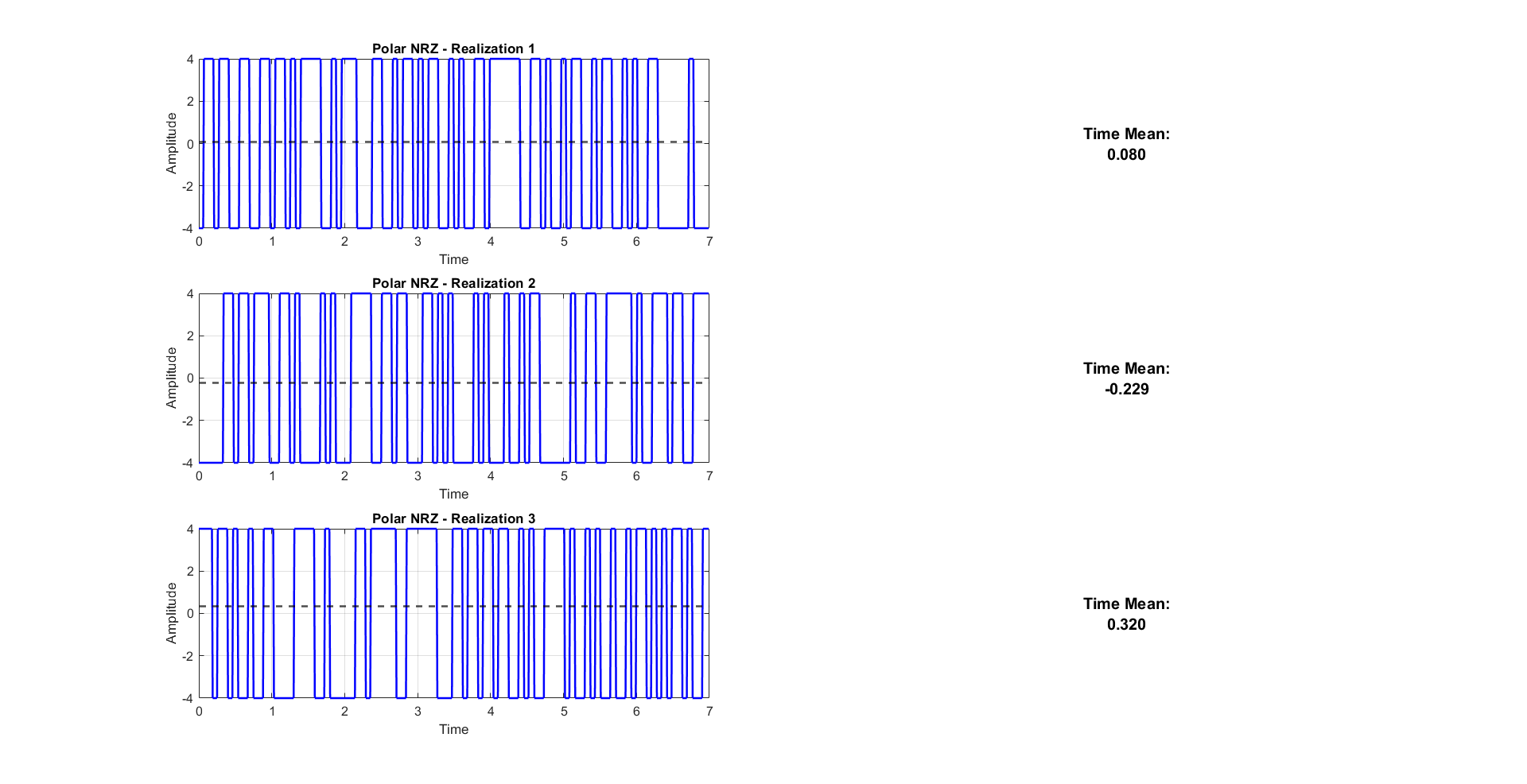
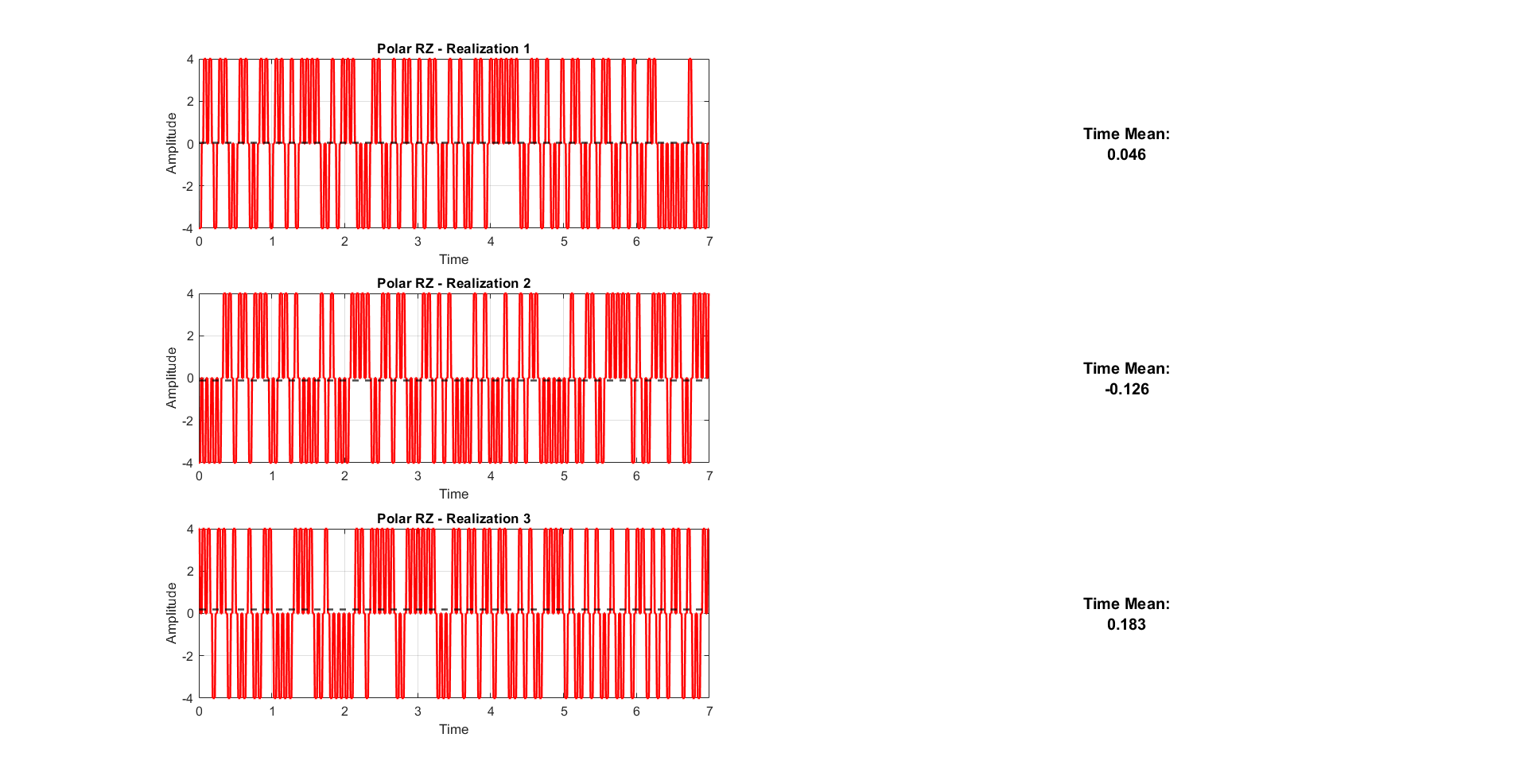
* As expected, polar RZ & NRZ have almost zero mean and the uni polar has mean around 2 Because its amplitude ranges from 0 : 4

Figure 24 Time Mean for Polar NRZ

Figure 25 Time Mean for Polar RZ

### Time Mean Vs Realization

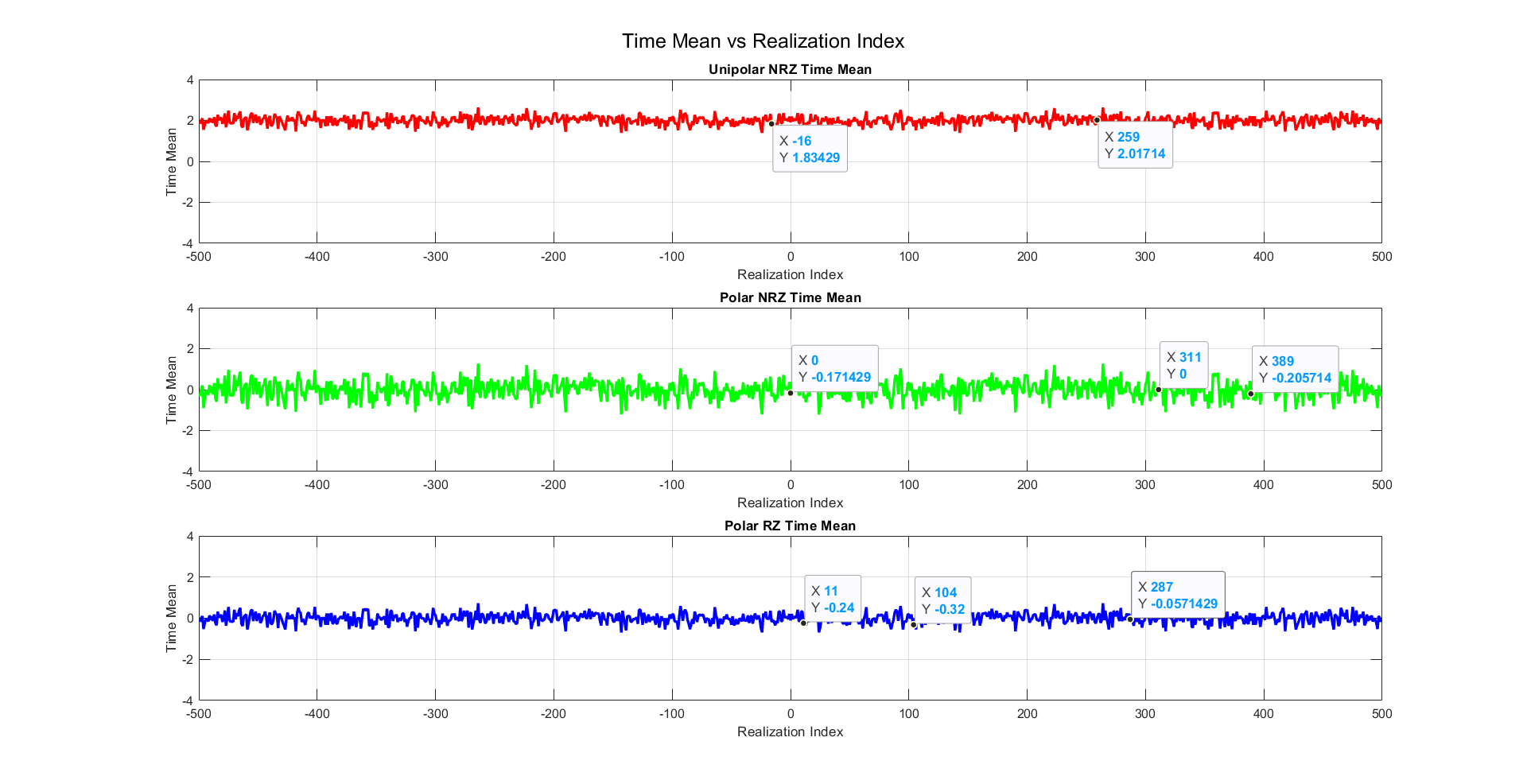
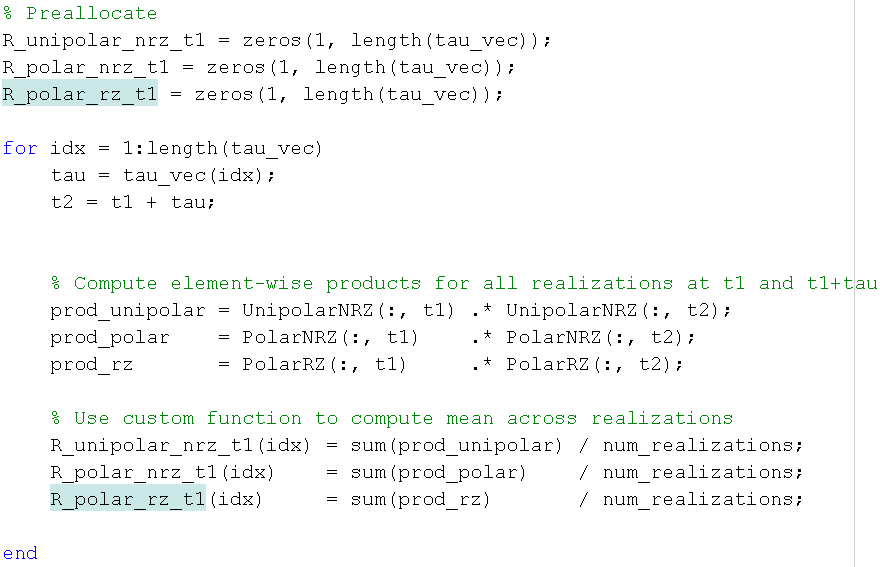


Figure 26 Time Mean Vs Realization

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

### Time Auto Correlation

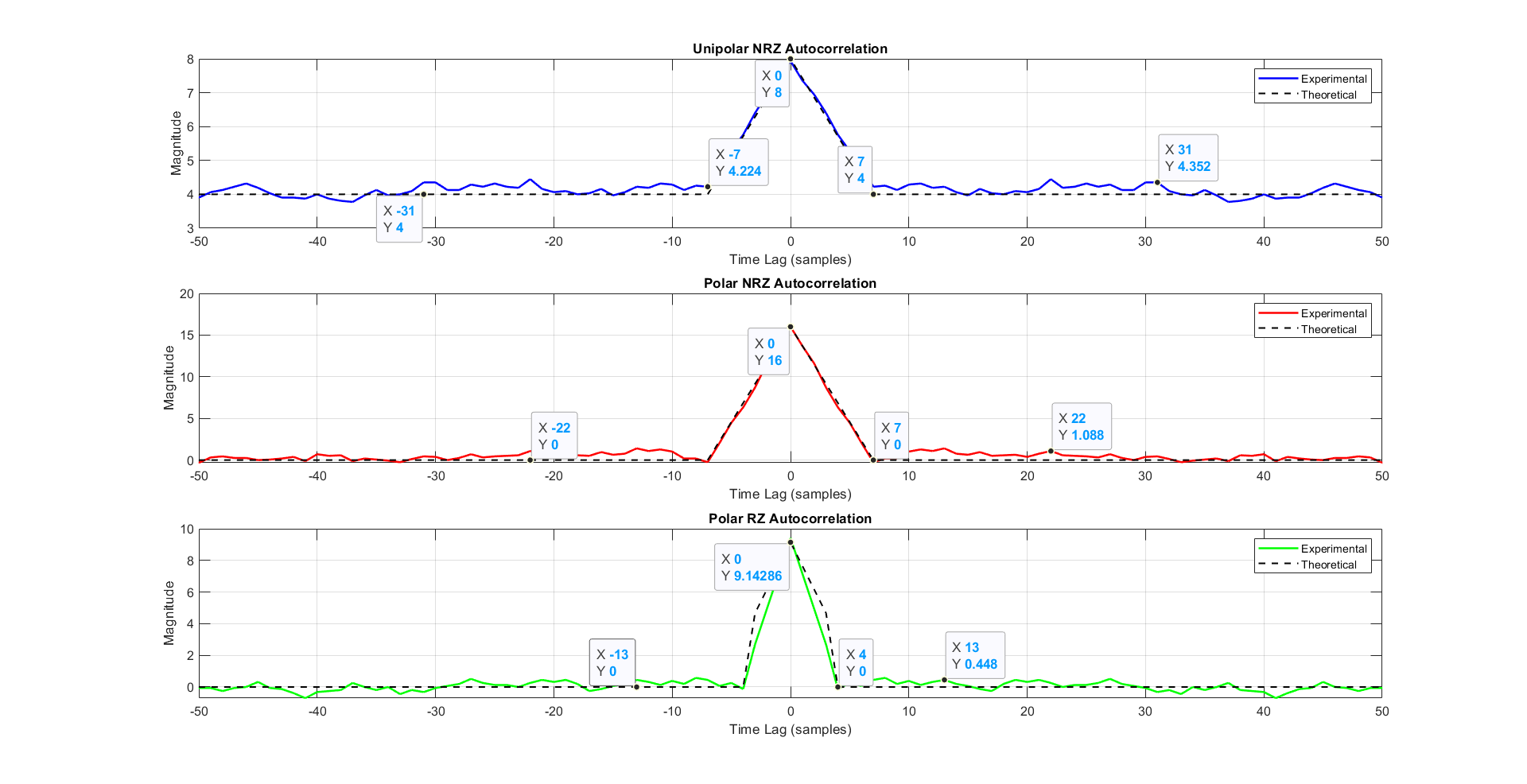
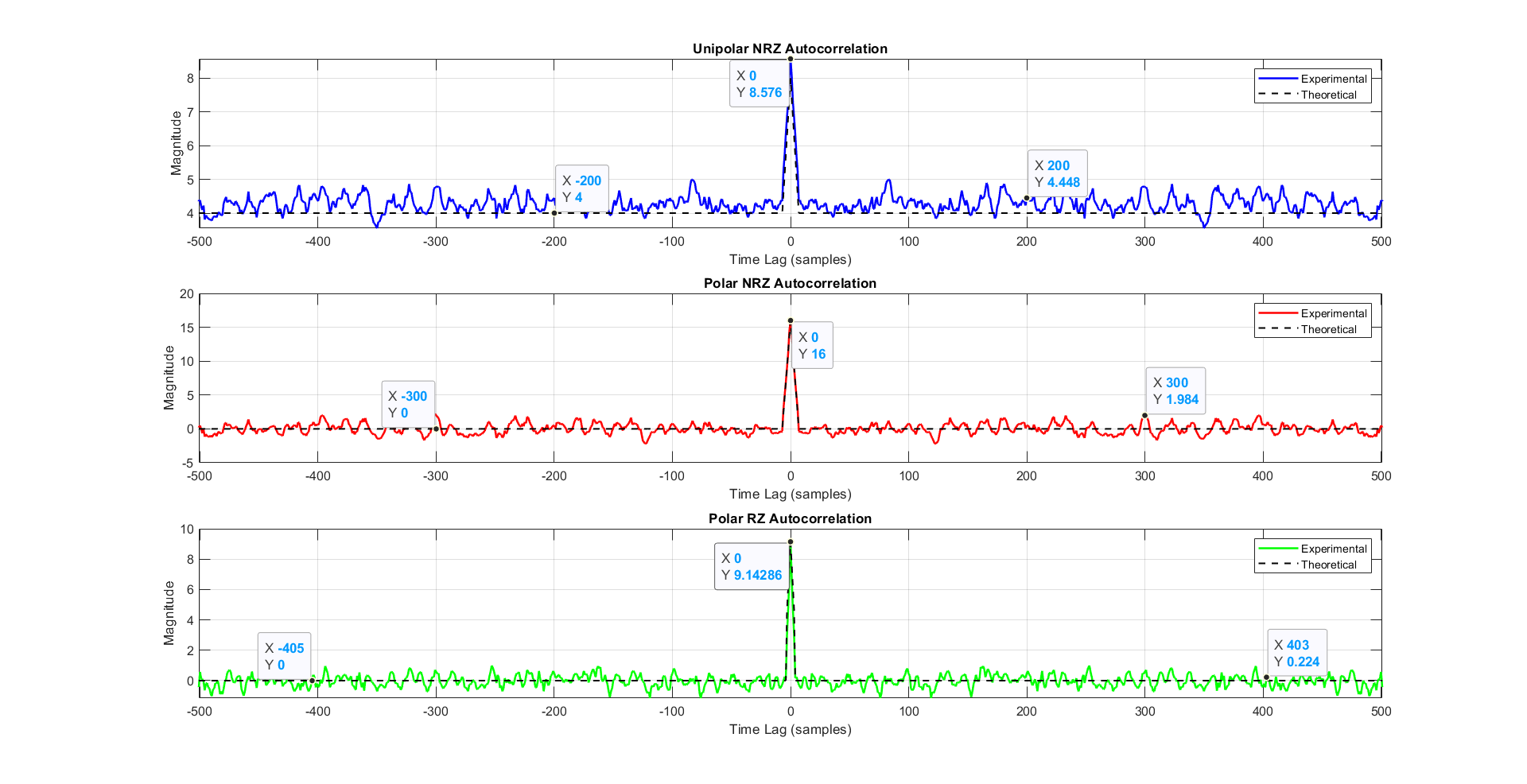
For the time Auto Correlation we’re going to use this function



### Time Auto Correlation for one wave form:

Figure 27 Time Auto Correction plot zoomed

Figure 28 Time Auto Correction plot



As shown in the graphs:

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

## Is The Random Process Ergodic?

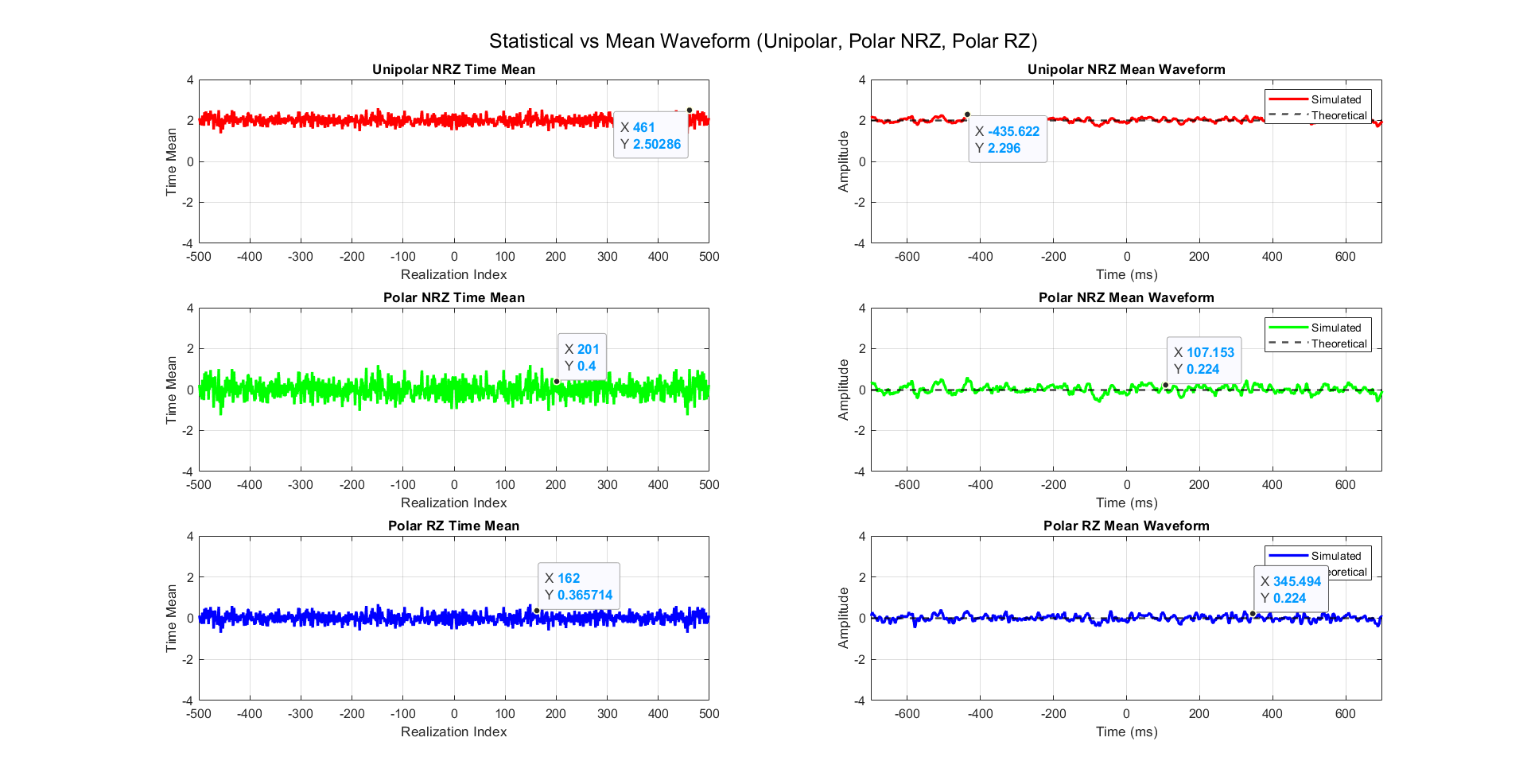


Figure 29 Time Mean vs Statistical

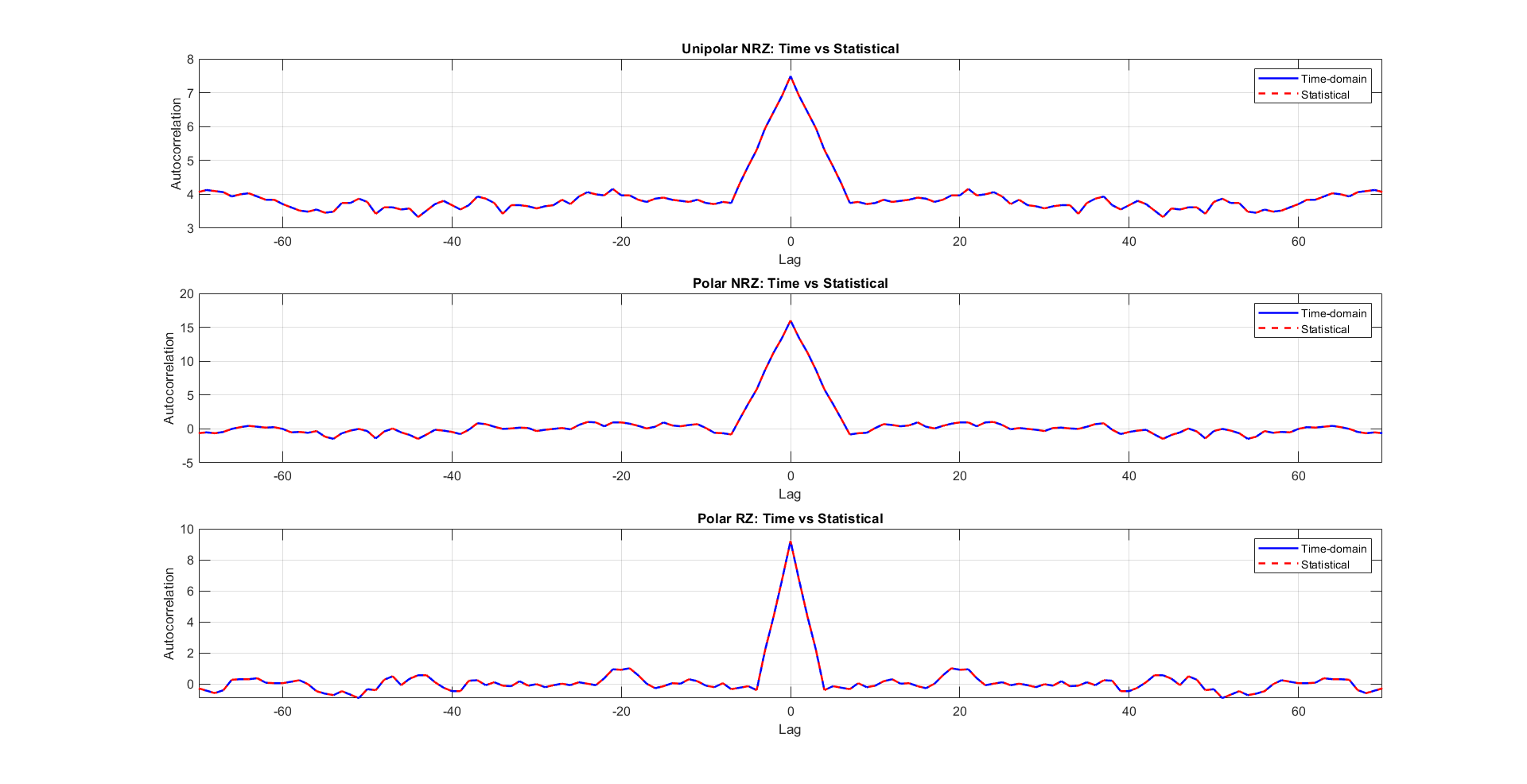


Figure 30 Time Auto Correlation Vs Statistical

* For the **mean**, the Time mean is almost equal to the statistical mean.
* For the **Auto Correlation,** the Time looks almost identical to the statistical.

But, There not fully identical as we ran this code snippet



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

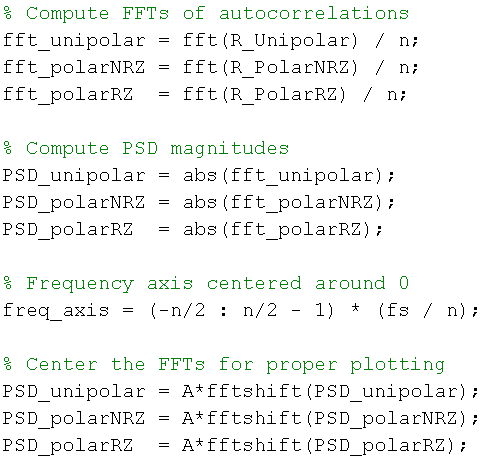
* Yes, because the time mean ≈ the Statistical mean and the time autocorrelation is ≈ the ensemble autocorrelation.

**Then this process is ergodic**

## the PSD & Bandwidth of the Ensemble

### PSD using fft:

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



* We take the Fourier transform of the avg time autocorrelation **= 0.5\*(R(t1)+ R(t2))** then centralize the graph around zero.
* since
* **For the BW**
* the BW is the frequency of the first zero of sinc^2 function (intersection with frequency-axis)

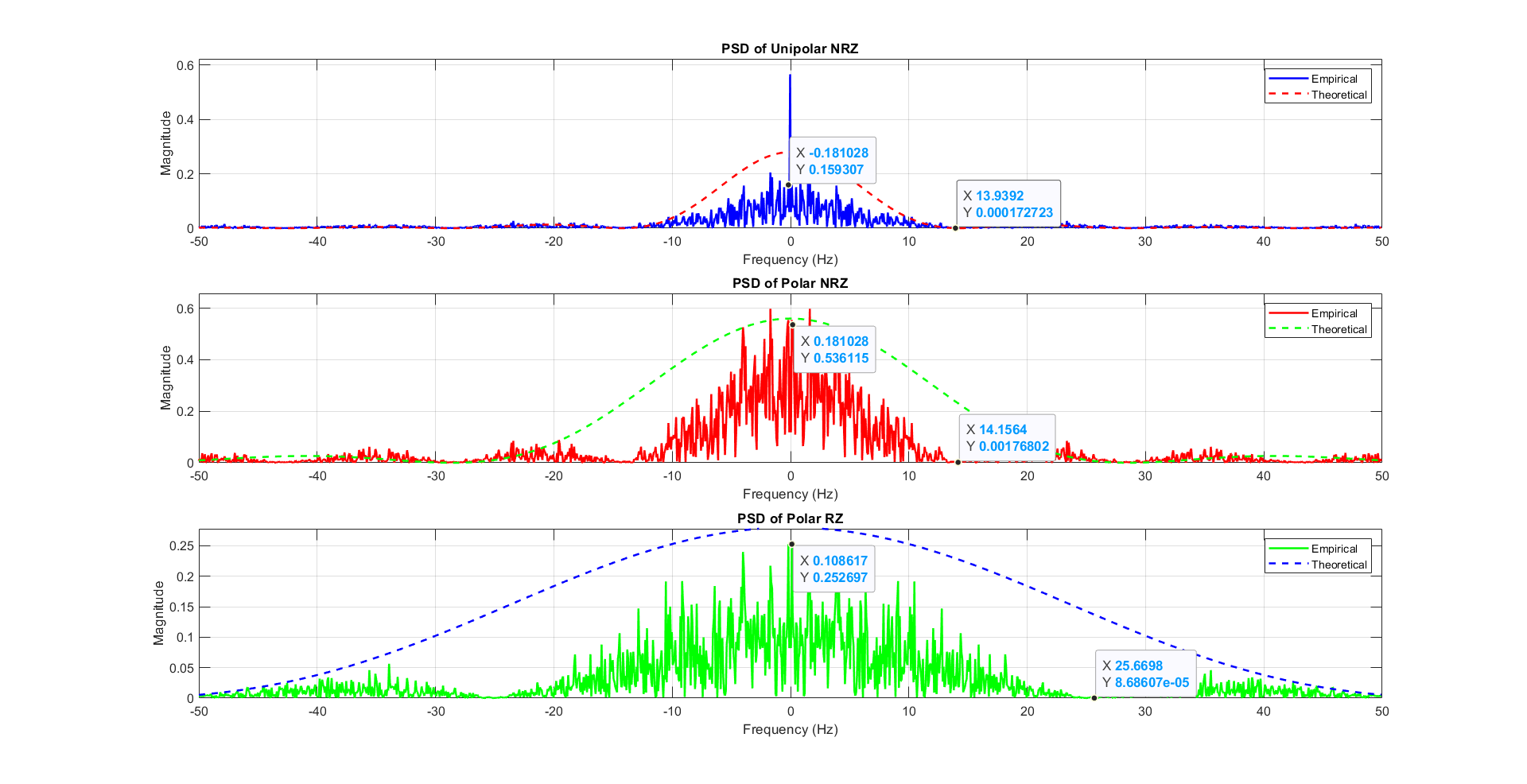


Figure 31 PSD plot of the Ensemble

**Annotations**

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

### Theoritical PSD:

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

|  |  |
| --- | --- |
| Line Code | PSD |
| Uni Polar | S(f)=A2/4\*Tb\*(sin(πfTb)/πfTb)2 + A2/4\* δ(f) |
| Polar NRZ | S(f)=A2\*Tb\*(sin(πfTb/2)/πfTb/2)2 |
| Polar RZ | S(f)=A2\*Tb\*(sin(πfTb/4)/πfTb/4)2 |

Note that:

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

So by comparing the practical vs theoretical:

|  |  |  |
| --- | --- | --- |
| Line Code | Theoritcal PSD at f=0 | Paractical PSD at f=0 |
| Uni Polar | A2/4\*Tb =0.28 | 0.159 |
| Polar NRZ | A2/2\*Tb =0.56 | 0.536 |
| Polar RZ | A2/4\*Tb =0.28 | 0.252 |

For **BW:**

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

# References:

[1] **Dr. Mohammed Nafie,** "Lecture Slides in Advanced Communications," ELC4020 Advanced Communication Systems, Cairo University, 2025.

[2] **Eng. Mohamed Khaled,** "Section Slides in Advanced Communications," ELC4020 Advanced Communication Systems, Cairo University, 2025.

[**3]** <https://github.com/youefkh05/Advanced_Communication_Coding>

# Appendix

%%------------------------------------------------------------

% Problem 1: Binary Huffman Coding

%------------------------------------------------------------

clc; clear; close all force;

% Given Symbols probabilities

symbols = {'A','B','C','D','E','F','G'};

P = [0.35 0.30 0.20 0.10 0.04 0.005 0.005];

% Create Input Dictionary

[dict\_input,err\_flag, H] = create\_symbols\_dictionary(symbols, P);

% Check Input

if err\_flag ==1

disp('? Stopping execution due to invalid dictionary.');

return; % exits the current script or function

end

% Print the dictionary neatly

print\_symbols\_dic(dict\_input, H);

% -------------------------------------------------------------------------

% Manual Huffman Coding (with custom output)

% -------------------------------------------------------------------------

dict\_huffman = huffman\_encoding\_visual(dict\_input);

disp('--- Manual Huffman Encoding ---');

disp(dict\_huffman);

% Print the codded dictionary neatly

print\_coded\_dict(dict\_huffman, H,"Huffman");

%%------------------------------------------------------------

% Problem 2: Binary Fano Coding

%------------------------------------------------------------

% -------------------------------------------------------------------------

% Manual Fano Coding (with custom output)

% -------------------------------------------------------------------------

dict\_Fano = fano\_encoding\_visual(dict\_input)

disp('--- Manual Fano Encoding ---');

disp(dict\_Fano);

% Print the codded dictionary neatly

print\_coded\_dict(dict\_Fano, H,"Fano");

%%

% -------------------------------------------------------------------------

% Function Definition

% -------------------------------------------------------------------------

%% -------------------------------------------------------------------------

% Entropy Calculation

% -------------------------------------------------------------------------

function H = entropy\_calc(P)

%ENTROPY\_CALC Compute the source entropy H(P(x))

% H = entropy\_calc(P)

% P : vector of symbol probabilities

% H : entropy in bits

% Validate input

if any(P < 0) || abs(sum(P) - 1) > 1e-6

warning('Probabilities should sum to 1. Normalizing...');

P = P / sum(P);

end

% Remove zeros (to avoid log2(0))

P = P(P > 0);

% Compute entropy

H = -sum(P .\* log2(P));

end

%% -------------------------------------------------------------------------

% Average Length Calculation

% -------------------------------------------------------------------------

function L = average\_length\_calc(dict)

%AVERAGE\_LENGTH\_CALC Compute average codeword length L(C)

% L = average\_length\_calc(dict, P)

% dict : Huffman dictionary cell array {symbol, code}

% P : vector of symbol probabilities (same order as dict)

%

% If P is empty, it tries to extract from dict(:,2) if present

% L : average code length

P = dict(:,2);

% --- Handle inputs ---

if nargin < 2 || isempty(P)

% Check if dict has a probability column (3 columns)

if size(dict, 2) >= 3 && isnumeric(dict{1,2})

P = cell2mat(dict(:,2));

codes = dict(:,3);

else

error('Probability vector P is required or must be in dict(:,2)');

end

else

% Extract codes (assumed in 2nd column)

codes = dict(:,3);

end

% --- Compute code lengths ---

code\_lengths = cellfun(@length, codes);

% --- Normalize probabilities ---

P = P(:) / sum(P);

% --- Check dimensions ---

if length(P) ~= length(code\_lengths)

error('Number of probabilities does not match number of codewords.');

end

% --- Compute average code length ---

L = sum(P .\* code\_lengths);

end

%% -------------------------------------------------------------------------

% Efficiency Calculation

% -------------------------------------------------------------------------

function eta = efficiency\_calc(H, L)

%EFFICIENCY\_CALC Compute Huffman coding efficiency ?

% eta = efficiency\_calc(H, L)

% H : entropy

% L : average code length

% eta : efficiency in percentage (%)

if L <= 0

error('Average length L must be positive.');

end

eta = (H / L) \* 100;

end

%% -------------------------------------------------------------------------

% Print Kraft Inequality Function

% -------------------------------------------------------------------------

function [kraft\_sum, kraft\_flag]=kraft\_analysis(dict)

% KRAFT\_ANALYSIS Compute Kraft's inequality and visualize Kraft tree

% kraft\_analysis(dict)

%

% Input:

% dict : cell array {N x 3} ? {symbol, P, code}

%

% Example:

% dict = {'A','0'; 'B','10'; 'C','110'; 'D','111'};

% kraft\_analysis(dict);

if ~iscell(dict) || size(dict,2) < 2

error('Input must be a cell array {symbol, code}');

end

% Extract codes

codes = dict(:,3);

N = length(codes);

% --- 1. Compute Kraft's inequality ---

code\_lengths = cellfun(@length, codes);

kraft\_sum = sum(2.^(-code\_lengths));

fprintf('\n=== Kraft Inequality Check ===\n');

fprintf('Sum(2^{-l\_i}) = %.4f\n', kraft\_sum);

if abs(kraft\_sum - 1) < 1e-6

fprintf('? Code satisfies equality ? Complete Prefix Code.\n');

kraft\_flag=2;

elseif kraft\_sum < 1

fprintf('? Code satisfies inequality (valid but not complete).\n');

kraft\_flag=1;

else

fprintf('? Invalid prefix code (violates Kraft''s inequality).\n');

kraft\_flag=0;

end

end

%% -------------------------------------------------------------------------

% Create Dictionary Input Definition

% -------------------------------------------------------------------------

function [dict\_input,err\_flag, H] = create\_symbols\_dictionary(symbols, P)

%CREATE\_DICTIONARY Combines symbols and probabilities into a validated dictionary.

%

% dict\_input = create\_dictionary(symbols, P)

%

% Inputs:

% symbols - cell array of symbols, e.g. {'A','B','C'}

% P - corresponding probabilities (row or column vector)

%

% Output:

% dict\_input - cell array {symbol, probability}

%

% Example:

% symbols = {'A','B','C'};

% P = [0.5 0.3 0.2];

% dict\_input = create\_dictionary(symbols, P);

% Combine into dictionary-like cell array

dict\_input = [symbols(:), num2cell(P(:))];

% Assume not great until great

err\_flag = 1;

% Validate using the check\_symbols() function

[ok, msg] = check\_symbols(dict\_input);

% Display validation result

if ok

disp('? Dictionary is valid!');

err\_flag =0;

else

disp(['? Error: ' msg]);

err\_flag = 1;

end

% ---------------------------------------------------------------------

% Compute entropy (only if valid)

% ---------------------------------------------------------------------

H = entropy\_calc(P)

end

%% -------------------------------------------------------------------------

% Check Input Validation Function

% -------------------------------------------------------------------------

function [isValid, errMsg] = check\_symbols(dict\_input)

% CHECK\_SYMBOLS Validates a symbol-probability dictionary

%

% [isValid, errMsg] = check\_symbols(dict\_input)

%

% Input:

% dict\_input : Cell array {N×2}, where first column = symbols,

% second column = probabilities

%

% Output:

% isValid : Logical true if valid, false otherwise

% errMsg : String describing validation error (if any)

% Default output

isValid = false;

errMsg = '';

try

% Extract symbols and probabilities

symbols = dict\_input(:, 1);

P = cell2mat(dict\_input(:, 2));

% Check same length

if numel(symbols) ~= numel(P)

errMsg = 'Symbols and probabilities must have the same length.';

return;

end

% Check probabilities sum to 1 (within tolerance)

if abs(sum(P) - 1) > 1e-6

errMsg = sprintf('Probabilities do not sum to 1 (sum = %.6f).', sum(P));

return;

end

% Check all probabilities are positive

if any(P <= 0)

errMsg = 'All probabilities must be positive.';

return;

end

% If all checks passed

isValid = true;

catch ME

errMsg = ['Invalid dictionary input: ' ME.message];

end

end

%% -------------------------------------------------------------------------

% Print Dictionary Function

% -------------------------------------------------------------------------

function print\_symbols\_dic(dict\_input, H)

% PRINT\_SYMBOLS\_DIC Displays a formatted version of the symbol dictionary in a figure,

% and shows the calculated source entropy.

%

% print\_symbols\_dic(dict\_input, H)

%

% Inputs:

% dict\_input - cell array {symbol, probability}

% H - source entropy (bits/symbol)

% Validate input

if nargin < 1 || isempty(dict\_input)

error('Input dictionary is empty or missing.');

return;

end

% Convert symbols to char (uitable can't handle string objects)

symbols = cellfun(@char, dict\_input(:,1), 'UniformOutput', false);

probs = cell2mat(dict\_input(:,2));

% Display result in Command Window

fprintf('\nInformation Source Entropy: H = %.4f bits/symbol\n', H);

fprintf('-----------------------------------------------------\n');

% Create a responsive UI figure

f = uifigure('Name', 'Symbol Dictionary', ...

'NumberTitle', 'off', ...

'Color', 'w', ...

'Position', [500 400 350 320]);

% Format probabilities as strings

probStr = arrayfun(@(p) sprintf('%.4f', p), probs, 'UniformOutput', false);

% Combine into table data

data = [symbols probStr];

% Create a grid layout (auto-resizes)

gl = uigridlayout(f, [3,1]);

gl.RowHeight = {'fit', '1x', 'fit'}; % title, table, entropy

gl.ColumnWidth = {'1x'};

gl.Padding = [10 10 10 10];

% --- Title ---

uilabel(gl, ...

'Text', '--- Input Symbol Dictionary ---', ...

'FontSize', 14, ...

'FontWeight', 'bold', ...

'HorizontalAlignment', 'center');

% --- Table ---

uitable(gl, ...

'Data', data, ...

'ColumnName', {'Symbol', 'Probability'}, ...

'FontSize', 12, ...

'ColumnWidth', {'1x', '1x'}, ...

'RowStriping', 'on');

% --- Entropy Display ---

uilabel(gl, ...

'Text', sprintf('Entropy: H = %.4f bits/symbol', H), ...

'FontSize', 12, ...

'FontWeight', 'bold', ...

'FontColor', [0 0.3 0.7], ...

'HorizontalAlignment', 'center');

end

%% -------------------------------------------------------------------------

% Print Coded Dictionary Function (General Purpose)

% -------------------------------------------------------------------------

function print\_coded\_dict(dict, H, title\_str)

% PRINT\_CODED\_DICT Display coding dictionary with entropy, avg length, efficiency, and Kraft analysis.

%

% print\_coded\_dict(dict, H, title\_str)

%

% Inputs:

% dict - cell array {symbol, probability, code}

% H - entropy (bits/symbol)

% title\_str - string title for the table window (e.g. 'Fano Coding Results')

%

% This function:

% • Calculates average codeword length L(C)

% • Calculates efficiency ? = (H / L) \* 100%

% • Checks Kraft’s inequality

% • Displays results in MATLAB UI + command window

%

% Example:

% print\_coded\_dict(fano\_dict, H, 'Fano Coding Summary');

% === Validate input ===

if nargin < 1 || isempty(dict)

disp('Input dictionary is missing or empty.');

return;

end

if size(dict,2) < 3

disp('Dictionary must have 3 columns: {symbol, probability, code}.');

return;

end

if nargin < 3 || isempty(title\_str)

title\_str = 'Coded Dictionary Summary';

end

% === Extract data ===

symbols = cellfun(@char, dict(:,1), 'UniformOutput', false);

P = cell2mat(dict(:,2));

codes = dict(:,3);

% === Compute metrics ===

L = average\_length\_calc(dict);

eta = efficiency\_calc(H, L);

[kraft\_sum, kraft\_flag] = kraft\_analysis(dict);

% === Print to Command Window ===

fprintf('\n--- %s ---\n', title\_str);

fprintf('Symbol\tProb.\t\tCode\n');

fprintf('-----------------------------------------\n');

for i = 1:length(symbols)

fprintf('%s\t%.4f\t\t%s\n', symbols{i}, P(i), codes{i});

end

fprintf('-----------------------------------------\n');

fprintf('Entropy (H): %.4f bits/symbol\n', H);

fprintf('Average length (L): %.4f bits/symbol\n', L);

fprintf('Efficiency (?): %.2f %%\n', eta);

fprintf('Kraft Sum: %.4f\n', kraft\_sum);

if kraft\_flag == 2

fprintf('Kraft Result: ? Complete Prefix Code\n');

elseif kraft\_flag == 1

fprintf('Kraft Result: ? Valid but Not Complete\n');

else

fprintf('Kraft Result: ? Invalid Code\n');

end

% === UI Figure ===

f = uifigure('Name', title\_str, ...

'NumberTitle', 'off', ...

'Color', 'w', ...

'Position', [500 200 480 450]);

gl = uigridlayout(f, [3 1]);

gl.RowHeight = {'fit', '1x', 'fit'};

gl.Padding = [10 10 10 10];

% --- Title ---

uilabel(gl, ...

'Text', ['--- ' title\_str ' ---'], ...

'FontSize', 14, ...

'FontWeight', 'bold', ...

'HorizontalAlignment', 'center');

% --- Table ---

data = [symbols, arrayfun(@(p) sprintf('%.4f', p), P, 'UniformOutput', false), codes];

uitable(gl, ...

'Data', data, ...

'ColumnName', {'Symbol', 'Probability', 'Code'}, ...

'FontSize', 12, ...

'RowStriping', 'on', ...

'ColumnWidth', {'1x', '1x', '1x'});

% --- Summary Line ---

uilabel(gl, ...

'Text', sprintf('H = %.4f | L = %.4f | ? = %.2f %% | Kraft = %.4f', H, L, eta, kraft\_sum), ...

'FontSize', 12, ...

'FontWeight', 'bold', ...

'FontColor', [0 0.3 0.7], ...

'HorizontalAlignment', 'center');

end

%% -------------------------------------------------------------------------

% Huffman Encoding with Visualization Function

% -------------------------------------------------------------------------

function dict = huffman\_encoding\_visual(dict\_input)

%HUFFMAN\_ENCODING\_VISUAL Visual Huffman encoding with full table output (UI-based)

%

% dict = huffman\_encoding\_visual(symbols, P)

% - symbols: cell array of symbol names (e.g. {'A','B','C','D','E','F','G'})

% - P: vector of probabilities (same length as symbols)

%

% Creates a UI figure showing the probability & code propagation table,

% and prints the final Huffman dictionary.

% get the info from dictionary

symbols = dict\_input(:,1);

P = cell2mat(dict\_input(:,2));

% === Input Validation ===

if numel(symbols) ~= numel(P)

error('Symbols and probabilities must have same length.');

end

% === Normalize probabilities ===

P = P(:);

P = P / sum(P);

% === Step 1: Generate merging history ===

history\_table = merge\_probabilities(P);

% === Step 1 Visualization ===

visualize\_merging\_process(P, history\_table);

% === Step 2: Assign Huffman codes ===

history\_table\_full = assign\_coding(history\_table);

% === Step 3: Prepare data for visualization ===

% Convert numeric NaNs to empty strings for table display

final\_visual\_data = cell(size(history\_table\_full));

for r = 1:size(history\_table\_full,1)

for c = 1:size(history\_table\_full,2)

val = history\_table\_full{r,c};

if isnumeric(val)

if isnan(val)

final\_visual\_data{r,c} = '';

else

final\_visual\_data{r,c} = num2str(val, '%.4f');

end

else

final\_visual\_data{r,c} = val;

end

end

end

% Generate column headers (P1, C1, P2, C2, ...)

numCols = size(history\_table\_full,2);

final\_visual\_headers = cell(1,numCols);

for c = 1:numCols

if mod(c,2)==1

final\_visual\_headers{c} = sprintf('P%d', ceil(c/2)-1);

else

final\_visual\_headers{c} = sprintf('C%d', ceil(c/2)-1);

end

end

% === Step 4: Build UI Visualization ===

close all;

f = uifigure('Name','Huffman Encoding Visualization', ...

'Position',[100 100 1000 500]);

gl = uigridlayout(f,[2 1]);

gl.RowHeight = {'fit','1x'};

uilabel(gl, ...

'Text','Huffman Encoding: Probability and Code Evolution (P/C Steps)', ...

'FontSize',16, ...

'FontWeight','bold', ...

'HorizontalAlignment','center');

% Column widths (narrow for numeric columns, wider for code columns)

col\_widths = repmat({70}, 1, numCols);

col\_widths(2:2:end) = {100}; % widen code columns

uitable(gl, ...

'Data',final\_visual\_data, ...

'ColumnName',final\_visual\_headers, ...

'RowName',{}, ...

'FontSize',12, ...

'ColumnWidth',col\_widths, ...

'RowStriping','on', ...

'BackgroundColor',[1 1 1; 0.95 0.95 1]);

% === Step 5: Extract Final Huffman Dictionary ===

% Make a copy

dict = dict\_input;

% Ensure dict has at least 3 columns

if size(dict,2) < 3

dict(:,end+1:3) = {[]};

end

dict(:,3)=history\_table\_full(:,2);

% === Step 6: Console Output ===

firstPcol = 1;

firstCcol = 2;

probs = cell2mat(history\_table\_full(:, firstPcol));

codes = history\_table\_full(:, firstCcol);

validIdx = ~isnan(probs);

symbols = symbols(validIdx);

codes = codes(validIdx);

probs = probs(validIdx);

fprintf('\n--- Final Huffman Codes ---\n');

for i = 1:length(symbols)

fprintf('Symbol %s (%.4f): %s\n', symbols{i}, probs(i), codes{i});

end

fprintf('===============================================================\n\n');

end

% === Probability Merge helper function ===

function history\_table = merge\_probabilities(P)

%MERGE\_PROBABILITIES Builds Huffman probability merging history (descending)

%

% history\_table = merge\_probabilities(P)

%

% Input:

% P - vector of symbol probabilities

%

% Output:

% history\_table - table of probabilities after each merge

% Columns: P0, P1, P2, ... (N-1 total)

%

% Note: Probabilities are shown in descending order.

% --- Input check ---

if numel(P) < 2

error('At least two probabilities are required.');

end

% --- Initialization ---

P = P(:);

P = sort(P, 'descend'); % sort descending

N = numel(P);

% Number of P columns = N - 1

numCols = N - 1;

maxRows = N;

% Initialize history as cell

history = cell(maxRows, numCols);

% --- Step 0: Fill P0 (descending order) ---

for i = 1:maxRows

history{i,1} = P(i);

end

curP = P;

% --- Iteratively merge ---

for step = 2:numCols

% Sort ascending to pick smallest two

curP = sort(curP, 'ascend');

if numel(curP) >= 2

p1 = curP(1);

p2 = curP(2);

mergedP = p1 + p2;

% Remove two smallest and add merged one

curP = [mergedP; curP(3:end)];

end

% Sort descending for display

curP = sort(curP, 'descend');

% Fill current column

for r = 1:maxRows

if r <= numel(curP)

history{r,step} = curP(r);

else

history{r,step} = NaN;

end

end

end

% --- Column names ---

colNames = cell(1,numCols);

for i = 1:numCols

colNames{i} = sprintf('P%d', i-1);

end

% --- Convert to table ---

history\_table = cell2table(history, 'VariableNames', colNames);

end

% === Visualization of Probability Merging Process ===

function visualize\_merging\_process(P, history\_table)

% VISUALIZE\_MERGING\_PROCESS Display a UI table of Huffman probability merging

%

% visualize\_merging\_process(P, history\_table)

%

% Displays the merging steps used in Huffman encoding as a clean table UI.

%

% Inputs:

% P - Original probability vector (used for scaling)

% history\_table - Table of merging probabilities (output of merge\_probabilities)

%

% Example:

% history\_table = merge\_probabilities([0.4 0.2 0.15 0.15 0.1]);

% visualize\_merging\_process([0.4 0.2 0.15 0.15 0.1], history\_table);

% --- Input validation ---

if nargin < 2

error('Usage: visualize\_merging\_process(P, history\_table)');

end

if ~istable(history\_table)

error('history\_table must be a MATLAB table.');

end

% === Step 1: Convert numeric values (NaN ? empty string for display) ===

merge\_visual\_data = cell(size(history\_table));

for r = 1:size(history\_table,1)

for c = 1:size(history\_table,2)

val = history\_table{r,c};

if isnumeric(val)

if isnan(val)

merge\_visual\_data{r,c} = '';

else

merge\_visual\_data{r,c} = num2str(val, '%.4f');

end

else

merge\_visual\_data{r,c} = val;

end

end

end

% === Step 2: Build UI Table Figure ===

f\_merge = uifigure('Name','Step 1: Probability Merging History', ...

'Position',[150 150 700 400]);

gl\_merge = uigridlayout(f\_merge,[2 1]);

gl\_merge.RowHeight = {'fit','1x'};

uilabel(gl\_merge, ...

'Text','Step 1: Huffman Probability Merging Process', ...

'FontSize',16, ...

'FontWeight','bold', ...

'HorizontalAlignment','center');

uitable(gl\_merge, ...

'Data',merge\_visual\_data, ...

'ColumnName',history\_table.Properties.VariableNames, ...

'RowName',{}, ...

'FontSize',12, ...

'ColumnWidth',repmat({80}, 1, width(history\_table)), ...

'RowStriping','on', ...

'BackgroundColor',[1 1 1; 0.95 0.95 1]);

% === Step 3: Optional console summary ===

fprintf('\n--- Probability Merging Steps ---\n');

disp(history\_table);

fprintf('=================================\n\n');

end

% === Assign code helper function ===

function history\_table\_full = assign\_coding(history\_table)

% assign\_coding - expands the history table and assigns Huffman codes

%

% Input:

% history\_table : numeric matrix or table (probability merging history)

%

% Output:

% history\_table\_full : cell array with 2N columns

% Odd columns: probability values

% Even columns: assigned codes

% If table, convert to numeric array

if istable(history\_table)

history\_table = table2array(history\_table);

end

% Determine size

[numRows, numCols] = size(history\_table);

newCols = 2 \* numCols;

% Initialize

history\_table\_full = cell(numRows, newCols);

% === Fill odd columns with probabilities ===

for col = 1:numCols

history\_table\_full(:, 2 \* col - 1) = num2cell(history\_table(:, col));

end

% === Initialize last code column (start with last merge) ===

lastPcol = 2 \* numCols - 1;

lastCcol = lastPcol + 1;

history\_table\_full{1, lastCcol} = '0';

history\_table\_full{2, lastCcol} = '1';

raw\_counter=1; %for parent assignment

% === Backward propagation of codes ===

for col = numCols:-1:2 % start from last column going backward

currPcol = 2 \* col - 1;

currCcol = currPcol + 1;

prevPcol = 2 \* (col - 1) - 1;

prevCcol = prevPcol + 1;

raw\_counter = raw\_counter+1;

% Get non-NaN values from P prev column

prevPvals = cell2mat(history\_table\_full(:, prevPcol));

prevPvals = prevPvals(~isnan(prevPvals));

% Get non-NaN values from C curr column

currCvals = history\_table\_full(:, currCcol);

% Identify merged value

if length(prevPvals) >= 2

mergedVal = prevPvals(end) + prevPvals(end-1);

else

continue;

end

% Find which row in current P col matches the mergedVal

currPvals = cell2mat(history\_table\_full(:, currPcol));

matchIdx = find(abs(currPvals - mergedVal) < 1e-12);

if numel(matchIdx) > 1

matchIdx = matchIdx(1); % take top one if duplicate

end

% Get parent code

parentCode = history\_table\_full{matchIdx, currCcol};

if isempty(parentCode)

parentCode = '';

end

% === Assign child codes ===

% Last two rows in previous P column are merged into this parent

history\_table\_full{raw\_counter, prevCcol} = [parentCode '0'];

history\_table\_full{raw\_counter+1, prevCcol} = [parentCode '1'];

% For each previous non-merged row (in display order top->bottom)

for ii = 1:(raw\_counter-1)

% Skip the rows that were just merged (raw\_counter and raw\_counter+1)

% Get the probability value in the previous column for this row

valPrev = history\_table\_full{ii, prevPcol};

if isnan(valPrev)

continue; % nothing to copy

end

% Find matching value in the current column (exclude merged parent)

currMatches = find(abs(currPvals - valPrev) < 1e-12);

% Remove the matchIdx (the merged parent) if it appears

currMatches(currMatches == matchIdx) = [];

if isempty(currMatches)

continue; % no corresponding match found

end

% If there are duplicates (two identical probabilities)

if numel(currMatches) > 1

% take both, and copy their codes to the two rows

history\_table\_full{ii, prevCcol} = currCvals{currMatches(1)};

if (ii+1) <= numRows

history\_table\_full{ii+1, prevCcol} = currCvals{currMatches(2)};

end

else

% single match — copy code directly

history\_table\_full{ii, prevCcol} = currCvals{currMatches(1)};

end

end

end

end

%% -------------------------------------------------------------------------

% Fano Encoding with Visualization Function

% -------------------------------------------------------------------------

function table\_out = fano\_encoding\_visual(dict\_input)

% FANO\_ENCODING\_VISUAL

% Fano encoding with dynamic stage-by-stage history table.

%

% Output table columns:

% Symbol | P0 | C0 | P1 | C1 | ... until convergence

% === STEP 1: INITIALIZE ===

symbols = dict\_input(:,1);

probs = cell2mat(dict\_input(:,2));

% Normalize and sort descending

probs = probs / sum(probs);

[probs, idx] = sort(probs, 'descend');

symbols = symbols(idx);

% Initialize codes and history

codes = repmat({''}, size(symbols));

history = cell(length(symbols), 0);

% Add first columns P0 and C0

history(:, end+1) = num2cell(probs); % P0

history(:, end+1) = codes; % C0

% Start with one full group

groups = {struct('idx', 1:length(symbols), 'prefix', '')};

iteration = 1;

% === STEP 2: ITERATIVE FANO GROUPING ===

while ~isempty(groups)

new\_groups = {};

for g = 1:length(groups)

cur = groups{g};

idxs = cur.idx;

prefix = cur.prefix;

% Skip if one symbol only

if numel(idxs) <= 1

continue;

end

pvals = probs(idxs);

% --- Compute reference dynamically ---

ref = 2^(-iteration);

% --- Decide split method (COMBINATION-BASED) ---

best\_diff = inf;

split\_idx = [];

n = length(pvals);

for mask = 1:(2^n - 1)

subset = bitget(mask, 1:n);

subset\_sum = sum(pvals(logical(subset)));

diff = abs(subset\_sum - ref);

if diff < best\_diff

best\_diff = diff;

split\_idx = find(subset);

end

end

% --- Create two new groups ---

g1 = idxs(split\_idx);

g2 = setdiff(idxs, idxs(split\_idx));

% Assign '0' and '1' respectively

for k = g1

codes{k} = [prefix '0'];

end

for k = g2

codes{k} = [prefix '1'];

end

% Queue for next stage

if ~isempty(g1)

new\_groups{end+1} = struct('idx', g1, 'prefix', [prefix '0']);

end

if ~isempty(g2)

new\_groups{end+1} = struct('idx', g2, 'prefix', [prefix '1']);

end

end

% Stop if all groups are singletons

if isempty(new\_groups)

break;

end

% Record stage snapshot

history(:, end+1) = num2cell(probs); % Pi

history(:, end+1) = codes; % Ci

groups = new\_groups;

iteration = iteration + 1;

end

% === STEP 3: COMPILE OUTPUT TABLE ===

cols = {'Symbol'};

for i = 0:(size(history,2)/2 - 1)

cols{end+1} = sprintf('P%d', i);

cols{end+1} = sprintf('C%d', i);

end

table\_out = [symbols history];

% Display as uitable

figure('Name','Fano Encoding Table','Position',[300 200 1100 400]);

uitable('Data', table\_out, 'ColumnName', cols, ...

'ColumnWidth', num2cell(repmat(80,1,numel(cols))), ...

'FontSize', 11, 'Units','normalized', 'Position',[0 0 1 1]);

% === STEP 3: FINAL OUTPUT (Symbol, Prob, Code) ===

table\_out = [symbols num2cell(probs) codes];

cols = {'Symbol', 'Probability', 'Code'};

end