**MAJOR PROJECT REPORT ON**

**“Code Detection Using MATLAB”**

**Submitted in Partial Fulfillment of the Requirements**

**For the Award**

**Of**

**Degree of B. Tech**

**To**



**Guru Gobind Singh Indraprastha University, Delhi**

**Under the Guidance of**

**MR. MAHENDER SINGH**

**Submitted By:**

**NAME-1 NAME-2**

**NAME-3 NAME-4**



**Electronics and Communication Department**

**Guru Tegh Bahadur Institute of Technology**

**G-8 Area, Rajouri Garden, New Delhi, Delhi-64**

****

**CERTIFICATE**

We hereby certify that the work which is being presented in the project report entitled

**“Code detection using MATLAB”** in the partial fulfillment of the requirements for the awardof **Bachelor of Tech.** in **Electronics and Communication Engineering** and submitted tothe **Electronics and Communication Engineering Department** of **Guru Tegh Bahadur Institute of Technology**, New Delhi is an authentic record of my own work carried out during a period from **January 2016 to June 2016**, under the guidance of **MR. MAHINDER SINGH**, Electronics and Communication Engineering Department. The matter presented in this project has not been submitted by us for the award of other degree elsewhere.

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

**MR. AMRISH KUMAR MR. MAHENDER SINGH**

**(Project Coordinator) (Project Mentor)**

**MS. MANINDER KAUR DR. VANEET SINGH**

**(Project Coordinator) (H.O.D. ECE Department)**

**MS. SHIVANI DUGGAL**

**(Project Coordinator)**

**ACKNOWLEDGEMENT**

We would like to acknowledge the contributions of the following people, without whose help and guidance this project would not have been completed.

We acknowledge the counsel and support of our project mentor **MR. MAHINDER SINGH, Assistant Professor**, Electronics and Communication Engineering Department, with respect and gratitude whose expertise, guidance, support, encouragement, and enthusiasm has made this project possible. His feedback vastly improved the quality of this report and provided an enthralling experience. I am indeed proud and fortunate to be supervised by her.

We are also thankful to **Dr. VANEET SINGH, H.O.D.** Electronics and Communication Engineering Department, **Guru Tegh Bahadur Institute of Technology,** New Delhi for his constant encouragement, valuable suggestions and moral support and blessings.

We shall ever remain indebted to the faculty members of **GTBIT, New Delhi.**

Finally, yet importantly, we would like to express our heartfelt thanks to our friends/classmates for their help and wishes for successful completion of this project. This acknowledgement will remain incomplete if we fail to express our deep sense of obligation to our parents and god for their consistent blessings and encouragement.

**NAME1 NAME2**

**ROLL NO ROLL NO**

**NAME3 NAME4**

**ROLL NO ROLL NO**

**CONTENTS**

certificateiII

acknowledgementiV

introduction1

software requirements3

software used3

MATLAB 8.03

TYPES OF CODING5

UNIPOLAR NRZ5

UNIPOLAR RZ6

POLAR NRZ7

POLAR RZ8

MANCHESTER9

BIPOLAR12

DaTA COMPRESSION TECHNIQUES12

SHANNON- FANO CODING12

HUFFMAN CODING13

appendix a14

code for LINE CODE PLOTTING14

code for SHANNON CODING19

code for HUFFMAN CODING23

code for LINE CODE PSD25

references28

1. **INTRODUCTION**

The terminology line coding originated in telephony with the need to transmit digital information

across a copper telephone line; more specifically, binary data over a digital repeater line. The

concept of line coding, however, readily applies to any transmission line or channel. In a digital communication system, there exists a known set of symbols to be transmitted. These can be designated as {mi}, i = 1, 2,..., N, with a probability of occurrence {pi}, i = 1, 2,..., N, where the sequentially transmitted symbols are generally assumed to be statistically independent. The conversion or coding of these abstract symbols into real, temporal waveforms to be transmitted in baseband is the process of line coding. Since the most common type of line coding is for binary data, such a waveform can be succinctly termed a direct format for serial bits. The concentration in this section will be line coding for binary data.

Different channel characteristics, as well as different applications and performance requirements,

have provided the impetus for the development and study of various types of line coding [1, 2].

For example, the channel might be ac coupled and, thus, could not support a line code with a dc

component or large dc content. Synchronization or timing recovery requirements might necessitate a discrete component at the data rate. The channel bandwidth and crosstalk limitations might dictate the type of line coding employed. Even such factors as the complexity of the encoder and the economy of the decoder could determine the line code chosen. Each line code has its own distinct properties.

Depending on the application, one property may be more important than the other. In what follows, we describe, in general, the most desirable features that are considered when choosing a line code. It is commonly accepted [1, 2, 5, 8] that the dominant considerations effecting the choice of a line code are: 1) timing, 2) dc content, 3) power spectrum, 4) performance monitoring, 5) probability of error, and 6) transparency. Each of these are detailed in the following paragraphs:

1) **Timing**: The waveform produced by a line code should contain enough timing information

such that the receiver can synchronize with the transmitter and decode the received signal properly. The timing content should be relatively independent of source statistics, i.e., a long string of 1s or 0s should not result in loss of timing or jitter at the receiver.

2) **DC content**: Since the repeaters used in telephony are ac coupled, it is desirable to have zero

dc in the waveform produced by a given line code. If a signal with significant dc content is used

in ac coupled lines, it will cause dc wander in the received waveform. That is, the received signal

baseline will vary with time. Telephone lines do not pass dc due to ac coupling with transformers

and capacitors to eliminate dc ground loops. Because of this, the telephone channel causes a droop in constant signals. This causes dc wander. It can be eliminated by dc restoration circuits, feedback systems, or with specially designed line codes.

3) **Power spectrum**: The power spectrum and bandwidth of the transmitted signal should be

matched to the frequency response of the channel to avoid significant distortion. Also, the power

spectrum should be such that most of the energy is contained in as small bandwidth as possible. The smaller is the bandwidth, the higher is the transmission efficiency.

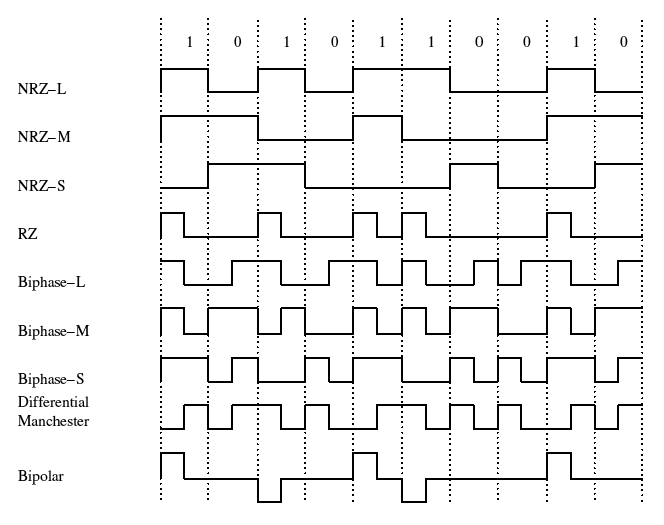
4) **Performance monitoring**: It is very desirable to detect errors caused by a noisy transmission

channel. The error detection capability in turn allows performance monitoring while the channel is in use (i.e., without elaborate testing procedures that require suspending use of the channel).

5) **Probability of error**: The average error probability should be as small as possible for a given

transmitter power. This reflects the reliability of the line code.

6) **Transparency**: A line code should allow all the possible patterns of 1s and 0s. If a certain pattern is undesirable due to other considerations, it should be mapped to a unique alternative pattern.

  
  
**SOFTWARE REQUIREMENTS**

* 1. **SOFTWARE USED**

What Is MATLAB?

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

* Math and computation
* Algorithm development
* Modeling, simulation, and prototyping
* Data analysis, exploration, and visualization
* Scientific and engineering graphics
* Application development, including Graphical User Interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non interactive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects, which together represent the state-of-the-art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

The MATLAB system consists of five main parts:

* The MATLAB language.

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

* The MATLAB working environment.

This is the set of tools and facilities that you work with as the MATLAB user or programmer. It includes facilities for managing the variables in your workspace and importing and exporting data. It also includes tools for developing, managing, debugging, and profiling M-files, MATLAB's applications.

* Handle Graphics.

This is the MATLAB graphics system. It includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level commands that allow you to fully customize the appearance of graphics as well as to build complete Graphical User Interfaces on your MATLAB applications.

* The MATLAB mathematical function library.

This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

* The MATLAB Application Program Interface (API).

This is a library that allows you to write C and Fortran programs that interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

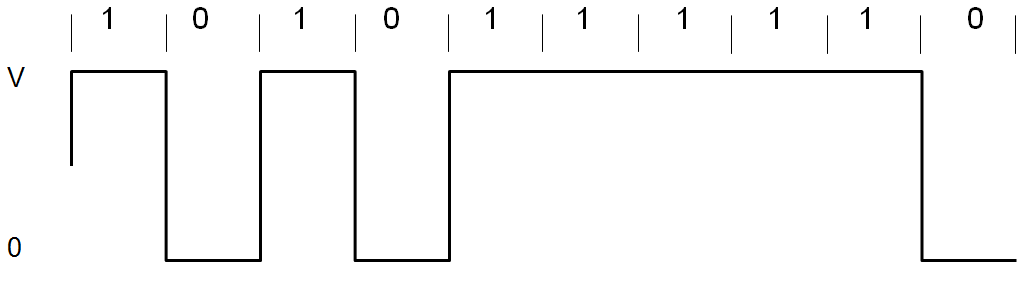
1. **TYPES OF LINE CODING**

* **Unipolar NRZ and RZ -** Unipolar encoding is a line code. A positive voltage represents a binary 1, and zero volts indicates a binary 0. It is the simplest line code, directly encoding the bitstream, and is analogous to on-off keying in modulation.

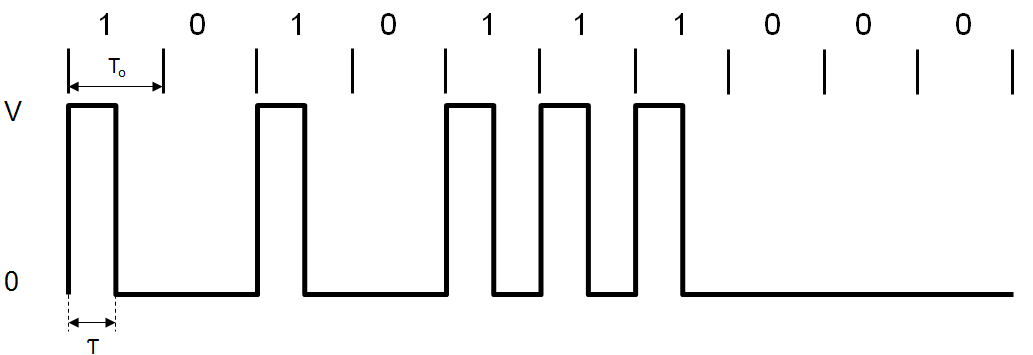
Its drawbacks are that it is not self-clocking and it has a significant DC component, which can be halved by using return-to-zero, where the signal returns to zero in the middle of the bit period. With a 50% duty cycle each rectangular pulse is only at a positive voltage for half of the bit period. This is ideal if one symbol is sent much more often than the other and power considerations are necessary, and also makes the signal self-clocking.

NRZ(Non-Return-to-Zero) - Traditionally, a unipolar scheme was designed as a non-return-to-zero (NRZ) scheme, in which the positive voltage defines bit 1 and the zero voltage defines bit 0. It is called NRZ because the signal does not return to zero at the middle of the bit, as instead happens in other line coding schemes, such as Manchester code. Compared with its polar counterpart, polar NRZ, this scheme applies a DC bias to the line and unnecessarily wastes power – The normalized power (power required to send 1 bit per unit line resistance) is double that for polar NRZ. For this reason, unipolar encoding is not normally used in data communications today.

RZ (return-to-zero) refers to a form of digital data transmission in which the binary low and high states, represented by numerals 0 and 1, are transmitted by voltage pulses having certain characteristics.

****

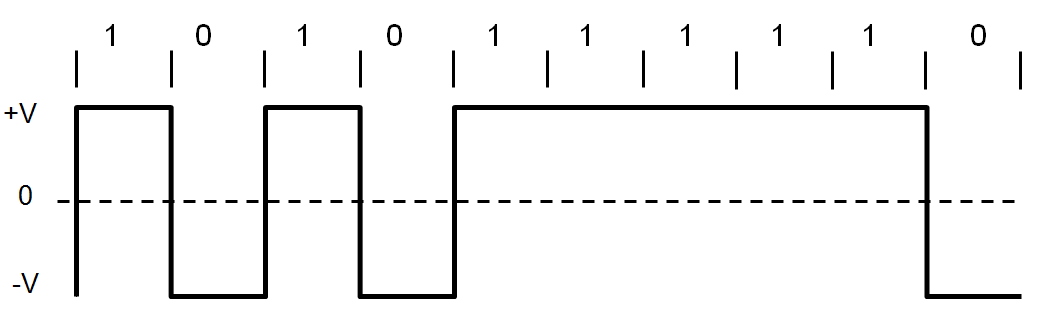
**Unipolar NRZ**

****

**Unipolar RZ**

* **Polar NRZ -** Polar NRZ

In this type of Polar signaling, a High in data is represented by a positive pulse, while a Low in data is represented by a negative pulse. The following figure depicts this well.



**Polar NRZ**

Advantages

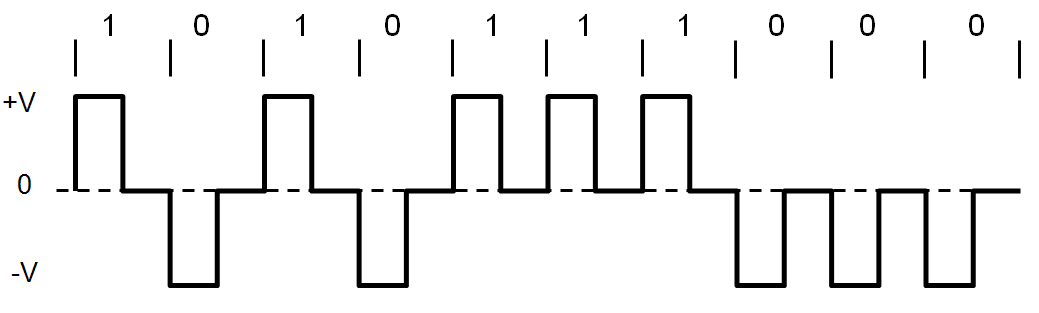
* The advantages of Polar NRZ are −
* It is simple.
* No low-frequency components are present.

Disadvantages

* The disadvantages of Polar NRZ are −
* No error correction.
* No clock is present.
* The signal droop is caused at the places where the signal is non-zero at 0 Hz..
* **Polar RZ -** Polar RZ

In this type of Polar signaling, a High in data, though represented by a Mark pulse, its duration T0 is less than the symbol bit duration. Half of the bit duration remains high but it immediately returns to zero and shows the absence of pulse during the remaining half of the bit duration.

However, for a Low input, a negative pulse represents the data, and the zero level remains same for the other half of the bit duration. The following figure depicts this clearly.



**Polar RZ**

Advantages

The advantages of Polar RZ are

* It is simple.
* No low-frequency components are present.

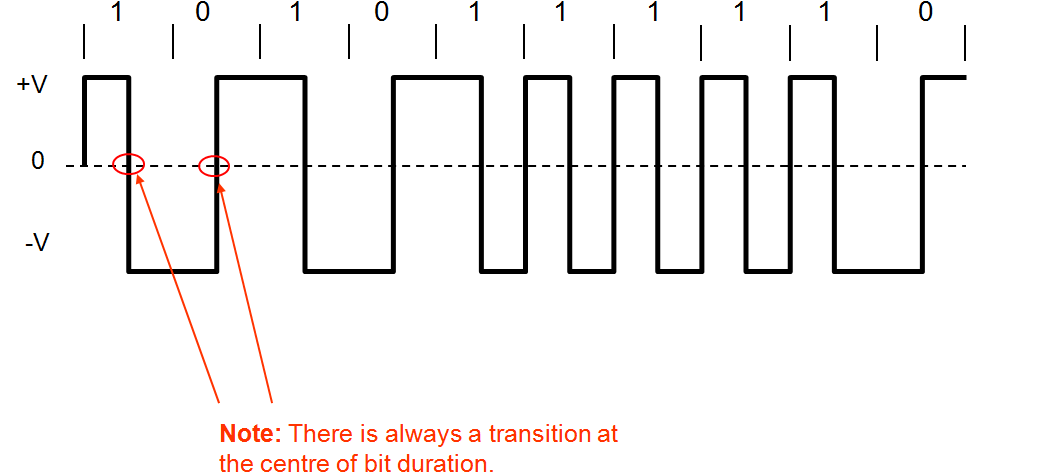
Disadvantages

The disadvantages of Polar RZ are −

* No error correction.
* No clock is present.
* Occupies twice the bandwidth of Polar NRZ.
* The signal droop is caused at places where the signal is non-zero at 0 Hz.
* **Manchester -**

Manchester encoding is a synchronous clock encoding technique used by the physical layer to encode the clock and data of a synchronous bit stream. In this technique, the actual binary data to be transmitted over the cable are not sent as a sequence of logic 1's and 0's (known technically as Non Return to Zero (NRZ)). Instead, the bits are translated into a slightly different format that has a number of advantages over using straight binary encoding (i.e. NRZ).

In the Manchester encoding shown, logic 0 is indicated by a 0 to 1 transition at the centre of the bit and logic 1 is indicated by a 1 to 0 transition at the centre of the bit. Note that signal transitions do not always occur at the ‘bit boundaries’ (the division between one bit and another), but that there is always a transition at the centre of each bit. The Manchester encoding rules are summarized below:



**Manchester**

* **Bipolar NRZ and RZ**

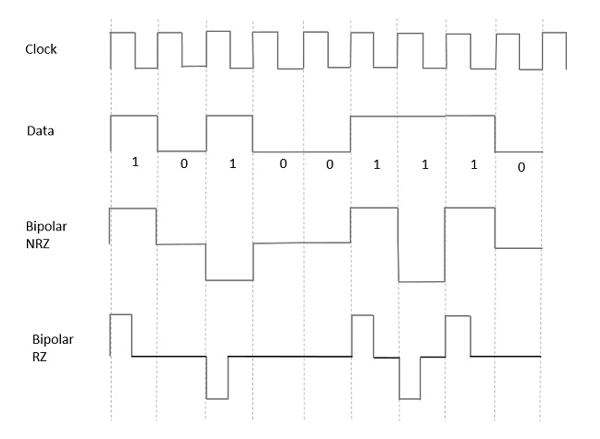
This is an encoding technique which has three voltage levels namely +, - and 0. Such a signal is called as duo-binary signal.

An example of this type is Alternate Mark Inversion (AMI). For a 1, the voltage level gets a transition from + to – or from – to +, having alternate 1s to be of equal polarity. A 0 will have a zero voltage level.

Even in this method, we have two types.

* Bipolar NRZ
* Bipolar RZ

From the models so far discussed, we have learnt the difference between NRZ and RZ. It just goes in the same way here too. The following figure clearly depicts this.



**Bipolar NRZ and RZ**

Advantages

Following are the advantages −

* It is simple.
* No low-frequency components are present.
* Occupies low bandwidth than unipolar and polar NRZ schemes.
* This technique is suitable for transmission over AC coupled lines, as signal drooping doesn’t occur here.
* A single error detection capability is present in this.

Disadvantages

* Following are the disadvantages −
* No clock is present.
* Long strings of data cause loss of synchronization.

1. **DATA COMPRESSION TECHNIQUES**

* **Shannon Fano Coding**

The Shannon-Fano technique has as an advantage its simplicity. The code is constructed as follows: the source messages a(i) and their probabilities p( a(i) ) are listed in order of non increasing probability. This list is then divided in such a way as to form two groups of as nearly equal total probabilities as possible. Each message in the first group receives 0 as the first digit of its codeword; the messages in the second half have code words beginning with 1. Each of these groups is then divided according to the same criterion and additional code digits are appended. The process is continued until each subset contains only one message. Clearly the Shannon-Fano algorithm yields a minimal prefix code.

a 1/2 0

b 1/4 10

c 1/8 110

d 1/16 1110

e 1/32 11110

f 1/32 11111

**A Shannon-Fano Code.**

The above figure shows the application of the method to a particularly simple probability distribution. The length of each codeword x is equal to -lg p(x). This is true as long as it is possible to divide the list into subgroups of exactly equal probability. When this is not possible, some code words may be of length -lg p(x)+1. The Shannon-Fano algorithm yields an average codeword length S which satisfies H <= S <= H + 1. In Figure 3.2, the Shannon-Fano code for ensemble EXAMPLE is given. As is often the case, the average codeword length is the same as that achieved by the Huffman code (see Figure 1.3). That the Shannon-Fano algorithm is not guaranteed to produce an optimal code is demonstrated by the following set of probabilities: { .35, .17, .17, .16, .15 }.

g 8/40 00

f 7/40 010

e 6/40 011

d 5/40 100

space 5/40 101

c 4/40 110

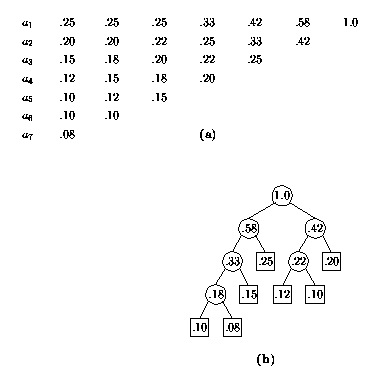
b 3/40 1110

a 2/40 1111

**A Shannon-Fano Code for EXAMPLE** (code length=117).

* **Huffman Coding**

Huffman's algorithm, expressed graphically, takes as input a list of nonnegative weights {w(1), ... ,w(n) } and constructs a full binary tree [a binary tree is full if every node has either zero or two children] whose leaves are labeled with the weights. When the Huffman algorithm is used to construct a code, the weights represent the probabilities associated with the source letters. Initially there is a set of singleton trees, one for each weight in the list. At each step in the algorithm the trees corresponding to the two smallest weights, w(i) and w(j), are merged into a new tree whose weight is w(i)+w(j) and whose root has two children which are the sub trees represented by w(i) and w(j). The weights w(i) and w(j) are removed from the list and w(i)+w(j) is inserted into the list. This process continues until the weight list contains a single value. If, at any time, there is more than one way to choose a smallest pair of weights, any such pair may be chosen. In Huffman's paper, the process begins with a non increasing list of weights. This detail is not important to the correctness of the algorithm, but it does provide a more efficient implementation . The Huffman algorithm is demonstrated in Figure



**The Huffman process: (a) The list; (b) the tree.**

**APPENDIX A**

**A.1 Code for LINE CODE PLOT**

LINE CODING

% Demo of using different line codings

bits = [1 0 1 0 0 0 1 1 0];

bitrate = 1; % bits per second

%Function of Unipolar Non – Return to Zero (UNRZ)

figure;

[t,s] = unrz(bits,bitrate);

plot(t,s,'LineWidth',3);

axis([0 t(end) -0.1 1.1])

grid on;

title(['Unipolar NRZ: [' num2str(bits) ']']);

%Function of Unipolar Return to Zero (URZ)

figure;

[t,s] = urz(bits,bitrate);

plot(t,s,'LineWidth',3);

axis([0 t(end) -0.1 1.1])

grid on;

title(['Unipolar RZ: [' num2str(bits) ']']);

%Function of Polar Return to Zero (PRZ)

figure;

[t,s] = prz(bits,bitrate);

plot(t,s,'LineWidth',3);

axis([0 t(end) -1.1 1.1])

grid on;

title(['Polar RZ: [' num2str(bits) ']']);

%Function of Manchester coding

figure;

[t,s] = manchester(bits,bitrate);

plot(t,s,'LineWidth',3);

axis([0 t(end) -1.1 1.1])

grid on;

title(['Manchester: [' num2str(bits) ']']);

**Unipolar Non- Return to Zero (UNRZ)**

function [t,x] = unrz(bits, bitrate)

% UNRZ Encode bit string using unipolar NRZ code.

% [T, X] = UNRZ(BITS, BITRATE) encodes BITS array using unipolar NRZ

% code with given BITRATE. Outputs are time T and encoded signal

% values X.

T = length(bits)/bitrate; % full time of bit sequence

n = 200;

N = n\*length(bits);

dt = T/N;

display(dt);

t = 0:dt:T;

x = zeros(1,length(t)); % output signal

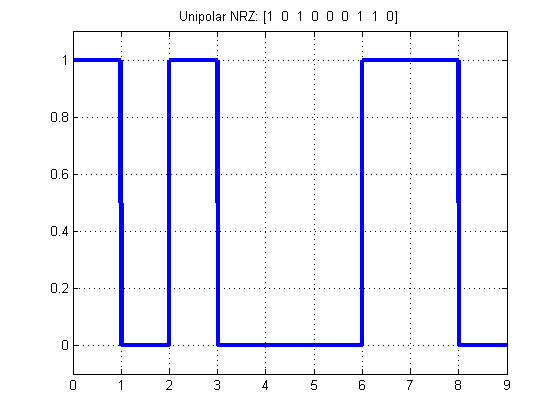
for i = 0:length(bits)-1

if bits(i+1) == 1

x(i\*n+1:(i+1)\*n) = 1;

else

x(i\*n+1:(i+1)\*n) = 0;

 end

end

**Unipolar Return to Zero (URZ)**

function [t,x] = urz(bits, bitrate)

% URZ Encode bit string using unipolar RZ code.

% [T, X] = URZ(BITS, BITRATE) encodes BITS array using unipolar RZ

% code with given BITRATE. Outputs are time T and encoded signal

% values X.

T = length(bits)/bitrate; % full time of bit sequence

n = 200;

N = n\*length(bits);

dt = T/N;

t = 0:dt:T;

x = zeros(1,length(t)); % output signal

for i = 0:length(bits)-1

if bits(i+1) == 1

x(i\*n+1:(i+0.5)\*n) = 1;

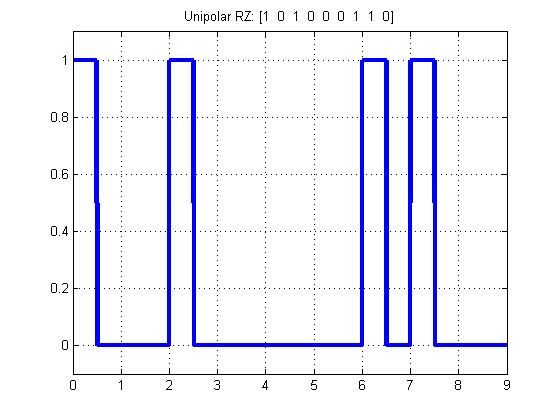
x((i+0.5)\*n+1:(i+1)\*n) = 0;

else

x(i\*n+1:(i+1)\*n) = 0;

end

end



**Polar Return to Zero (URZ)**

function [t,x] = prz(bits, bitrate)

% PRZ Encode bit string using polar RZ code.

% [T, X] = PRZ(BITS, BITRATE) encodes BITS array using polar RZ

% code with given BITRATE. Outputs are time T and encoded signal

% values X..

T = length(bits)/bitrate; % full time of bit sequence

n = 200;

N = n\*length(bits);

dt = T/N;

t = 0:dt:T;

x = zeros(1,length(t)); % output signal

for i = 0:length(bits)-1

if bits(i+1) == 1

x(i\*n+1:(i+0.5)\*n) = 1;

x((i+0.5)\*n+1:(i+1)\*n) = 0;

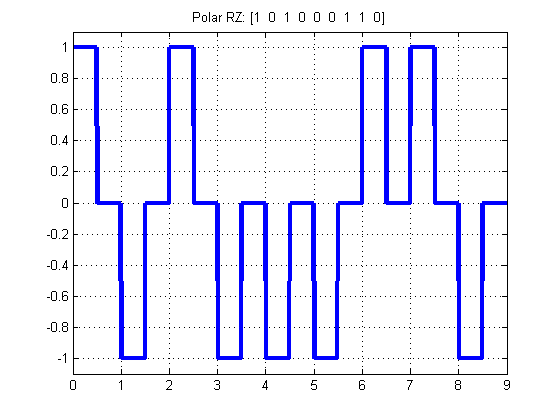
else

x(i\*n+1:(i+0.5)\*n) = -1;

x((i+0.5)\*n+1:(i+1)\*n) = 0;

end

end



**Polar Return to Zero (URZ)**

function [t,x] = manchester(bits, bitrate)

% MANCHESTER Encode bit string using Manchester code.

% [T, X] = MANCHESTER(BITS, BITRATE) encodes BITS array using Manchester

% code with given BITRATE. Outputs are time T and encoded signal

% values X.

T = length(bits)/bitrate; % full time of bit sequence

n = 200;

N = n\*length(bits);

dt = T/N;

t = 0:dt:T;

x = zeros(1,length(t)); % output signal

for i = 0:length(bits)-1

if bits(i+1) == 1

x(i\*n+1:(i+0.5)\*n) = 1;

x((i+0.5)\*n+1:(i+1)\*n) = -1;

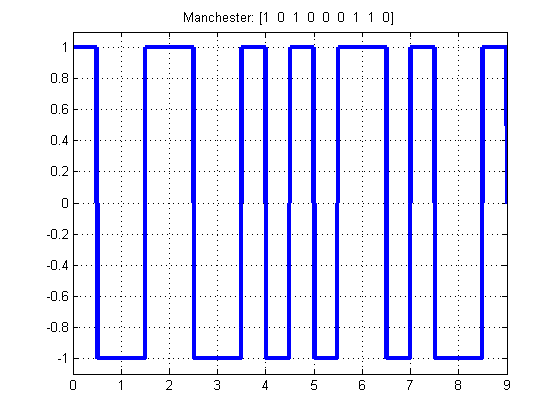
else

x(i\*n+1:(i+0.5)\*n) = -1;

x((i+0.5)\*n+1:(i+1)\*n) = 1;

end

end



**A.2 Code for SHANNON CODING**

clc;

clear all;

close all;

m=input('Enter the no. of message ensembles : ');

z=[];

h=0;l=0;

display('Enter the probabilities in descending order');

for i=1:m

fprintf('Ensemble %d\n',i);

p(i)=input('');

end

%Finding each alpha values

a(1)=0;

for j=2:m;

a(j)=a(j-1)+p(j-1);

end

fprintf('\n Alpha Matrix');

display(a);

%Finding each code length

for i=1:m

if(p(i) > 0.02)

n(i)= round(-1\*(log2(p(i))));

elseif (p(i) <= 0.02)

n(i) = 5;

elseif (p(i) <= 0.01)

n(i) = 6;

end

end

fprintf('\n Code length matrix');

display(n);

%Computing each code

for i=1:m

int=a(i);

for j=1:n(i)

frac=int\*2;

c=floor(frac);

frac=frac-c;

z=[z c];

int=frac;

end

fprintf('Codeword %d',i);

display(z);

z=[];

end

%Computing Avg. Code Length & Entropy

fprintf('Avg. Code Length');

for i=1:m

x=p(i)\*n(i);

l=l+x;

x=p(i)\*log2(1/p(i));

h=h+x;

end

display(l);

fprintf('Entropy');

display(h);

%Computing Efficiency

fprintf('Efficiency');

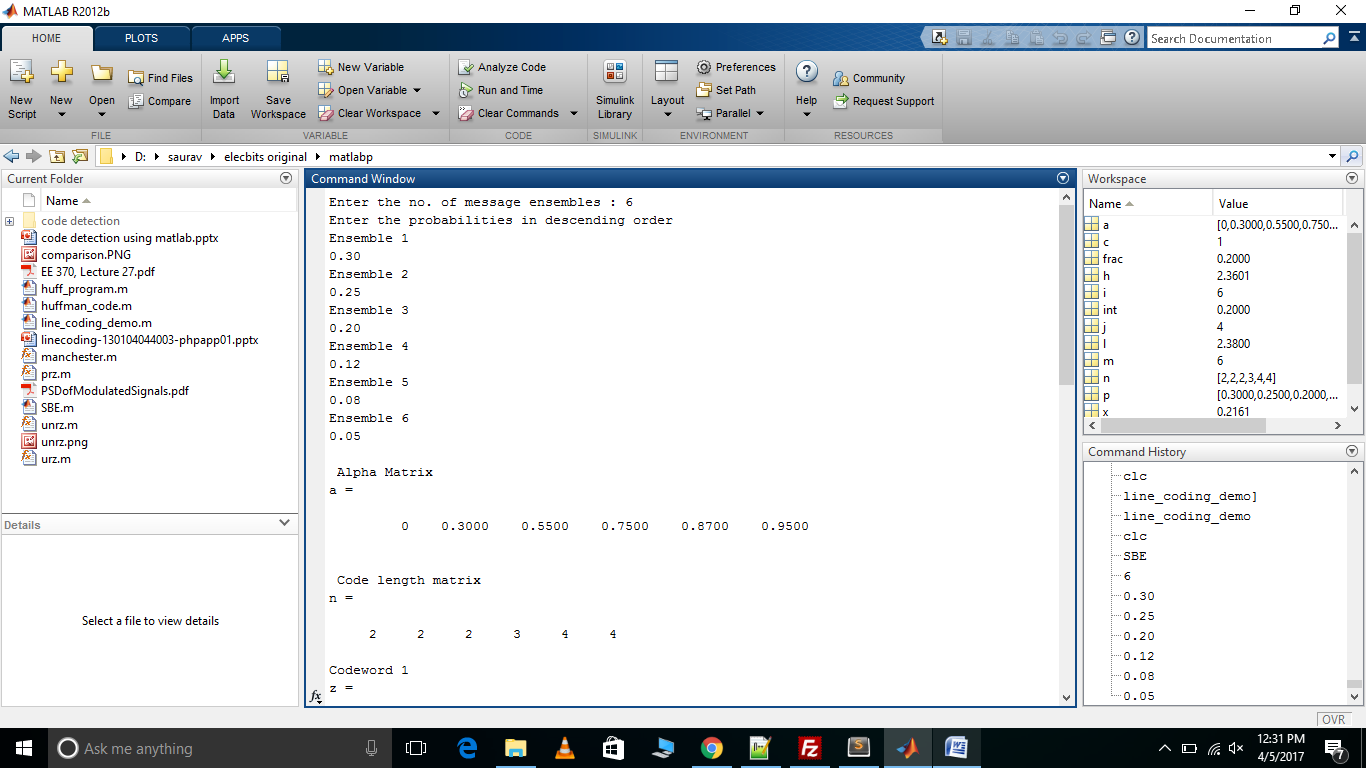
display(100\*h/l);

fprintf('Redundancy');

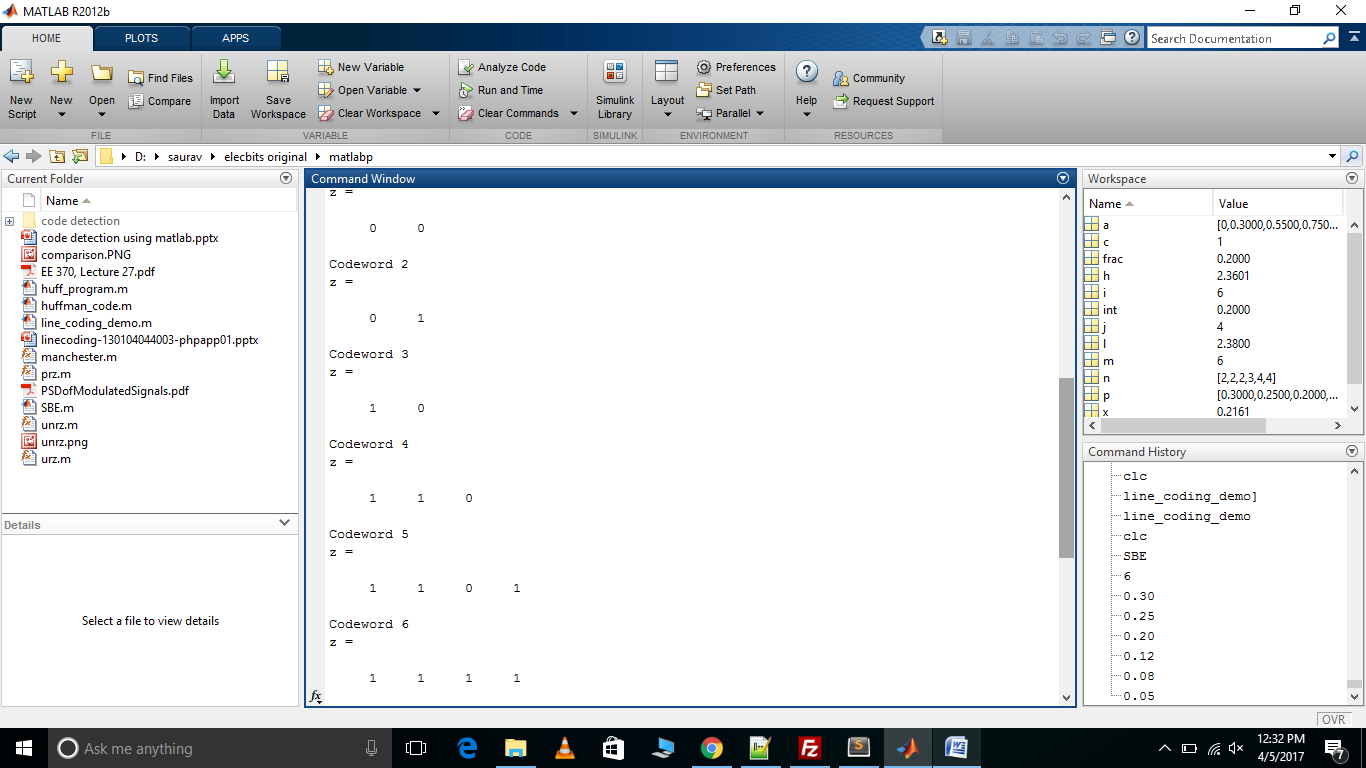
display(100-(100\*h/l));

OUTPUTS –

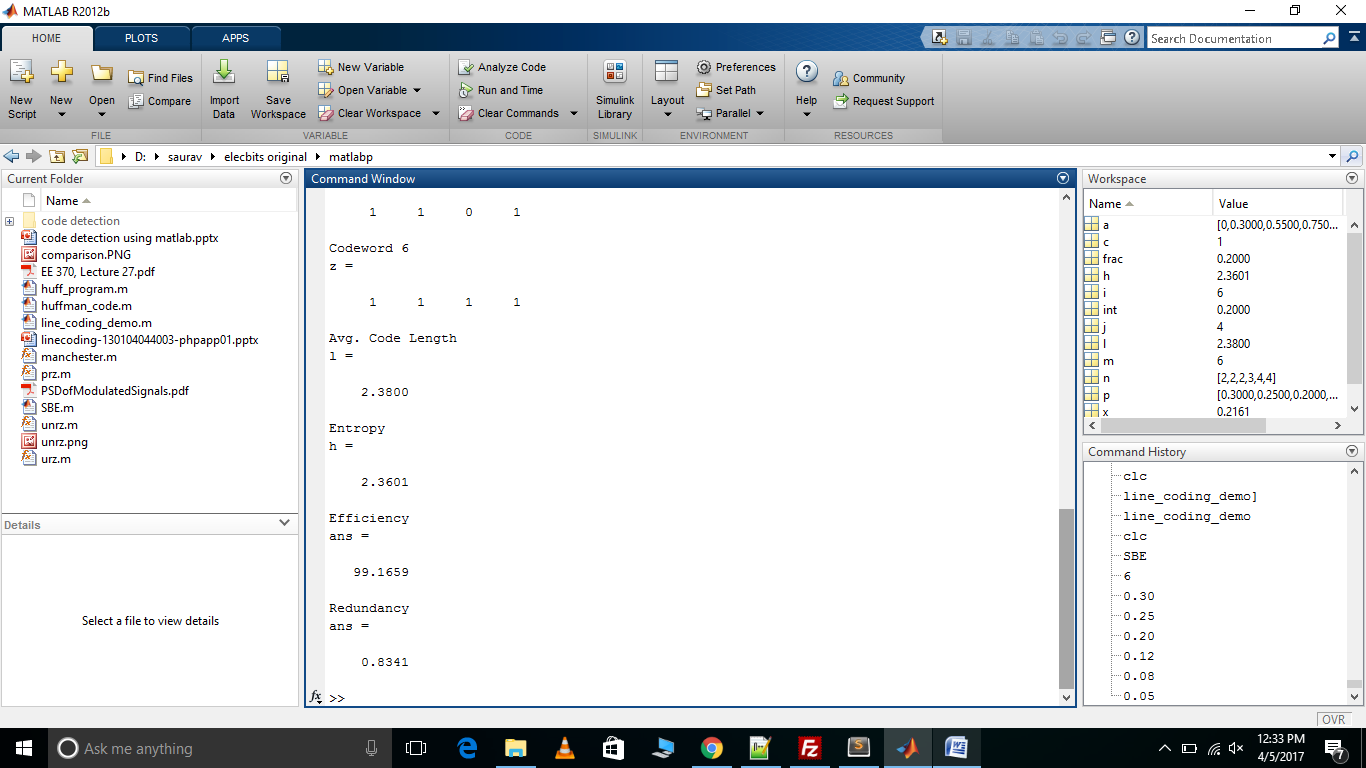
**Providing the probability-**



**Codewords -**



**Entropy, Efficiency, Redundancy and Average Code length**



**A.3 Code for the HUFFMAN CODING**

clc;

p=input('Enter the probabilities:');

n=length(p);

symbols=[1:n];

[dict,avglen]=huffmandict(symbols,p);

temp=dict;

t=dict(:,2);

for i=1:length(temp)

temp{i,2}=num2str(temp{i,2});

end

disp('The huffman code dict:');

disp(temp)

fprintf('Enter the symbols between 1 to %d in[]',n);

sym=input(':')

encod=huffmanenco(sym,dict);

disp('The encoded output:');

disp(encod);

bits=input('Enter the bit stream in[];');

decod=huffmandeco(bits,dict);

disp('The symbols are:');

disp(decod);

H=0;

Z=0;

for(k=1:n)

H=H+(p(k)\*log2(1/p(k)));

end

fprintf(1,'Entropy is %f bits',H);

N=H/avglen;

fprintf('\n Efficiency is:%f',N);

for(r=1:n)

l(r)=length(t{r});

end

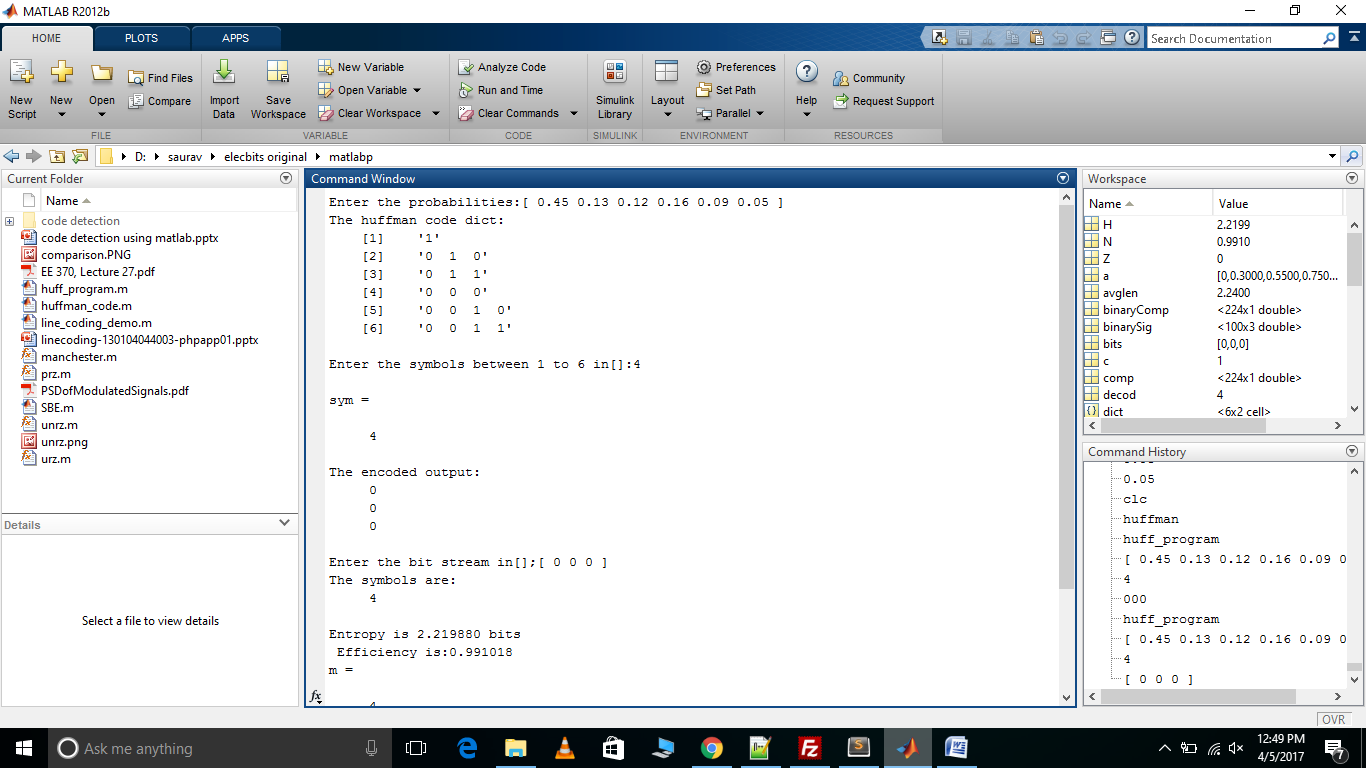
m=max(l)

s=min(l)

v=m-s;

fprintf('the variance is:%d',v);

**ENTROPY, EFFICIENCY AND CODEWORD**

****

**A.4 Code for LINE CODING PSD**

PSD ILLUSTRATION CODING

inputBits=randi([0 1],1,1000); % R2008a syntax =&gt; randsrc(1,10000,[0 1]);

colordef white;

Rb=2e6; % Bit Rate

amplitude=2; % Peak-Peak Amplitude

[time,xiPolar,Fs]=Line\_Encoder(inputBits,Rb,amplitude,'polar');

[pyy,fy]=psd(xiPolar);

figure(1);

plotHandle=plot(fy\*Fs/2,10\*log10((pyy)),'k');

set(plotHandle,'LineWidth',2.5);

hold on;

[time,xiUnipolar,Fs]=Line\_Encoder(inputBits,Rb,amplitude,'unipolar');

[pyy,fy]=psd(xiUnipolar);

plotHandle=plot(fy\*Fs/2,10\*log10((pyy)),'r');

set(plotHandle,'LineWidth',2.5);

hold on;

[time,xiManchester,Fs]=Line\_Encoder(inputBits,Rb,amplitude,'Manchester');

[pyy,fy]=psd(xiManchester);

plotHandle=plot(fy\*Fs/2,10\*log10((pyy)),'g');

set(plotHandle,'LineWidth',2.5);

legend('polar','unipolar','Manchester');

title('PSD of Line Codes');

grid on;

hold off;

figure(2);

subplot(4,1,1);

plot(inputBits);

xlabel('Time (Seconds)');

ylabel('Amplitude (Voltage)');

title('Input bit stream');

maxTime=max(time);

maxAmp=max(inputBits);

minAmp=min(inputBits);

axis([0,1000,minAmp-0.5,maxAmp+0.5]);

subplot(4,1,2);

plot(time,xiPolar);

xlabel('Time (Seconds)');

ylabel('Amplitude (Voltage)');

title('Polar NRZ coded');

maxTime=max(time);

maxAmp=max(xiPolar);

minAmp=min(xiPolar);

axis([0,maxTime,minAmp-0.5,maxAmp+0.5]);

subplot(4,1,3);

plot(time,xiUnipolar);

xlabel('Time (Seconds)');

ylabel('Amplitude (Voltage)');

title('Unipolar NRZ coded');

maxTime=max(time);

maxAmp=max(xiUnipolar);

minAmp=min(xiUnipolar);

axis([0,maxTime,minAmp-0.5,maxAmp+0.5]);

subplot(4,1,4);

plot(time,xiManchester);

xlabel('Time (Seconds)');

ylabel('Amplitude (Voltage)');

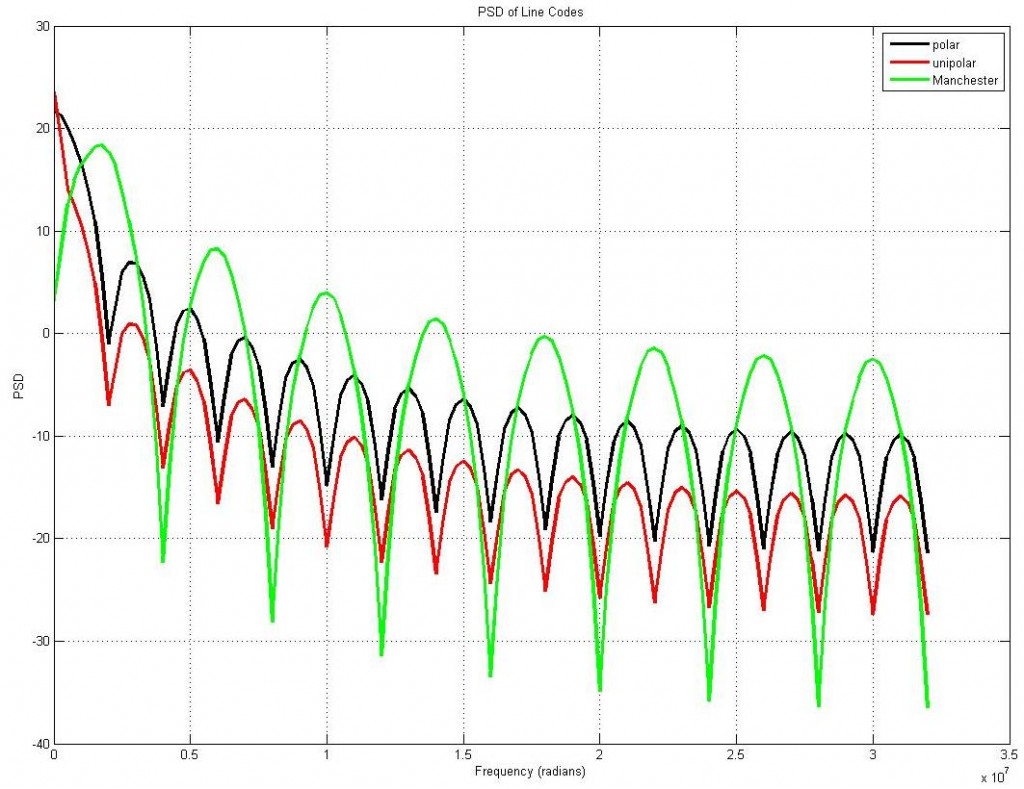
title('Manchester Coded');

maxTime=max(time);

maxAmp=max(xiManchester);

minAmp=min(xiManchester);

axis([0,maxTime,minAmp-0.5,maxAmp+0.5]);



**REFERENCES**

**Books:**

* **Matlab for Beginners: A Gentle Approach**

**Web URLs:**

* <http://ieeexplore.ieee.org/document/4909180/>
* <http://ieeexplore.ieee.org/document/6072008/>