Scilab Code for Digital Communication, by Simon Haykin [1]

Created by
Prof. R. Senthilkumar
Institute of Road and Transport Technology
rsenthil_signalprocess@in.com

Cross-Checked by
Prof. Saravanan Vijayakumaran, IIT Bombay
sarva@ee.iitb.ac.in

23 August 2010

[1]Funded by a grant from the National Mission on Education through ICT, http://spoken-tutorial.org/NMEICT-Intro. This Text Book Companion and Scilab codes written in it can be downlaoded from the website www.scilab.in

Book Details

Authors: Simon Haykins

Title: Digital Communication

Publisher: Willey India

Edition: Wiley India Edition

Year: Reprint 2010

Place: Delhi

ISBN: 9788126508242

Scilab numbering policy used in this document and the relation to the above book.

Prb Problem (Unsolved problem)

Exa Example (Solved example)

Tab Table

ARC Additionally Required Code (Scilab Code that is not part of the above book but required to solve a particular Example)

AE Appendix to Example(Scilab Code that is an Appednix to a particular Example of the above book)

CF Code for Figure(Scilab code that is used for plotting the respective figure of the above book)

For example, Prb 4.56 means Problem 4.56 of the above book. Exa 3.51 means solved example 3.51 of this book. Sec 2.3 means a scilab code whose theory is explained in Section 2.3 of the book.

Contents

List of Scilab Codes

List of Figures

Chapter 1

Introduction

Scilab code CF 1.2 Digital Representation of Analog signal

```
//Caption:Digital Representation of Analog signal
//Figure 1.2: Analog to Digital Conversion
clear;
close;
clc;
t = -1:0.01:1;
x = 2*sin((%pi/2)*t);
dig_data = [0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,1,0,1]
//
figure
a=gca();
a.x_location ="origin";
a.y_location ="origin";
a.data_bounds =[-2,-3;2,3]
plot(t,x)
plot2d3('gnn',0.5,sqrt(2),-9)
plot2d3('gnn',-0.5,-sqrt(2),-9)
plot2d3('gnn',1,2,-9)
plot2d3('gnn',-1,-2,-9)
xlabel('
   Time')
ylabel('
```

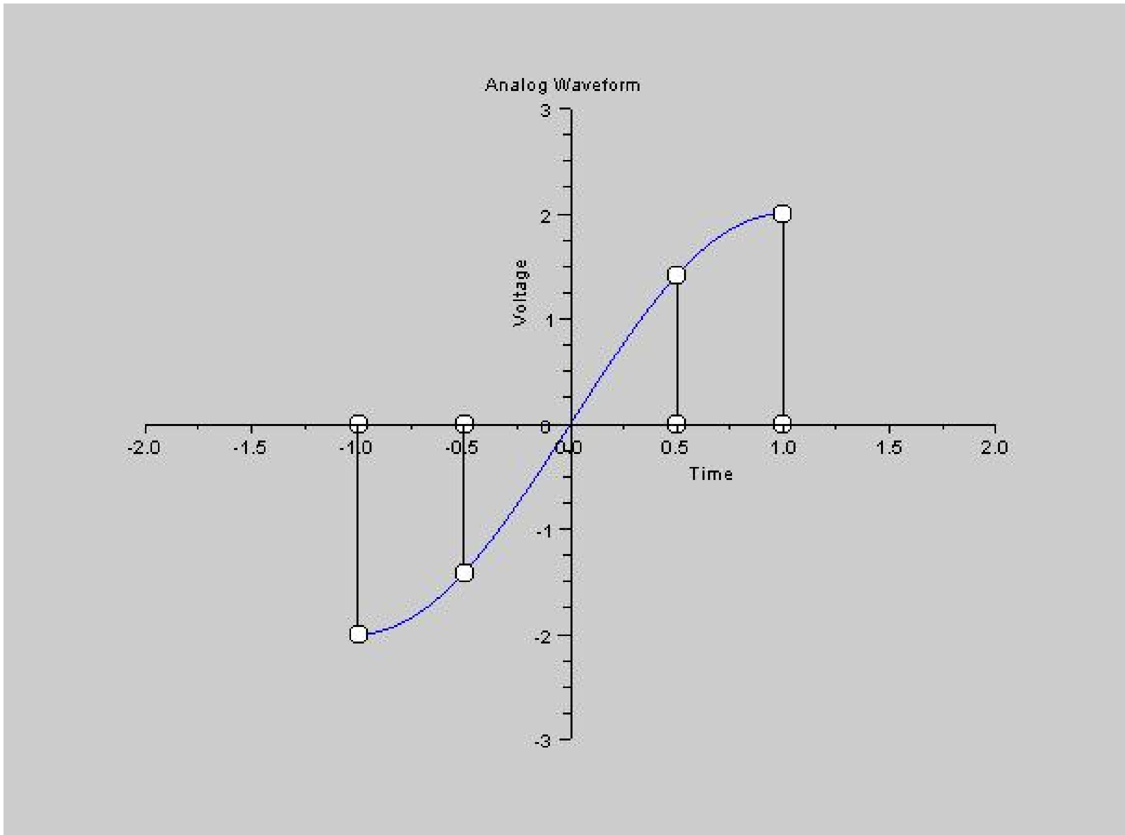

Figure 1.1: Figure1.2a

```
   Voltage')
title('Analog Waveform')
//
figure
a = gca();
a.data_bounds = [0,0;21,5];
plot2d2([1:length(dig_data)],dig_data,5)
title('Digital Representation')
```

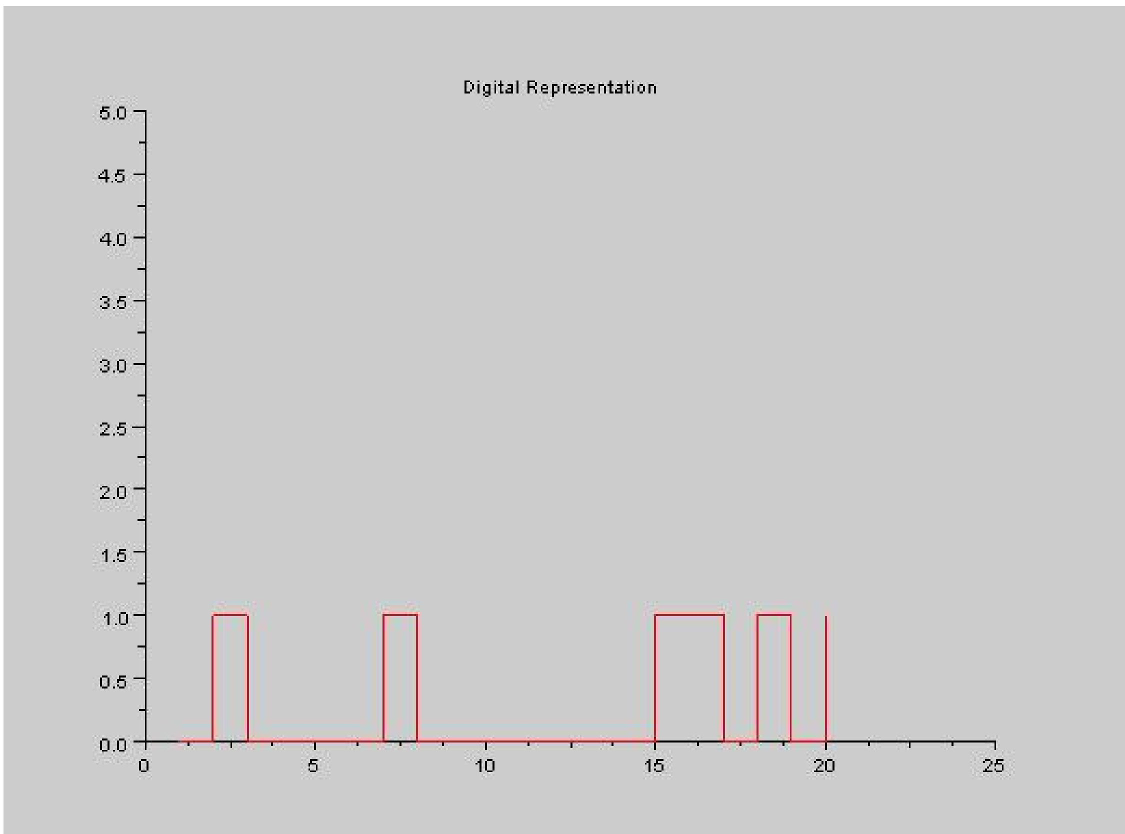

Figure 1.2: Figure1.2b

Chapter 2

Fundamental Limit on Performance

Scilab code Exa 2.1 Entropy of Binary Memoryless source

```
//Caption:Entropy of Binary Memoryless source
//Example 2.1: Entropy of Binary Memoryless Source
//page 18
clear;
close;
clc;
Po = 0:0.01:1;
H_Po = zeros(1,length(Po));
for i = 2:length(Po)-1
  H_Po(i) = -Po(i)*log2(Po(i))-(1-Po(i))*log2(1-Po(i));
end
//plot
plot2d(Po,H_Po)
xlabel('Symbol Probability, Po')
ylabel('H(Po)')
title('Entropy function H(Po)')
plot2d3('gnn',0.5,1)
```

Scilab code Exa 2.2 Second order Extension of Discrete Memoryless Source

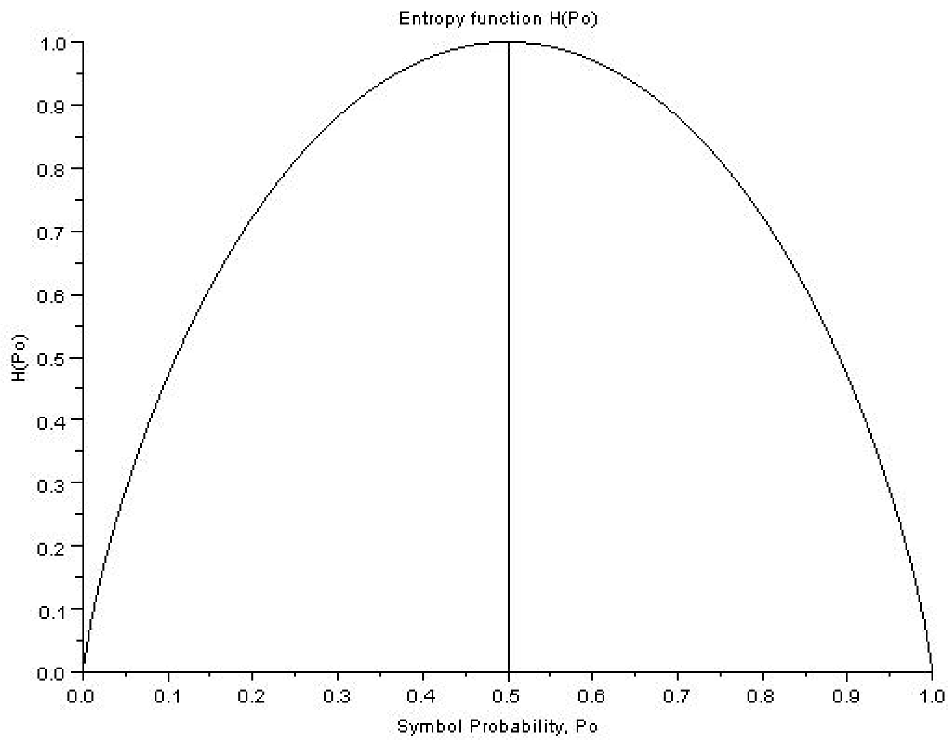

Figure 2.1: Example2.1

```
//caption:Second order Extension of Discrete Memoryless Source
//Example 2.2:Entropy of Discrete Memoryless source
//page 19
clear;
clc;
P0 = 1/4; //probability of source alphabet S0
P1 = 1/4; //probability of source alphabet S1
P2 = 1/2; //probability of source alphabet S2
H_Ruo = P0*log2(1/P0)+P1*log2(1/P1)+P2*log2(1/P2);
disp('Entropy of Discrete Memoryless Source')
disp('bits',H_Ruo)
//Second order Extension of discrete Memoryless source
P_sigma = [P0*P0,P0*P1,P0*P2,P1*P0,P1*P1,P1*P2,P2*P0,P2*P1,P2*P2];
disp('Table 2.1 Alphabet Particulars of Second-order Extension of a Discrete Memoryless Source')
disp('-------------------------------------------------------------')
disp('Sequence of Symbols of ruo2:')
disp('  S0*S0     S0*S1     S0*S2     S1*S0     S1*S1     S1*S2     S2*S0     S2*S1   S2*S2')
disp(P_sigma,'Probability p(sigma), i =0,1.....8')
disp('-------------------------------------------------------------')
disp('   ')
H_Ruo_Square =0;
for i = 1:length(P_sigma)
  H_Ruo_Square = H_Ruo_Square+P_sigma(i)*log2(1/P_sigma(i));
end
disp('bits', H_Ruo_Square,'H(Ruo_Square)=')
disp('H(Ruo_Square) = 2*H(Ruo)')
```

Scilab code Exa 2.3 Entropy,Average length, Variance of Huffman Encoding

```
//Caption:Entropy, Average length, Variance of
   Huffman Encoding
//Example 2.3: Huffman Encoding: Calculation of
// (a)Average code-word length 'L'
//(b)Entropy 'H'
clear;
clc;
P0 = 0.4; //probability of codeword '00'
L0 = 2;   //length of codeword S0
P1 = 0.2; //probability of codeword '10'
L1 = 2;   //length of codeword S1
P2 = 0.2; //probility of codeword '11'
L2 = 2;   //length of codeword S2
P3 = 0.1; //probility of codeword '010'
L3 = 3;   //length of codeword S3
P4 =0.1; //probility of codeword '011'
L4 = 3;   //length of codeword S4
L = P0*L0+P1*L1+P2*L2+P3*L3+P4*L4;
H_Ruo = P0*log2(1/P0)+P1*log2(1/P1)+P2*log2(1/P2)+P3
   *log2(1/P3)+P4*log2(1/P4);
disp('bits',L,'Average code-word Length L')
disp('bits',H_Ruo,'Entropy of Huffman coding result
   H')
disp('percent',((L-H_Ruo)/H_Ruo)*100,'Average code-
   word length L exceeds the entropy H(Ruo) by only'
   )
sigma_1 = P0*(L0-L)^2+P1*(L1-L)^2+P2*(L2-L)^2+P3*(L3
   -L)^2+P4*(L4-L)^2;
disp(sigma_1,'Varinace of Huffman code')
```

Scilab code Exa 2.4 Entropy, Average length, Variance of Huffman Encoding

```
//Caption:Entropy, Average length, Variance of
   Huffman Encoding
```

```
//Example2.4: Illustrating nonuniquess of the Huffman Encoding
// Calculation of (a)Average code-word length 'L' (b)Entropy 'H'
clear;
clc;
P0 = 0.4; //probability of codeword '1'
L0 = 1;   //length of codeword S0
P1 = 0.2; //probability of codeword '01'
L1 = 2;   //length of codeword S1
P2 = 0.2; //probility of codeword '000'
L2 = 3;   //length of codeword S2
P3 = 0.1; //probility of codeword '0010'
L3 = 4;   //length of codeword S3
P4 =0.1; //probility of codeword '0011'
L4 = 4;   //length of codeword S4
L = P0*L0+P1*L1+P2*L2+P3*L3+P4*L4;
H_Ruo = P0*log2(1/P0)+P1*log2(1/P1)+P2*log2(1/P2)+P3*log2(1/P3)+P4*log2(1/P4);
disp('bits',L,'Average code-word Length L')
disp('bits',H_Ruo,'Entropy of Huffman coding result H')
sigma_2 = P0*(L0-L)^2+P1*(L1-L)^2+P2*(L2-L)^2+P3*(L3-L)^2+P4*(L4-L)^2;
disp(sigma_2,'Varinace of Huffman code')
```

Scilab code Exa 2.5 Binary Symmetric Channel

```
//Caption:Binary Symmetric Channel
//Example2.5: Binary Symmetric Channel
clear;
clc;
close;
p = 0.4; //probability of correct reception
pe = 1-p;//probility of error reception (i.e) transition probility
disp(p,'probility of 0 receiving if a 0 is sent = probility of 1 receiving if a 1 is sent=')
```

```
disp('Transition probility')
disp(pe,'probility of 0 receiving if a 1 is sent = probility of 1 receiving if a 0 is sent=')
```

Scilab code Exa 2.6 Channel Capacity of a Binary Symmetric Channel

```
//Caption:Channel Capacity of a Binary Symmetric Channel
//Example2.6:Channel Capacity of Binary Symmetri Channel
clear;
close;
clc;
p = 0:0.01:0.5;
for i =1:length(p)
  if(i~=1)
    C(i) = 1+p(i)*log2(p(i))+(1-p(i))*log2((1-p(i)));
  elseif(i==1)
    C(i) =1;
  elseif(i==length(p))
    C(i)=0;
  end
end
plot2d(p,C,5)
xlabel('Transition Probility, p')
ylabel('Channel Capacity, C')
title('Figure 2.10 Variation of channel capacity of a binary symmetric channel with transition probility p')
```

Scilab code Exa 2.7 Significance of the Channel Coding theorem

```
//Caption:Significance of the Channel Coding theorem
//Example2.7: Significance of the channel coding theorem
//Average Probility of Error of Repetition Code
```

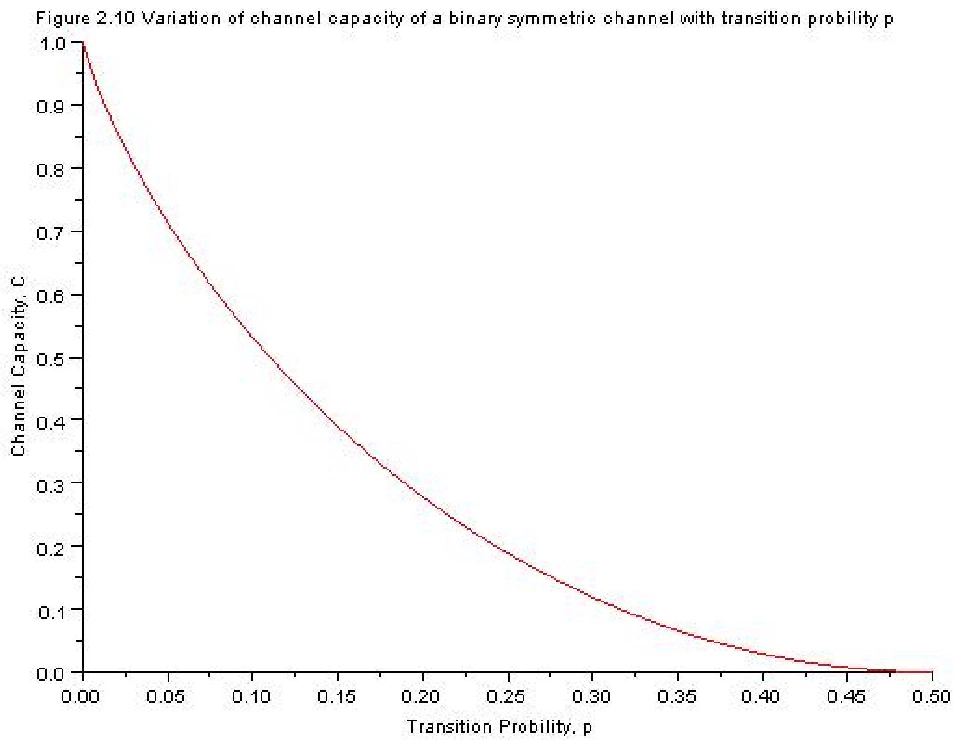

Figure 2.2: Example2.6

```
clear;
clc;
close;
p =10^-2;
pe_1 =p; //Average Probility of error for code rate r = 1
pe_3 = 3*p^2*(1-p)+p^3;//probility of error for code rate r =1/3
pe_5 = 10*p^3*(1-p)^2+5*p^4*(1-p)+p^5;//error for code rate r =1/5
pe_7 = ((7*6*5)/(1*2*3))*p^4*(1-p)^3+(42/2)*p^5*(1-p)^2+7*p^6*(1-p)+p^7;//error for code rate r =1/7
r = [1,1/3,1/5,1/7];
pe = [pe_1,pe_3,pe_5,pe_7];
a=gca();
a.data_bounds=[0,0;1,0.01];
plot2d(r,pe,5)
xlabel('Code rate, r')
ylabel('Average Probability of error, Pe')
title('Figure 2.12 Illustrating significance of the channel coding theorem')
legend('Repetition codes')
xgrid(1)
disp('Table 2.3 Average Probility of Error for Repetition Code')
disp('-----------------------------------------------------')
disp(r,'Code Rate, r =1/n',pe,'Average Probility of Error, Pe')
disp('-----------------------------------------------------')
```

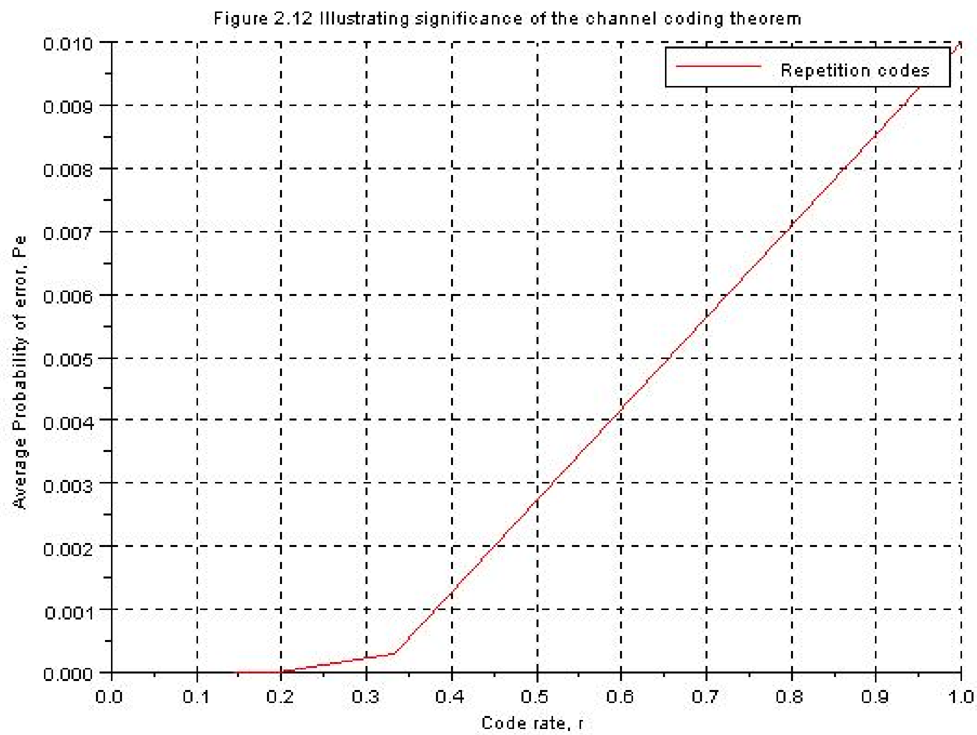

Figure 2.3: Example2.7

Chapter 3

Detection and Estimation

Scilab code Exa 3.1 Orthonormal basis for given set of signals

```
//Caption:Orthonormal basis for given set of signals
//Example3.1:Finding orthonormal basis for the given signals
//using Gram-Schmidt orthogonalization procedure
clear;
close;
clc;
T = 1;
t1 = 0:0.01:T/3;
t2 = 0:0.01:2*T/3;
t3 = T/3:0.01:T;
t4 = 0:0.01:T;
s1t = [0,ones(1,length(t1)-2),0];
s2t = [0,ones(1,length(t2)-2),0];
s3t = [0,ones(1,length(t3)-2),0];
s4t = [0,ones(1,length(t4)-2),0];
t5 = 0:0.01:T/3;
phi1t =  sqrt(3/T)*[0,ones(1,length(t5)-2),0];
t6 =T/3:0.01:2*T/3;
phi2t = sqrt(3/T)*[0,ones(1,length(t6)-2),0];
t7 = 2*T/3:0.01:T;
phi3t = sqrt(3/T)*[0,ones(1,length(t7)-2),0];
//
```

```
figure
title('Figure3.4(a) Set of signals to be
   orthonormalized')
subplot(4,1,1)
a =gca();
a.data_bounds = [0,0;2,2];
plot2d2(t1,s1t,5)
xlabel('t')
ylabel('s1(t)')
subplot(4,1,2)
a =gca();
a.data_bounds = [0,0;2,2];
plot2d2(t2,s2t,5)
xlabel('t')
ylabel('s2(t)')
subplot(4,1,3)
a =gca();
a.data_bounds = [0,0;2,2];
plot2d2(t3,s3t,5)
xlabel('t')
ylabel('s3(t)')
subplot(4,1,4)
a =gca();
a.data_bounds = [0,0;2,2];
plot2d2(t4,s4t,5)
xlabel('t')
ylabel('s4(t)')
//
figure
title('Figure3.4(b) The resulting set of orthonormal
    functions')
subplot(3,1,1)
a =gca();
a.data_bounds = [0,0;2,4];
plot2d2(t5,phi1t,5)
xlabel('t')
ylabel('phi1(t)')
subplot(3,1,2)
```

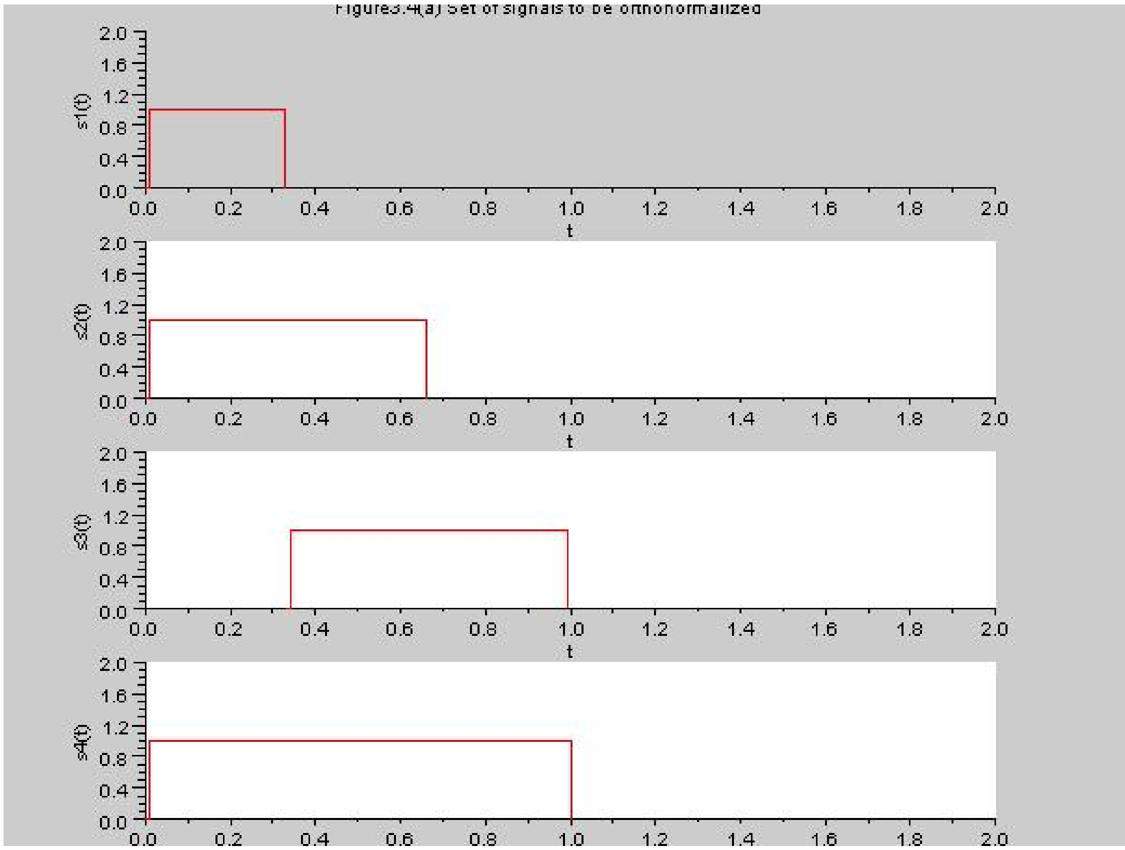

Figure 3.1: Example3.1a

```
a =gca();
a.data_bounds = [0,0;2,4];
plot2d2(t6,phi2t,5)
xlabel('t')
ylabel('phi2(t)')
subplot(3,1,3)
a =gca();
a.data_bounds = [0,0;2,4];
plot2d2(t7,phi3t,5)
xlabel('t')
ylabel('phi3(t)')
```

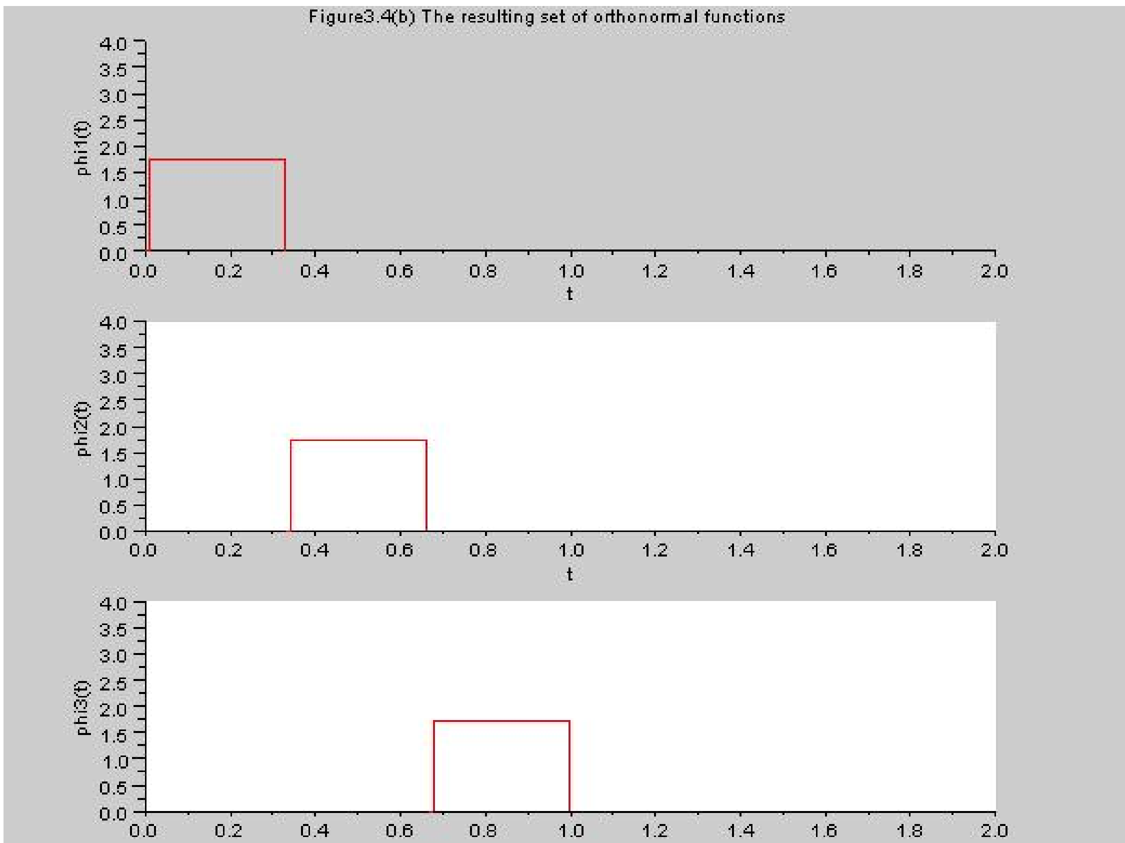

Figure 3.2: Example3.1b

Scilab code Exa 3.2 M ARY Signaling

```
//Caption:M-ARY Signaling
//Example3.2:M-ARY SIGNALING
//Signal constellation and Representation of dibits
clear;
close;
clc;
a =1;  //amplitude =1
T =1; //Symbol duration in seconds
//Four  message points
Si1 = [(-3/2)*a*sqrt(T),(-1/2)*a*sqrt(T),(3/2)*a*sqrt(T),(1/2)*a*sqrt(T)];
a =gca();
a.data_bounds = [-2,-0.5;2,0.5]
plot2d(Si1,[0,0,0,0],-10)
xlabel('phi1(t)')
title('Figure 3.8 (a) Signal constellation')
xgrid(1)
disp('Figure 3.8 (b).Representation of transmitted dibits')
disp('Loc. of meg.point| (-3/2)asqrt(T)|(-1/2)asqrt(T)|(3/2)asqrt(T)|(1/2)asqrt(T)')
disp('-----------------------------------------------------------------------')
disp('Transmitted dibit|          00     |        01          |    11          |    10')
disp('')
disp('')
disp('Figure 3.8 (c). Decision intervals for received dibits')
disp('Received dibit     |      00              |        01              |    11              |    10')
disp('-----------------------------------------------------------------------')
disp('Interval on phi1(t)| x1 < -a.sqrt(T) |-a.sqrt(
```

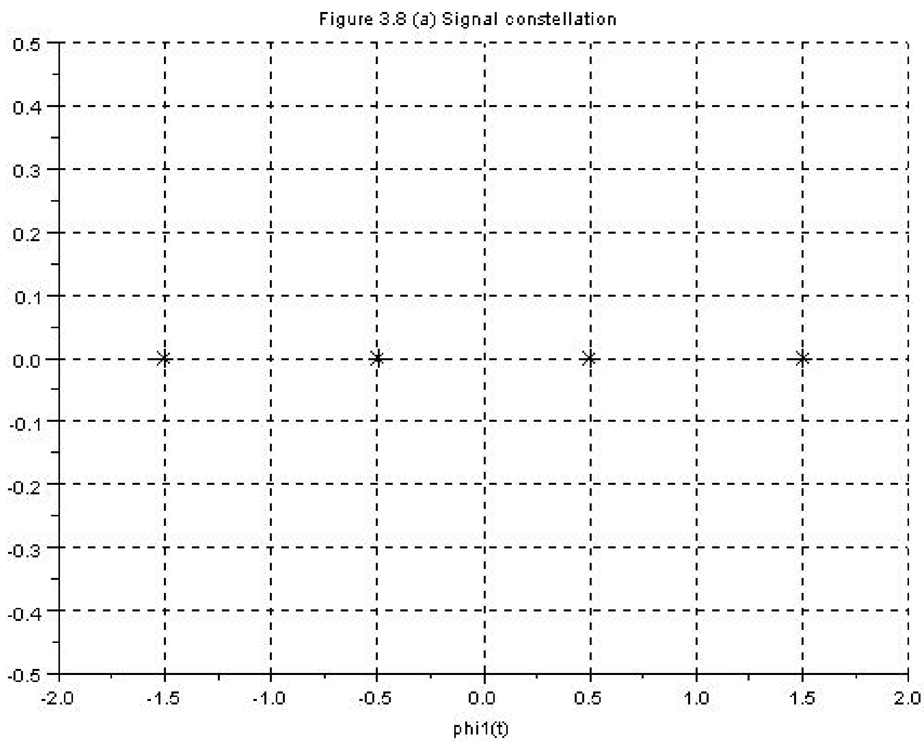

Figure 3.3: Example3.2

```
T)<x1<0| 0<x1<a.sqrt(T) | a.sqrt(T)<x1')
```

Scilab code Exa 3.3 Matched Filter output for RF pulse

```
//Caption:Matched Filter output for RF pulse
//Example3.3: MATCHED FILTER FOR RF PULSE
clear;
close;
clc;
fc =4; //carrier frequency in Hz
T =1;
t1 = 0:0.01:T;
```

```
phit = sqrt(2/T)*cos(2*%pi*fc*t1);
hopt = phit;
phiot = convol(phit,hopt);
phiot = phiot/max(phiot);
t2 = 0:0.01:2*T;
subplot(2,1,1)
a =gca();
a.x_location = "origin";
a.y_location = "origin";
a.data_bounds = [0,-1;1,1];
plot2d(t1,phit);
xlabel('
    t')
ylabel('
    phi(t)')
title('Figure 3.13 (a) RF pulse input')
subplot(2,1,2)
a =gca();
a.x_location = "origin";
a.y_location = "origin";
a.data_bounds = [0,-1;1,1];
plot2d(t2,phiot);
xlabel('
    t')
ylabel('
    phi0(t)')
title('Figure 3.13 (b) Matched Filter output')
```

Scilab code Exa 3.4 Matched Filter output for Noise-like signal

```
//Caption:Matched Filter output for Noise-like
    signal
//Example3.4: Matched Filter output for noise like
```

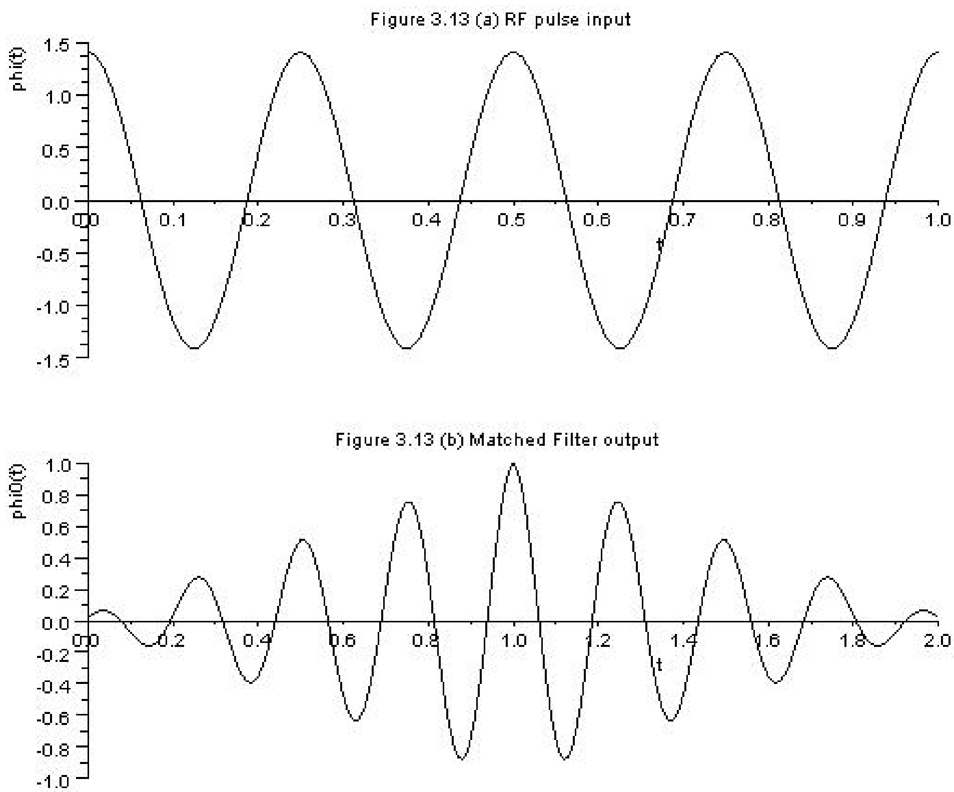

Figure 3.4: Example3.3

```
    input
clear;
close;
clc;
phit =0.1*rand(1,10,'uniform');
hopt = phit;
phi0t = convol(phit,hopt);
phi0t = phi0t/max(phi0t);
subplot(2,1,1)
a =gca();
a.x_location = "origin";
a.y_location = "origin";
a.data_bounds = [0,-1;1,1];
plot2d([1:length(phit)],phit);
xlabel('

    t')
ylabel('

    phi(t)')
title('Figure 3.16 (a) Noise Like input signal')
subplot(2,1,2)
a =gca();
a.x_location = "origin";
a.y_location = "origin";
a.data_bounds = [0,-1;1,1];
plot2d([1:length(phi0t)],phi0t);
xlabel('

    t')
ylabel('

    phi0(t)')
title('Figure 3.16 (b) Matched Filter output')
```

Scilab code Exa 3.6 Linear Predictor of Order one

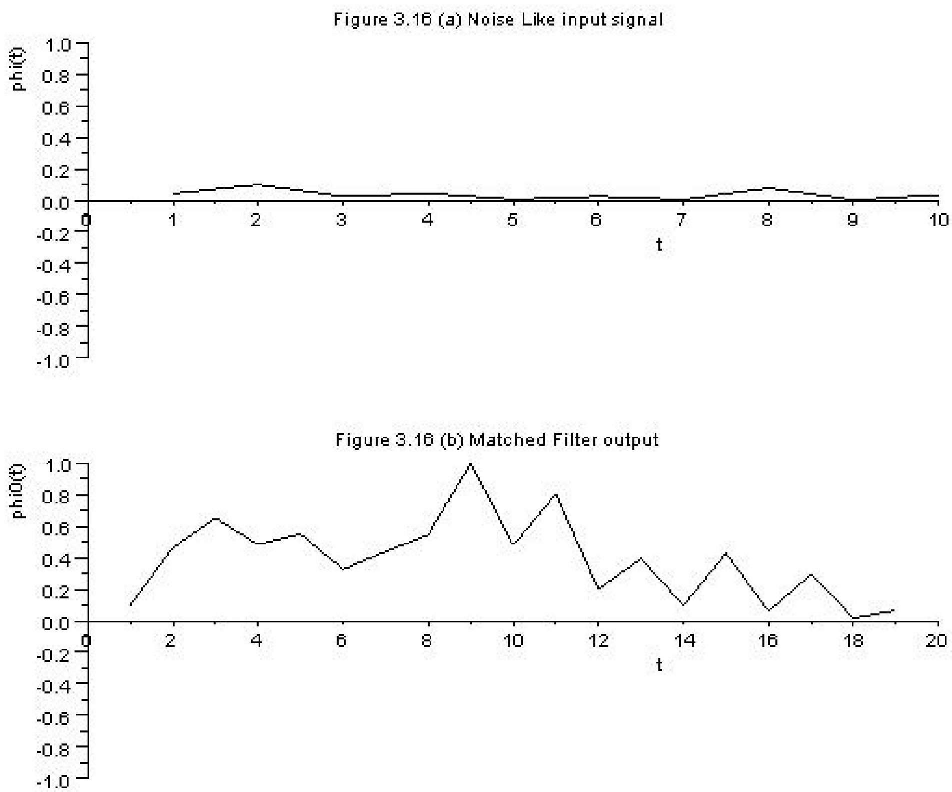

Figure 3.5: Example3.4

```
//Caption:Linear Predictor of Order one
//Example3.6: LINEAR PREDICTION: Predictor of Order
    One
clear;
close;
clc;
Rxx = [0.6 1 0.6];
h01 = Rxx(3)/Rxx(2);  //Rxx(2) = Rxx(0), Rxx(3) =
    Rxx(1)
sigma_E = Rxx(2) - h01*Rxx(3);
sigma_X = Rxx(2);
disp(sigma_E,'Predictor-error variance')
disp(sigma_X,'Predictor input variance')
if(sigma_X > sigma_E)
  disp('The predictor-error variance is less than
     the variance of the predictor input')
end
```

Scilab code CF 3.29 Implementation of LMS Adaptive Filter algorithm

```
//Implementation of LMS ADAPTIVE FILTER
//For noise cancellation application
clear;
clc;
close;
order = 18;
t =0:0.01:1;
x = sin(2*%pi*5*t);
noise =rand(1,length(x));
x_n = x+noise;
ref_noise = noise*rand(10);
w = zeros(order,1);
mu = 0.01*(sum(x.^2)/length(x));
N = length(x);
for k =1:1010
  for i = 1:N-order-1
    buffer = ref_noise(i:i+order-1);
    desired(i) = x_n(i)-buffer*w;
```

```
    w = w+(buffer*mu*desired(i))';
  end
end
subplot(4,1,1)
plot2d(t,x)
title('Orignal Input Signal')
subplot(4,1,2)
plot2d(t,noise,2)
title('random noise')
subplot(4,1,3)
plot2d(t,x_n,5)
title('Signal+noise')
subplot(4,1,4)
plot(desired)
title('noise removed signal')
```

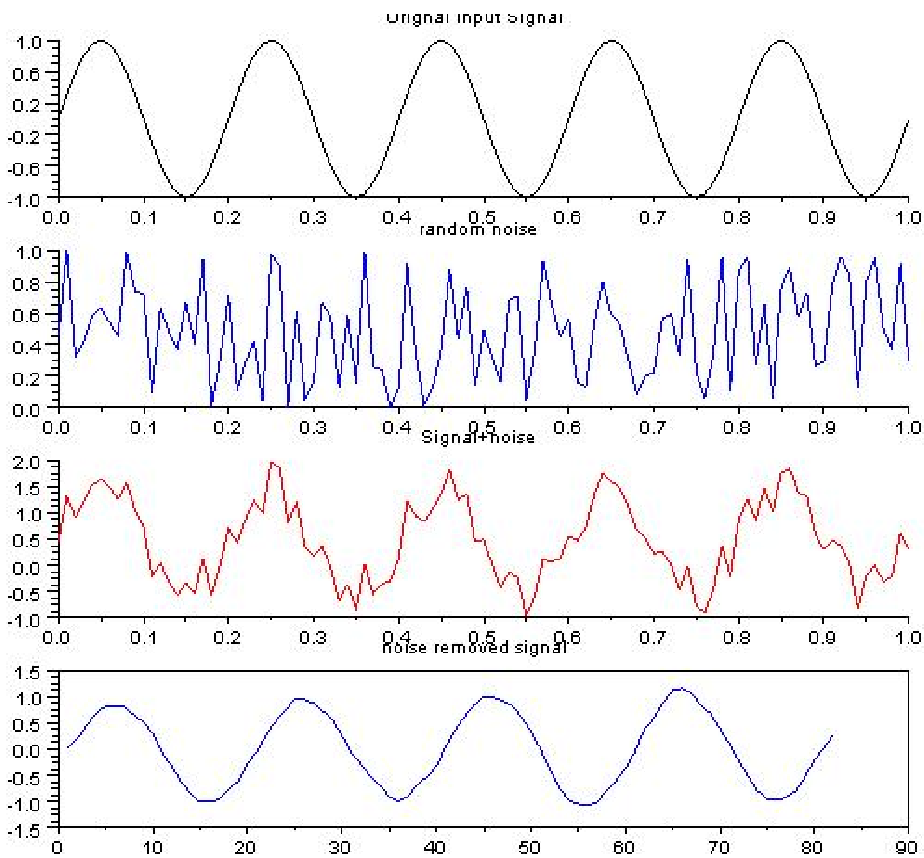

Figure 3.6: Figure3.29

Chapter 4

Sampling Process

Scilab code Exa 4.1 Bound on Aliasing error for Time-shifted sinc pulse

```
//Caption:Bound on Aliasing error for Time-shifted sinc pulse
//Example4.1:Maximum bound on aliasing error for sinc pulse
clc;
close;
t = -1.5:0.01:2.5;
g = 2*sinc_new(2*t-1);
disp(max(g),'Aliasing error cannot exceed max|g(t)|')
f = -1:0.01:1;
G = [0,0,0,0,ones(1,length(f)),0,0,0,0];
f1 = -1.04:0.01:1.04;
subplot(2,1,1)
a=gca();
a.data_bounds =[-3,-1;2,2];
a.x_location = "origin"
a.y_location = "origin"
plot2d(t,g)
xlabel('                                        t')
ylabel('                                        g(t)')
title('Figure 4.8 (a) Sinc pulse g(t)')
subplot(2,1,2)
```

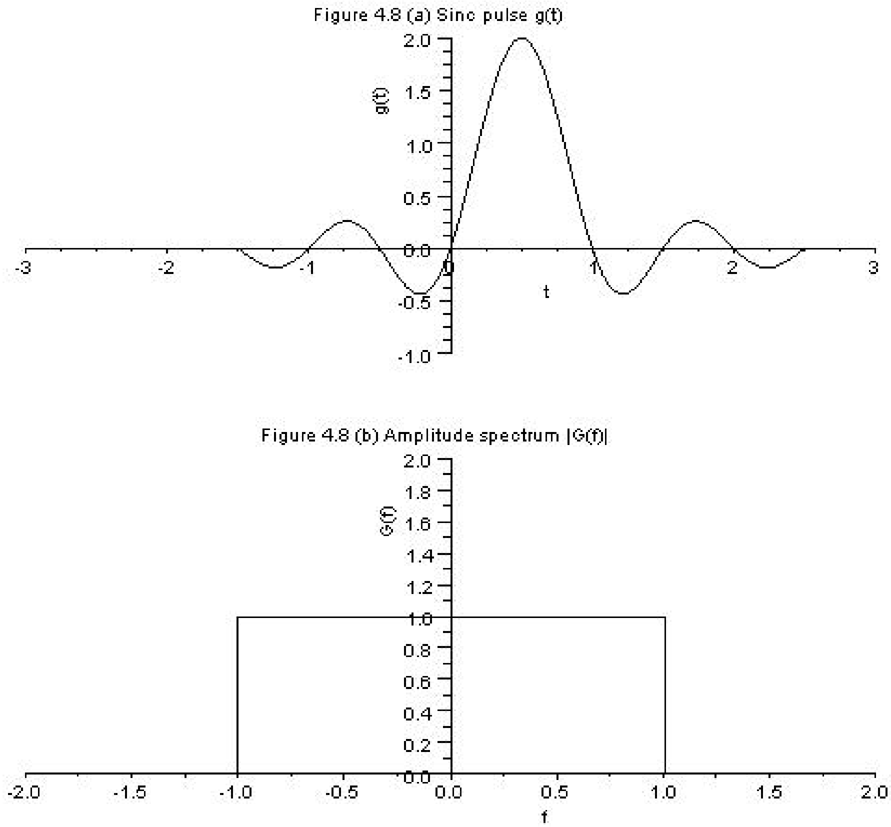

Figure 4.1: Example4.1

```
a=gca();
a.data_bounds =[-2,0;2,2];
a.x_location = "origin"
a.y_location = "origin"
plot2d2(f1,G)
xlabel('                                        f')
ylabel('                                        G(f)')
title('Figure 4.8 (b) Amplitude spectrum |G(f)|')
```

Scilab code Exa 4.3 Equalizier to compensate Aperture effect

```
//Caption:Equalizer to compensate Aperture effect
```

```
//Example4.3:Equalizer to Compensate for aperture
    effect
clc;
close;
T_Ts = 0.01:0.01:0.6;
//E = 1/(sinc_new(0.5*T_Ts));
E(1) =1;
for i = 2:length(T_Ts)
  E(i) = ((%pi/2)*T_Ts(i))/(sin((%pi/2)*T_Ts(i)));
end
a =gca();
a.data_bounds = [0,0.8;0.8,1.2];
plot2d(T_Ts,E,5)
xlabel('Duty cycle T/Ts')
ylabel('1/sinc(0.5(T/Ts))')
title('Figure 4.16 Normalized equalization (to
   compensate for aperture effect) plotted versus T/
   Ts')
```

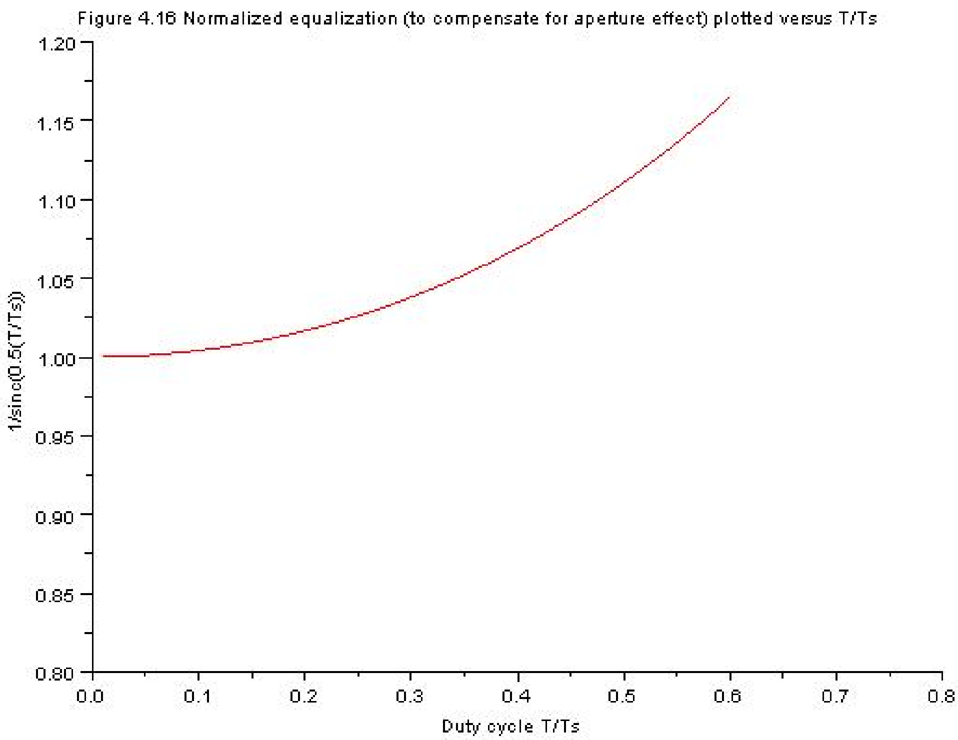

Figure 4.2: Example4.3

Chapter 5

Waveform Coding Techniques

Scilab code Exa 5.1 Average Transmitted Power for PCM

```
//Caption:Average Transmitted Power for PCM
//Example5.1:Average Transmitted Power of PCM
//Page 187
clear;
clc;
sigma_N = input('Enter the noise variance');
k = input('Enter the separation constant for on-off
    signaling');
M = input('Enter the number of discrete amplitude
    levels for NRZ polar');
disp('The average transmitted power is:')
P = (k^2)*(sigma_N)*((M^2)-1)/12;
disp(P)
//Result
//Enter the noise variance 10^-6
//Enter the separation constant for on-off signaling
    7
//Enter the number of discrete amplitude levels for
    NRZ polar 2
// The average transmitted power is: 0.0000122
```

Scilab code Exa 5.2 Comaprision of M-ary PCM with ideal system (Channel Capacity Theorem)

```
//Caption:Comparison of M-ary PCM with ideal system
    (Channel Capacity Theorem)
//Example5.2:Comparison of M-ary PCM system
//Channel Capacity theorem
clear;
close;
clc;
P_NoB_dB = [-20:30];//Input signal-to-noise ratio P/
    NoB, decibels
P_NoB = 10^(P_NoB_dB/10);
k =7; // for M-ary PCM system;
Rb_B = log2(1+(12/k^2)*P_NoB);//bandwidth efficiency
     in bits/sec/Hz
C_B = log2(1+P_NoB);//ideal system according to
    Shannon's channel capacity theorem
//plot
a =gca();
a.data_bounds = [-30,0;40,10];
plot2d(P_NoB_dB,C_B,5)
plot2d(P_NoB_dB,Rb_B,5)
poly1= a.children(1).children(1);
poly1.thickness =2;
poly1.line_style = 4;
xlabel('Input signal-to-noise ratio P/NoB, decibels'
    )
ylabel('Bandwidth efficiency, Rb/B, bits per second
    per hertz')
title('Figure 5.9 Comparison of M-ary PCM with the
    ideal ssytem')
legend(['Ideal System','PCM'])
```

Scilab code Exa 5.3 Signal-to-Quantization Noise Ratio of PCM

```
//Caption:Signal-to-Quantization Noise Ratio of PCM
//Example5.3:Signal-to-Quantization noise ratio
//Channel Bandwidth B
clear;
```

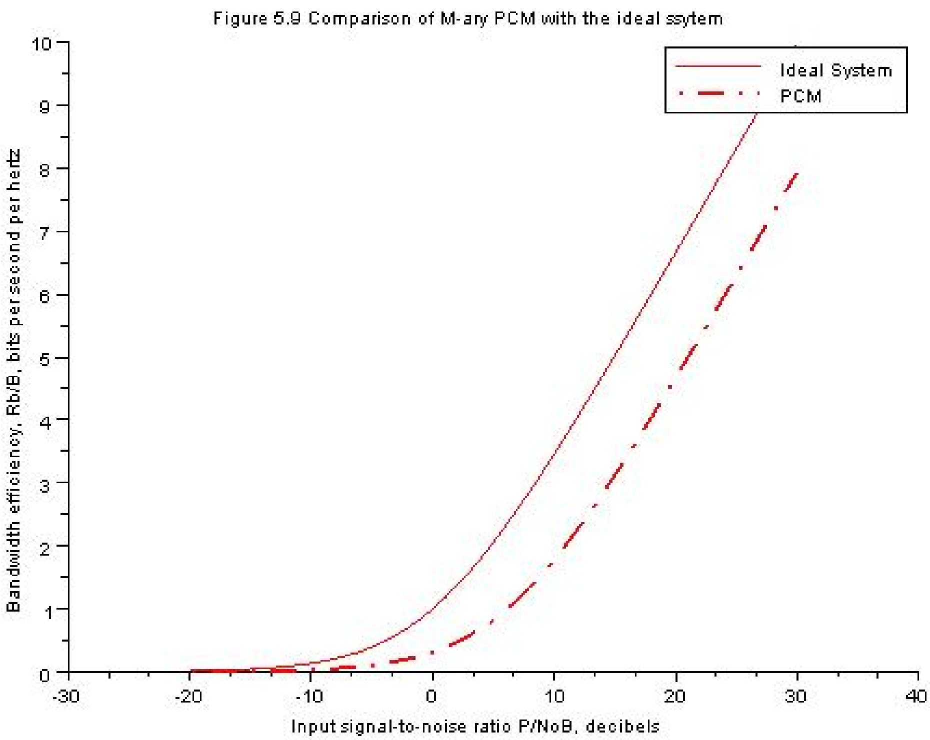

Figure 5.1: Example5.2

```
clc;
n = input('Enter no. of bits used to encode:')
W = input('Enter the message signal banwidth in Hz:'
  )
B = n*W;
disp(B,'Channel width in Hz:')
SNRo = 6*n - 7.2;
disp(SNRo,'Output Signal to noise ratio in dB:')
//Result 1 if  n = 8 bits
//Enter no. of bits used to encode: 8
//Enter the message signal banwidth in Hz: 4000
//Channel width in Hz:  32000.
//Output Signal to noise ratio in dB: 40.8
////////////////////////////////////////////////
//Result 2 if n = 9 bits
//Enter no. of bits used to encode:9
//Enter the message signal banwidth in Hz:4000
//Channel width in Hz: 36000.
//Output Signal to noise ratio in dB: 46.8
///////////////////////////////////////////////
//Conclusion: comparing result 1 with result 2 if
  number of bits increased by 1
//corresponding output signal to noise in PCM
  increased by 6 dB.
```

Scilab code Exa 5.5 Output Signal-to-Noise ratio for Sinusoidal Modulation

```
//Example 5:Delta Modulation – to avoid slope
  overload distortion
//maximum output signal–to–noise ratio for
  sinusoidal modulation
//page 207
clear;
clc;
a0 = input('Enter the amplitude of sinusoidal signal
  :');
```

```
f0 = input('Enter the frequency of sinusoidal signal in Hz:');
fs = input('Enter the sampling frequency in samples per seconds:');
Ts = 1/fs;//Sampling interval
delta = 2*%pi*f0*a0*Ts;//Step size to avoid slope overload
Pmax = (a0^2)/2;//maximum permissible output power
sigma_Q = (delta^2)/3;//Quantization error or noise variance
W = f0;//Maximum message bandwidth
N = W*Ts*sigma_Q; //Average output noise power
SNRo = Pmax/N; // Maximum output signal-to-noise ratio
SNRo_dB = 10*log10(SNRo);
disp(SNRo_dB,'Maximum output signal-to-nosie in dB for Delta Modualtion:')
//Result 1 for fs = 8000 Hertz
//Enter the amplitude of sinusoidal signal:1
//Enter the frequency of sinusoidal signal in Hz:4000
//Enter the sampling frequency in samples per seconds:8000
//Maximum output signal-to-nosie in dB for Delta Modualtion:-5.1717849
///////////////////////////////////////////////////////////////////
//Result 2 for fs = 16000 Hertz
//Enter the amplitude of sinusoidal signal:1
//Enter the frequency of sinusoidal signal in Hz:4000
//Enter the sampling frequency in samples per seconds:16000
//Maximum output signal-to-nosie in dB for Delta Modualtion:3.859115
///////////////////////////////////////////////////////////////////
```

```
//Conclusion: comparing result 1 with result 2, if the sampling frequency
//is doubled the signal to noise increased by 9 dB
```

Scilab code CF 5.13a (a) u-Law companding

```
//Caption:u-Law companding
//Figure5.13(a)Mulaw companding Nonlinear Quantization
//Plotting mulaw characteristics for different
//Values of mu
clc;
x = 0:0.01:1; //Normalized input
mu = [0,5,255];//different values of mu
for i = 1:length(mu)
  [Cx(i,:),Xmax(i)] =  mulaw(x,mu(i));
end
plot2d(x/Xmax(1),Cx(1,:),2)
plot2d(x/Xmax(2),Cx(2,:),4)
plot2d(x/Xmax(3),Cx(3,:),6)
xtitle('Compression Law: u-Law companding','Normalized Input |x|','Normalized Output |c(x)|');
legend(['u =0'],['u=5'],['u=255'])
```

Scilab code CF 5.13b (b) A-law companding

```
//Caption:A-law companding
//Figure5.13(b)A-law companding, Nonlinear Quantization
//Plotting A-law characteristics for different
//Values of A
clc;
x = 0:0.01:1; //Normalized input
A = [1,2,87.56];//different values of A
for i = 1:length(A)
```

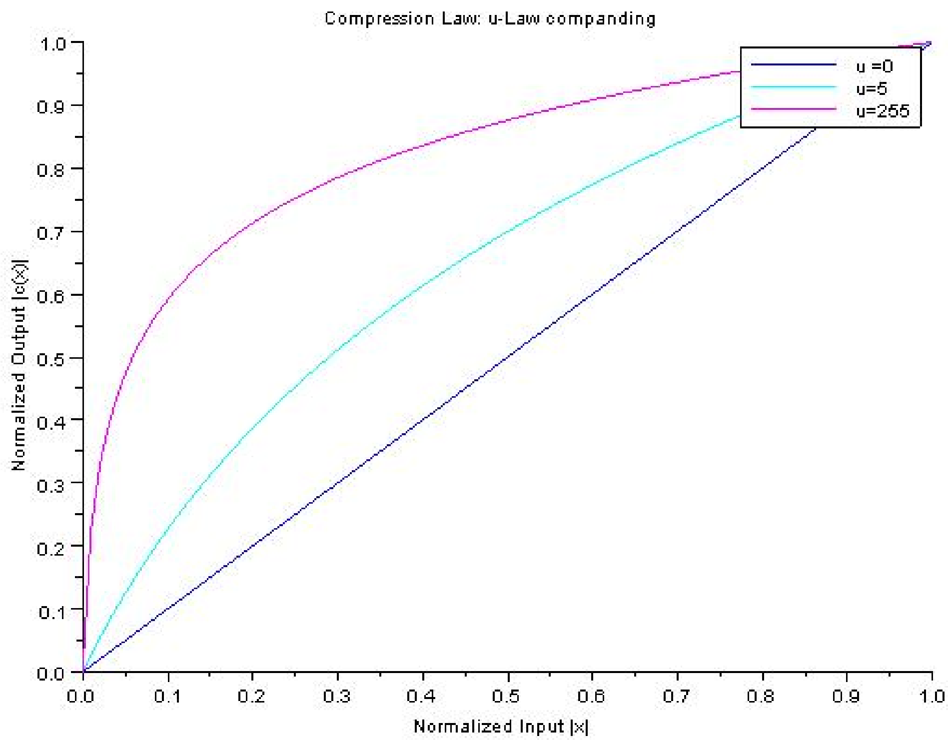

Figure 5.2: Figure5.13a

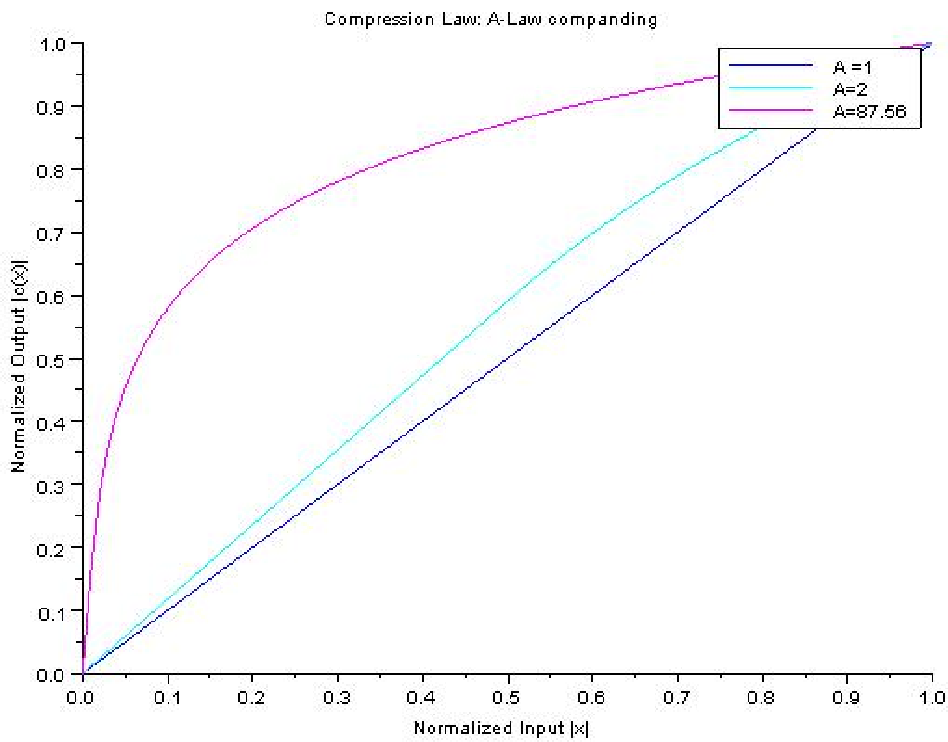

Figure 5.3: Figure5.13b

```
  [Cx(i,:),Xmax(i)] =  Alaw(x,A(i));
end
plot2d(x/Xmax(1),Cx(1,:),2)
plot2d(x/Xmax(2),Cx(2,:),4)
plot2d(x/Xmax(3),Cx(3,:),6)
xtitle('Compression Law: A–Law companding','Normalized Input |x|','Normalized Output |c(x)|');
legend(['A =1'],['A=2'],['A=87.56'])
```

Chapter 6

Baseband Shaping for Data Transmission

Scilab code Exa 6.1 Bandwidth Requirements of the T1 carrier

```
//Caption:Bandwidth Requirements of the T1 carrier
//Example6.1:Bandwidth Requirements of the T1
    Carrier
//Page 251
clear;
clc;
Tb = input('Enter the bit duration of the TDM signal
    :')
Bo = 1/(2*Tb);//minimum transmission bandwidth of T1
     system
//Transmission bandwidth for raised cosine spectrum
    'B'
alpha = 1;//cosine roll-off factor
f1 = Bo*(1-alpha);
B = 2*Bo-f1;
disp(B,'Transmission bandwidth for raised cosine
    spectrum in Hz:')
//Result
//Enter the bit duration of the TDM signal
    :0.647*10^-6
//Transmission bandwidth for raised cosine spectrum
```

```
in Hz:1545595.1
```

Scilab code CF 6.1a (a) Nonreturn-to-zero unipolar format

```
//Caption:Nonreturn-to-zero unipolar format
//Figure 6.1(a): Discrete PAM Signals Generation
//[1].Unipolar NRZ
//page 235
clear;
close;
clc;
x = [0 1 0 0 0 1 0 0 1 1];
binary_zero = [0 0 0 0 0 0 0 0 0 0];
binary_one = [1 1 1 1 1 1 1 1 1 1];
L = length(x);
L1 = length(binary_zero);
total_duration = L*L;
//plotting
a =gca();
a.data_bounds =[0 -2;L*L1 2];
for i =1:L
  if(x(i)==0)
    plot([i*L-L+1:i*L],binary_zero);
    poly1= a.children(1).children(1);
    poly1.thickness =3;
  else
    plot([i*L-L+1:i*L],binary_one);
    poly1= a.children(1).children(1);
    poly1.thickness =3;
  end
end
xgrid(1)
title('Unipolar NRZ')
```

Scilab code CF 6.1b (b) Nonreturn-to-zero polar format

```
//Caption:Nonreturn-to-zero polar format
```

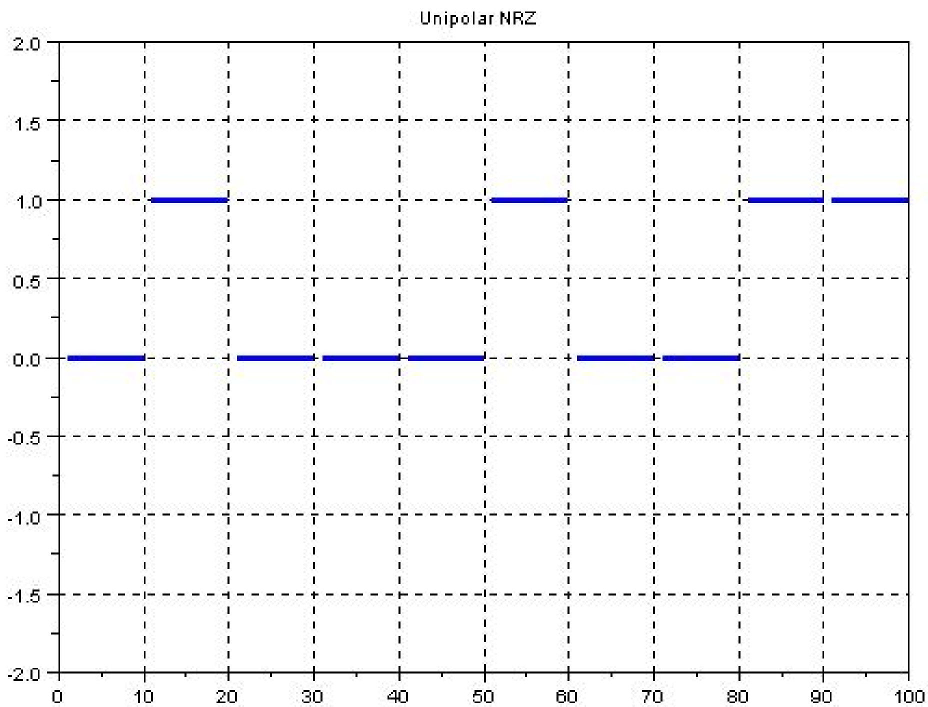

Figure 6.1: Figure6.1a

```
//Figure 6.1(b): Discrete PAM Signals Generation
// [2].Polar NRZ
//page 235
clear;
close;
clc;
x = [0 1 0 0 0 1 0 0 1 1];
binary_negative = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1];
binary_positive = [1 1 1 1 1 1 1 1 1 1];
L = length(x);
L1 = length(binary_negative);
total_duration = L*L1;
//plotting
a =gca();
a.data_bounds =[0 -2;L*L1 2];
for i =1:L
  if(x(i)==0)
    plot([i*L-L+1:i*L],binary_negative);
    poly1= a.children(1).children(1);
    poly1.thickness =3;
  else
    plot([i*L-L+1:i*L],binary_positive);
    poly1= a.children(1).children(1);
    poly1.thickness =3;
  end
end
xgrid(1)
title('Polar NRZ')
```

Scilab code CF 6.1c (c) Nonreturn-to-zero bipolar format

```
//Caption:Nonreturn-to-zero bipolar format
//Figure 6.1(c):Discrete PAM Signals Generation
// [3].BiPolar NRZ
//page 235
clear;
close;
```

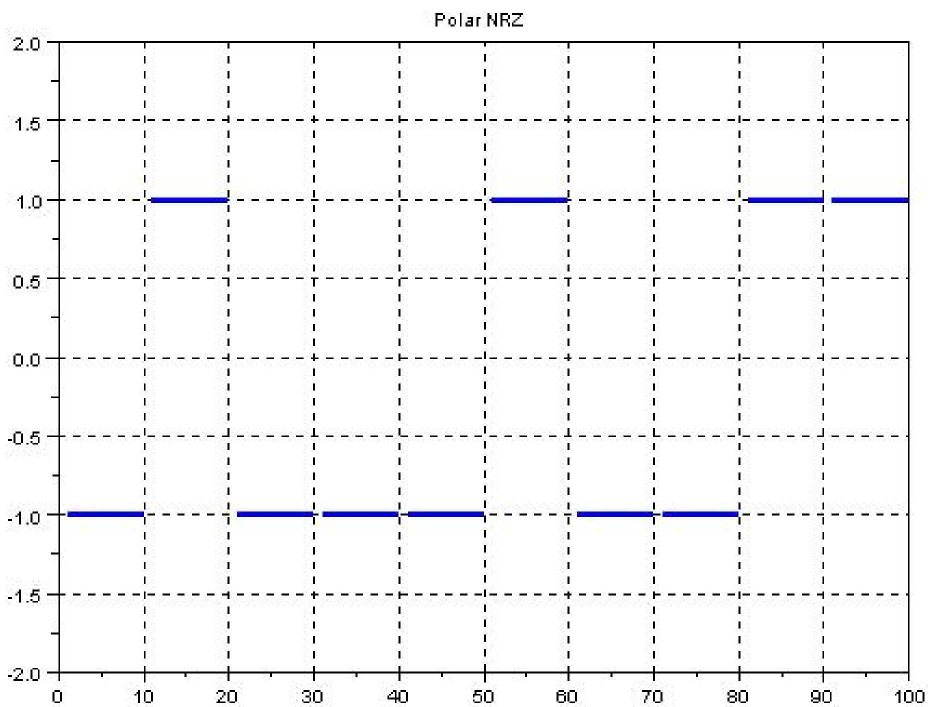

Figure 6.2: Figure6.1b

```
clc;
x = [0 1 1 0 0 1 0 0 1 1];
binary_negative = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1];
binary_zero = [0 0 0 0 0 0 0 0 0 0];
binary_positive = [1 1 1 1 1 1 1 1 1 1];
L = length(x);
L1 = length(binary_negative);
total_duration = L*L1;
//plotting
a =gca();
a.data_bounds =[0 -2;L*L1 2];
for i =1:L
  if(x(i)==0)
    plot([i*L-L+1:i*L],binary_zero);
    poly1= a.children(1).children(1);
    poly1.thickness =3;
  elseif((x(i)==1)&(x(i-1)~=1))
    plot([i*L-L+1:i*L],binary_positive);
    poly1= a.children(1).children(1);
    poly1.thickness =3;
  else
    plot([i*L-L+1:i*L],binary_negative);
    poly1= a.children(1).children(1);
    poly1.thickness =3;
  end
end
xgrid(1)
title('BiPolar NRZ')
```

Scilab code Exa 6.2 Duobinary Encoding

```
//Caption:Duobinary Encoding
//Example6.2: Precoded Duobinary coder and decoder
//Page 256
clc;
b = [0,0,1,0,1,1,0];//input binary sequence:precoder
    input
```

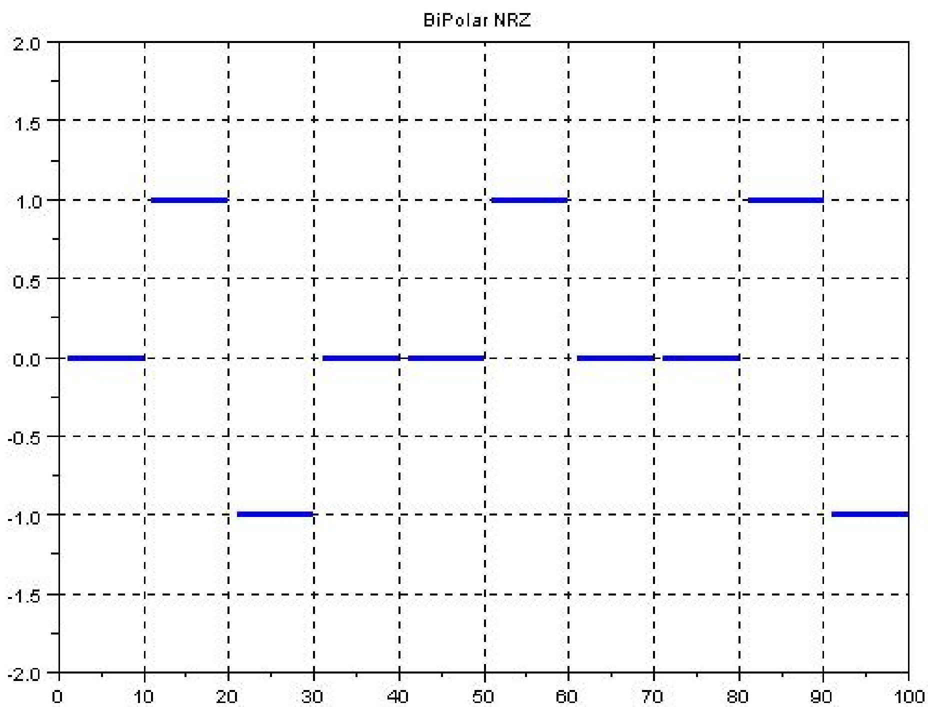

Figure 6.3: Figure6.1c

```
a(1) = xor(1,b(1));
if(a(1)==1)
  a_volts(1) = 1;
end
for k =2:length(b)
  a(k) = xor(a(k-1),b(k));
  if(a(k)==1)
    a_volts(k)=1;
  else
    a_volts(k)=-1;
  end
end
a = a';
a_volts = a_volts';
disp(a,'Precoder output in binary form:')
disp(a_volts,'Precoder output in volts:')
//Duobinary coder output in volts
c(1) = 1+ a_volts(1);
for k =2:length(a)
  c(k) =  a_volts(k-1)+a_volts(k);
end
c = c';
disp(c,'Duobinary coder output in volts:')
//Duobinary decoder output  by applying decision
   rule
for k =1:length(c)
  if(abs(c(k))>1)
    b_r(k) =  0;
  else
    b_r(k) = 1;
  end
end
b_r = b_r';
disp(b_r,'Recovered original sequence at detector
   oupupt:')
//Result
//Precoder output in binary form:
//
```

```
//   1.      1.      0.      0.      1.      0.      0.
//
// Precoder output in volts:
//
//   1.      1.    - 1.    - 1.      1.    - 1.    - 1.
//
// Duobinary coder output in volts:
//
//   2.      2.      0.    - 2.      0.      0.    - 2.
//
// Recovered original sequence at detector oupupt:
//
//   0.      0.      1.      0.      1.      1.      0.
```

Scilab code Exa 6.3 Generation of bipolar output for duobinary coder

```
//Caption:Generation of bipolar output for duobinary coder
//Example6.3:Operation of Circuit in figure 6.13
//for generating bipolar format
//page 256 and page 257
//Refer Table 6.4
clc;
x = [0,1,1,0,1,0,0,0,1,1];//input binary sequence:precoder input
y(1) = 1;
for k =2:length(x)+1
  y(k) = xor(x(k-1),y(k-1));
end
y_delay = y(1:$-1);
y = y';
y_delay = y_delay';
disp(y,'Modulo-2 adder output:')
disp(y_delay,'Delay element output:')
for k = 1:length(y_delay)
  z(k) = y(k+1)-y_delay(k);
end
z = z';
```

```
disp(z,'differential encoder bipolar output in volts
    :')
//Result
//Modulo-2 adder output:
//      1.      1.      0.      1.      1.      0.      0.      0.
        0.      1.      0.
// Delay element output:
//      1.      1.      0.      1.      1.      0.      0.      0.
        0.      1.
// differential encoder bipolar output in volts:
//      0.    - 1.      1.      0.    - 1.      0.      0.      0.
        1.    - 1.
```

Scilab code CF 6.4 Power Spectra of different binary data formats

```
//Caption:Power Spectra of different binary data
    formats
//Figure 6.4: Power Spectal Densities of
//Different Line Coding Techniques
//[1].NRZ Polar Format [2].NRZ Bipolar format
//[3].NRZ Unipolar format [4]. Manchester format
//Page 241
close;
clc;
//[1]. NRZ Polar format
a = input('Enter the Amplitude value:');
fb = input('Enter the bit rate:');
Tb = 1/fb;  //bit duration
f = 0:1/(100*Tb):2/Tb;
for i = 1:length(f)
  Sxxf_NRZ_P(i) = (a^2)*Tb*(sinc_new(f(i)*Tb)^2);
  Sxxf_NRZ_BP(i) = (a^2)*Tb*((sinc_new(f(i)*Tb))^2)
     *((sin(%pi*f(i)*Tb))^2);
  if (i==1)
    Sxxf_NRZ_UP(i) = (a^2)*(Tb/4)*((sinc_new(f(i)*Tb
       ))^2)+(a^2)/4;
  else
```

```
    Sxxf_NRZ_UP(i) = (a^2)*(Tb/4)*((sinc_new(f(i)*Tb
       ))^2);
  end
  Sxxf_Manch(i) = (a^2)*Tb*(sinc_new(f(i)*Tb/2)^2)*(
     sin(%pi*f(i)*Tb/2)^2);
end
//Plotting
a = gca();
plot2d(f,Sxxf_NRZ_P)
poly1= a.children(1).children(1);
poly1.thickness = 2;  // the tickness of a curve.
plot2d(f,Sxxf_NRZ_BP,2)
poly1= a.children(1).children(1);
poly1.thickness = 2;  // the tickness of a curve.
plot2d(f,Sxxf_NRZ_UP,5)
poly1= a.children(1).children(1);
poly1.thickness = 2;  // the tickness of a curve.
plot2d(f,Sxxf_Manch,9)
poly1= a.children(1).children(1);
poly1.thickness = 2;  // the tickness of a curve.
xlabel('f*Tb-------->')
ylabel('Sxx(f)-------->')
title('Power Spectral Densities of Different Line
   Codinig Techniques')
xgrid(1)
legend(['NRZ Polar Format','NRZ Bipolar format','NRZ
     Unipolar format','Manchester format']);
//Result
//Enter the Amplitude value:1
//Enter the bit rate:1
```

Scilab code CF 6.6b (b) Ideal solution for zero ISI

```
//Caption:Ideal solution for zero ISI
//Figure 6.6(b): Ideal Solution for Intersymbol
   Interference
//SINC pulse
```

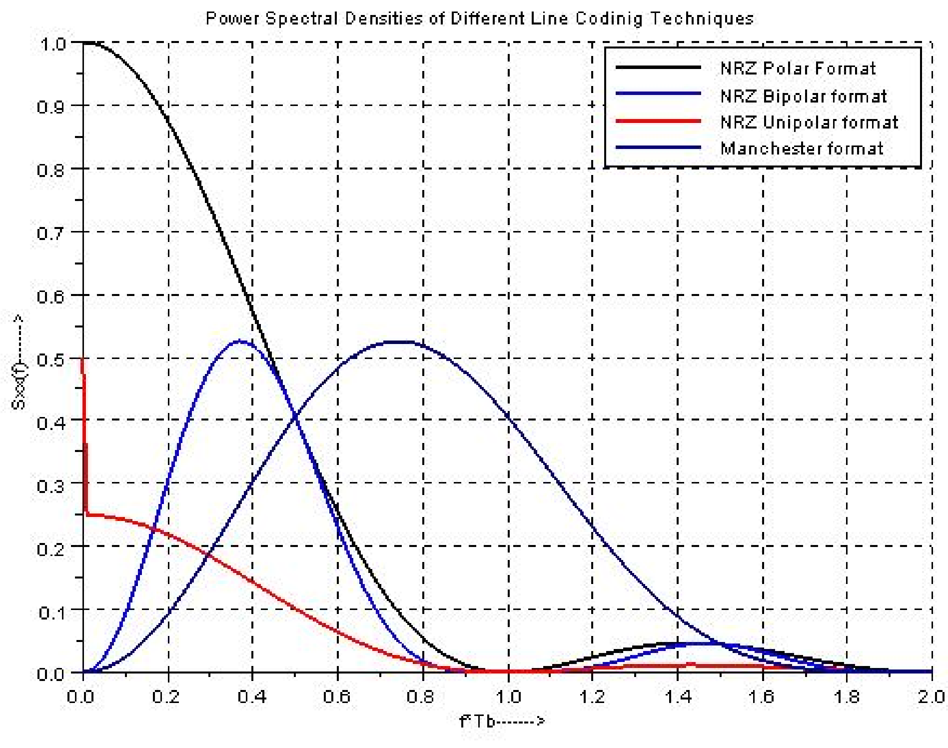

Figure 6.4: Figure6.4

```
//page 249
rb = input('Enter the bit rate:');
Bo = rb/2;
t =-3:1/100:3;
x = sinc_new(2*Bo*t);
plot(t,x)
xlabel('t-------->');
ylabel('p(t)--------->');
title('SINC Pulse for zero ISI')
xgrid(1)
//Result
//Enter the bit rate:1
```

Scilab code CF 6.7b (b) Practical solution: Raised Cosine

```
//Caption:Practical solution: Raised Cosine
//Figure6.7(b):Practical Solution for Intersymbol Interference
//Raised Cosine Spectrum
//page 250
close;
clc;
rb = input('Enter the bit rate:');
Tb =1/rb;
t =-3:1/100:3;
Bo = rb/2;
Alpha =0;       //Intialized to zero
x =t/Tb;
for j =1:3
  for i =1:length(t)
    if((j==3)&((t(i)==0.5)|(t(i)==-0.5)))
        p(j,i) = sinc_new(2*Bo*t(i));
    else
        num =  sinc_new(2*Bo*t(i))*cos(2*%pi*Alpha*Bo*t(i));
        den =   1-16*(Alpha^2)*(Bo^2)*(t(i)^2)+0.01;
        p(j,i)= num/den;
```

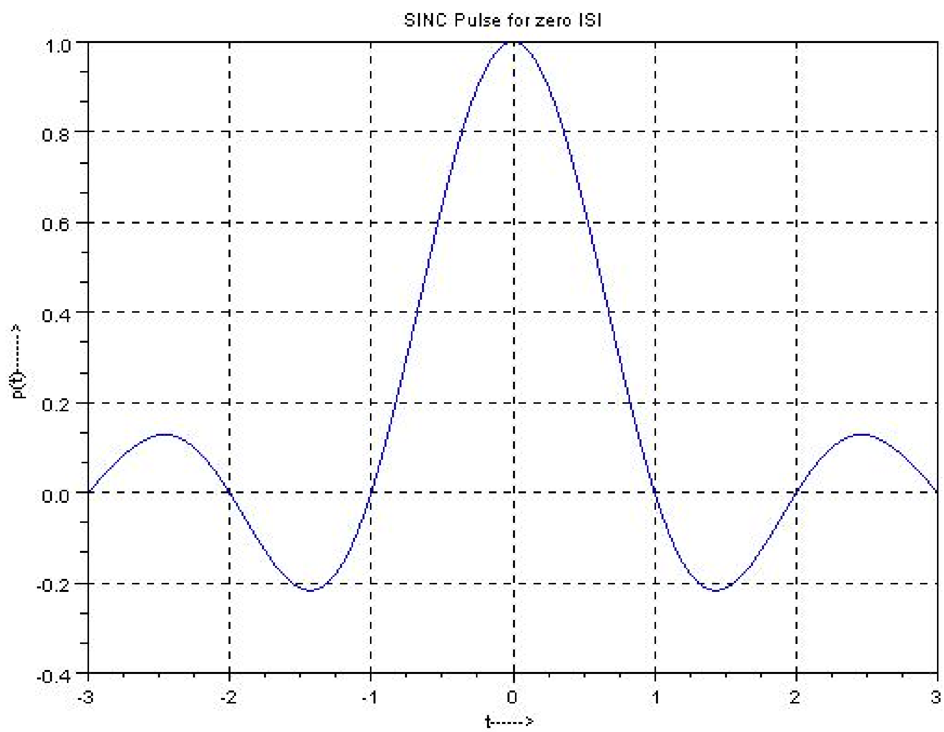

Figure 6.5: Figure6.6

```
    end
  end
  Alpha = Alpha+0.5;
end
a =gca();
plot2d(t,p(1,:))
plot2d(t,p(2,:))
poly1= a.children(1).children(1);
poly1.foreground=2;
plot2d(t,p(3,:))
poly2= a.children(1).children(1);
po1y2.foreground=4;
poly2.line_style = 3;
xlabel('t/Tb------------>');
ylabel('p(t)-------------->');
title('RAISED COSINE SPECTRUM - Practical Solution for ISI')
legend(['ROlloff Factor =0','ROlloff Factor =0.5','ROlloff Factor =1'])
xgrid(1)
//Result
//Enter the bit rate:1
```

Scilab code CF 6.9 Frequency response of duobinary conversion filter

```
//Caption:Frequency response of duobinary conversion filter
//Figure6.9:Frequency Response of Duobinary Conversion filter
//(a)Amplitude Response
//(b)Phase Response
//Page 254
clear;
close;
clc;
rb =  input('Enter the bit rate=');
Tb =1/rb;  //Bit duration
```

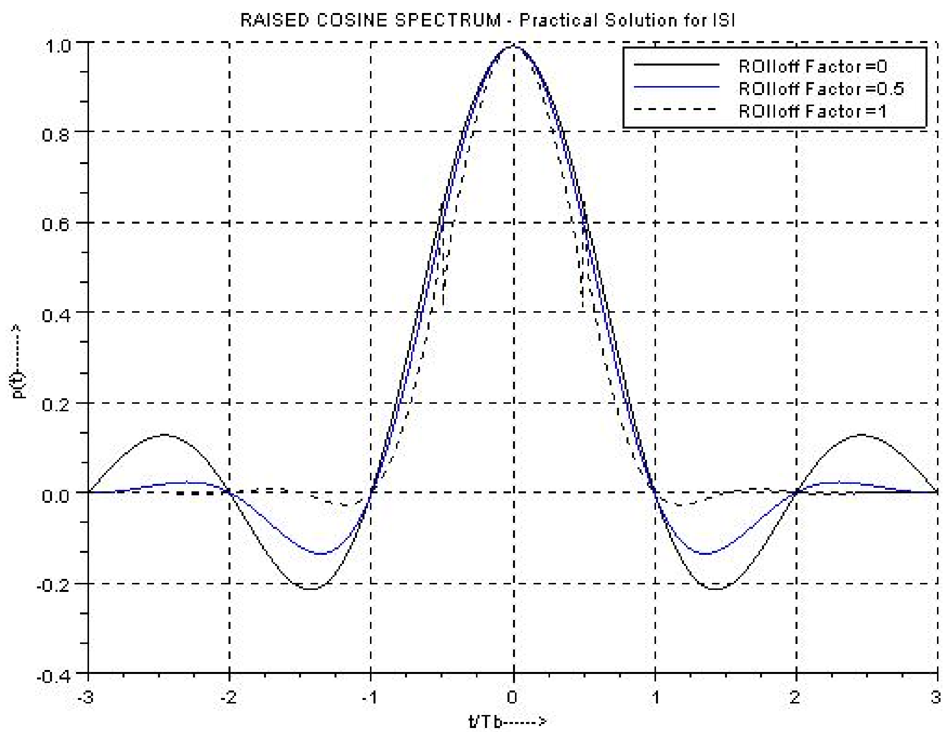

Figure 6.6: Figure6.7

```
f = -rb/2:1/100:rb/2;
Amplitude_Response = abs(2*cos(%pi*f.*Tb));
Phase_Response = -(%pi*f.*Tb);
subplot(2,1,1)
a=gca();
a.x_location ="origin";
a.y_location ="origin";
plot2d(f,Amplitude_Response,2)
poly1= a.children(1).children(1);
poly1.thickness = 2;  // the tickness of a curve.
xlabel('Frequency f———>')
ylabel('|H(f)| —————>')
title('Amplitude Repsonse of Duobinary Singaling')
subplot(2,1,2)
a=gca();
a.x_location ="origin";
a.y_location ="origin";
plot2d(f,Phase_Response,5)
poly1= a.children(1).children(1);
poly1.thickness = 2;  // the tickness of a curve.
xlabel('
    Frequency f———>')
ylabel('
    <H(f) —————>')
title('Phase Repsonse of Duobinary Singaling')
//Result
//Enter the bit rate=8
```

Scilab code CF 6.15 Frequency response of modified duobinary conversion filter

```
//Caption:Frequency response of modified duobinary
    conversion filter
//Figure 6.15: Frequency Response of Modified
    duobinary conversion filter
//(a)Amplitude Response
//(b)Phase Response
```

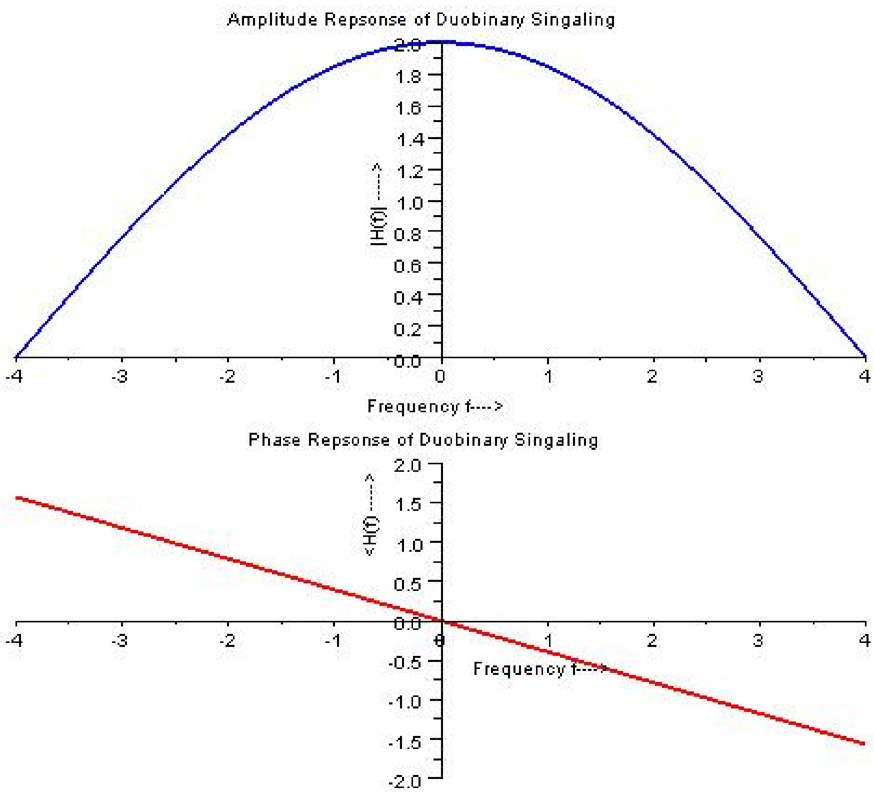

Figure 6.7: Figure6.9

```
//page 259
clear;
close;
clc;
rb =  input('Enter the bit rate=');
Tb =1/rb;  //Bit duration
f = -rb/2:1/100:rb/2;
Amplitude_Response = abs(2*sin(2*%pi*f.*Tb));
Phase_Response = -(2*%pi*f.*Tb);
subplot(2,1,1)
a=gca();
a.x_location ="origin";
a.y_location ="origin";
plot2d(f,Amplitude_Response,2)
poly1= a.children(1).children(1);
poly1.thickness = 2;  // the tickness of a curve.
xlabel('Frequency f---->')
ylabel('|H(f)| ----->')
title('Amplitude Repsonse of Modified Duobinary Singaling')
xgrid(1)
subplot(2,1,2)
a=gca();
a.x_location ="origin";
a.y_location ="origin";
plot2d(f,Phase_Response,5)
poly1= a.children(1).children(1);
poly1.thickness = 2;  // the tickness of a curve.
xlabel('Frequency f---->')
ylabel('<H(f) ----->')
title('Phase Repsonse of Modified Duobinary Singaling')
xgrid(1)
//Result
//Enter the bit rate=8
```

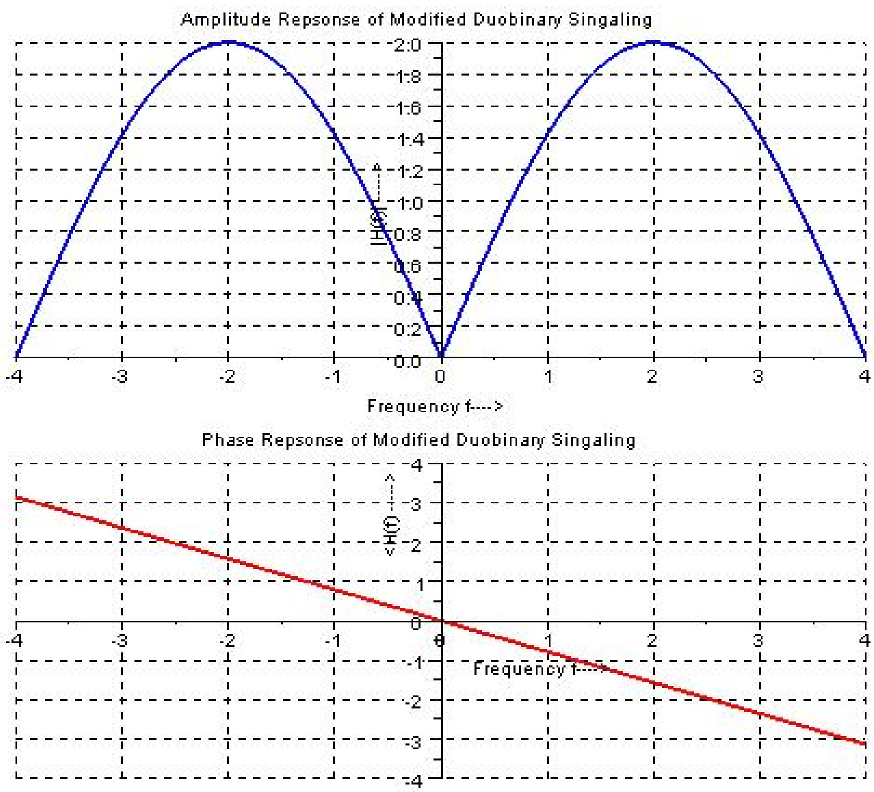

Figure 6.8: Figure6.15

Chapter 7

Digital Modulation Techniques

Scilab code Exa 7.1 QPSK Waveform

```
//Caption:Waveforms of Different Digital Modulation
    techniques
//Example7.1 Signal Space Diagram for coherent QPSK
    system
clear;
clc;
close;
M =4;
i = 1:M;
t = 0:0.001:1;
for i = 1:M
  s1(i,:) = cos(2*%pi*2*t)*cos((2*i-1)*%pi/4);
  s2(i,:) = -sin(2*%pi*2*t)*sin((2*i-1)*%pi/4);
end
S1 =[];
S2 = [];
S = [];
Input_Sequence =[0,1,1,0,1,0,0,0];
m = [3,1,1,2];
for i =1:length(m)
  S1 = [S1 s1(m(i),:)];
  S2 = [S2 s2(m(i),:)];
end
```

```
S = S1+S2;
figure
subplot(3,1,1)
a =gca();
a.x_location = "origin";
plot(S1)
title('Binary PSK wave of Odd-numbered bits of input sequence')
subplot(3,1,2)
a =gca();
a.x_location = "origin";
plot(S2)
title('Binary PSK wave of Even-numbered bits of input sequence')
subplot(3,1,3)
a =gca();
a.x_location = "origin";
plot(S)
title('QPSK waveform')
//-sin((2*i-1)*%pi/4)*%i;
//annot = dec2bin([0:length(y)-1],log2(M));
//disp(y,'coordinates of message points')
//disp(annot,'dibits value')
//figure;
//a =gca();
//a.data_bounds = [-1,-1;1,1];
//a.x_location = "origin";
//a.y_location = "origin";
//plot2d(real(y(1)),imag(y(1)),-2)
//plot2d(real(y(2)),imag(y(2)),-4)
//plot2d(real(y(3)),imag(y(3)),-5)
//plot2d(real(y(4)),imag(y(4)),-9)
//xlabel('                                        In-Phase');
//ylabel('                                        Quadrature');
```

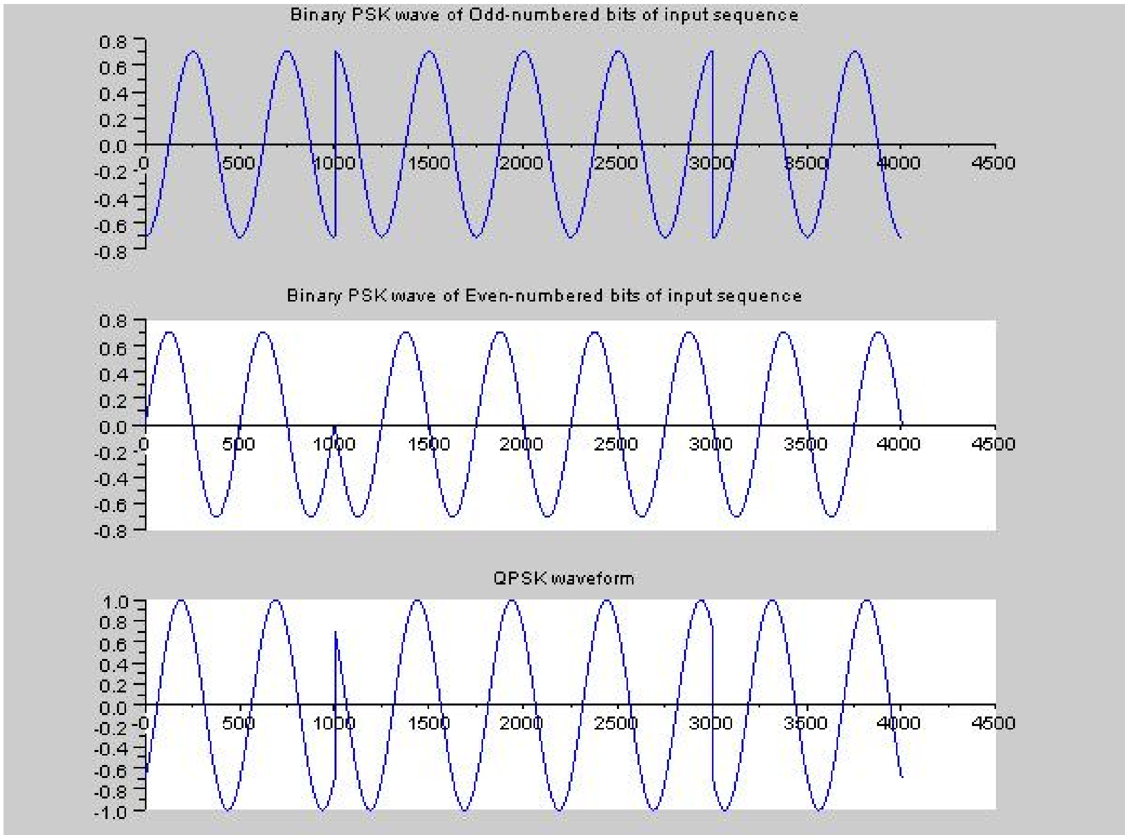

Figure 7.1: Example7.1

```
//title('Constellation for QPSK')
//legend(['message point 1 (dibit 10)';'message
    point 2 (dibit 00)';'message point 3 (dibit 01)
    ';'message point 4 (dibit 11)'],5)
```

Scilab code CF 7.1 Waveform of Different Digital Modulation techniques

```
//Caption:Waveforms of Different Digital Modulation
    techniques
//Figure7.1
//Digital Modulation Techniques
//To Plot the ASK, FSK and PSk Waveforms
```

```
clear;
clc;
close;
f = input('Enter the Analog Carrier Frequency in Hz'
   );
t = 0:1/512:1;
x = sin(2*%pi*f*t);
I = input('Enter the digital binary data');
//Generation of ASK Waveform
Xask = [];
for n = 1:length(I)
  if((I(n)==1)&(n==1))
    Xask = [x,Xask];
  elseif((I(n)==0)&(n==1))
    Xask = [zeros(1,length(x)),Xask];
  elseif((I(n)==1)&(n~=1))
    Xask = [Xask,x];
  elseif((I(n)==0)&(n~=1))
    Xask = [Xask,zeros(1,length(x))];
  end
end
//Generation of FSK Waveform
Xfsk = [];
x1 = sin(2*%pi*f*t);
x2 = sin(2*%pi*(2*f)*t);
for n = 1:length(I)
  if (I(n)==1)
      Xfsk = [Xfsk,x2];
  elseif (I(n)~=1)
    Xfsk = [Xfsk,x1];
  end
end
//Generation of PSK Waveform
Xpsk = [];
x1 = sin(2*%pi*f*t);
x2 = -sin(2*%pi*f*t);
for n = 1:length(I)
  if (I(n)==1)
```

```
      Xpsk = [Xpsk,x1];
    elseif (I(n)~=1)
      Xpsk = [Xpsk,x2];
    end
end
figure
plot(t,x)
xtitle('Analog Carrier Signal for Digital Modulation')
xgrid
figure
plot(Xask)
xtitle('Amplitude Shift Keying')
xgrid
figure
plot(Xfsk)
xtitle('Frequency Shift Keying')
xgrid
figure
plot(Xpsk)
xtitle('Phase Shift Keying')
xgrid
//Example
//Enter the Analog Carrier Frequency 2
//Enter the digital binary data[0,1,1,0,1,0,0,1]
```

Scilab code Exa 7.2 MSK waveforms

```
//Caption:Signal Space diagram for coherent BPSK
//Example7.2: Sequence and Waveforms for MSK signal
//Table 7.2 signal space characterization of MSK
clear
clc;
close;
M =2;
Tb =1;
t1 = -Tb:0.01:Tb;
```

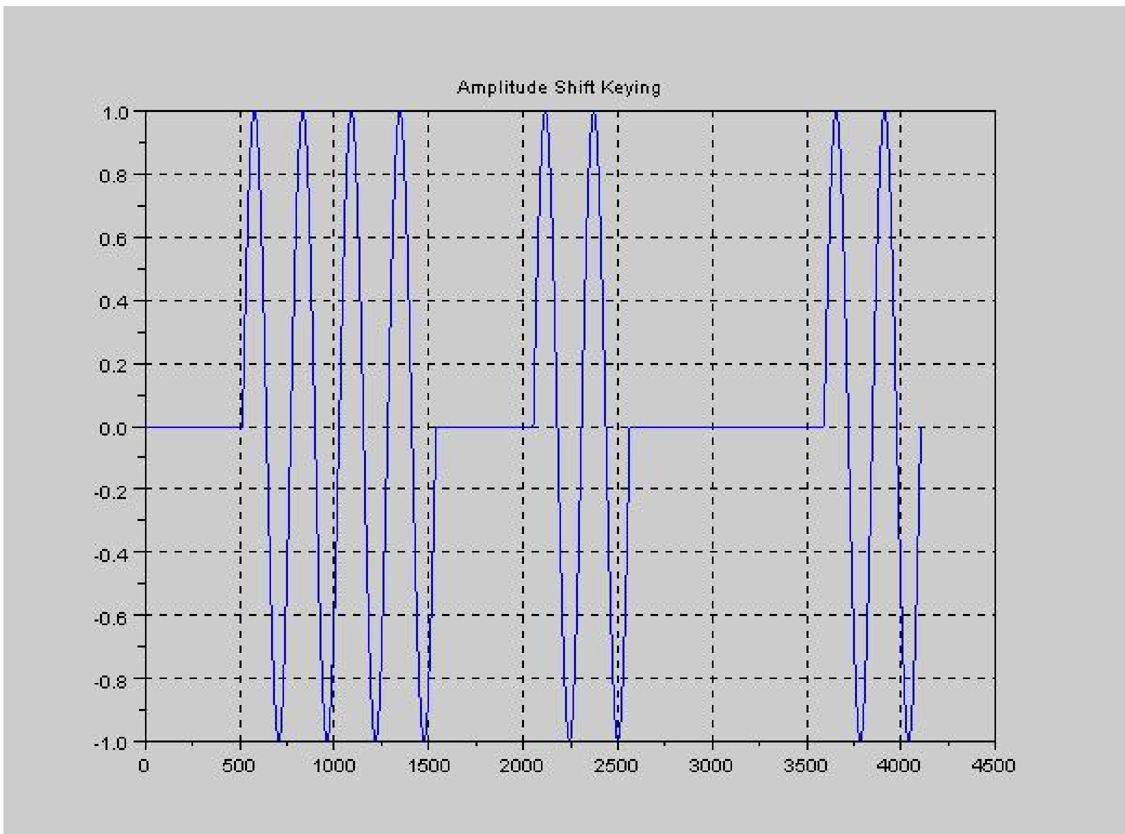

Figure 7.2: Figure7.1a

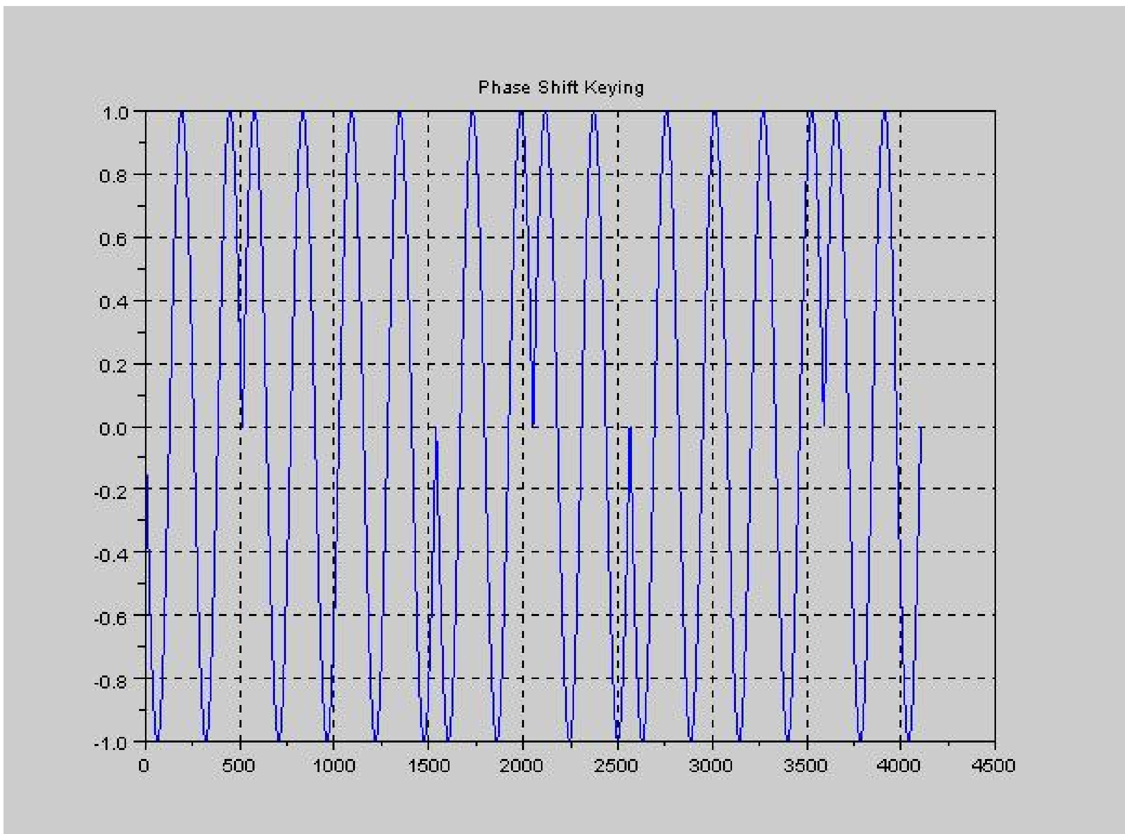

Figure 7.3: Figure7.1b

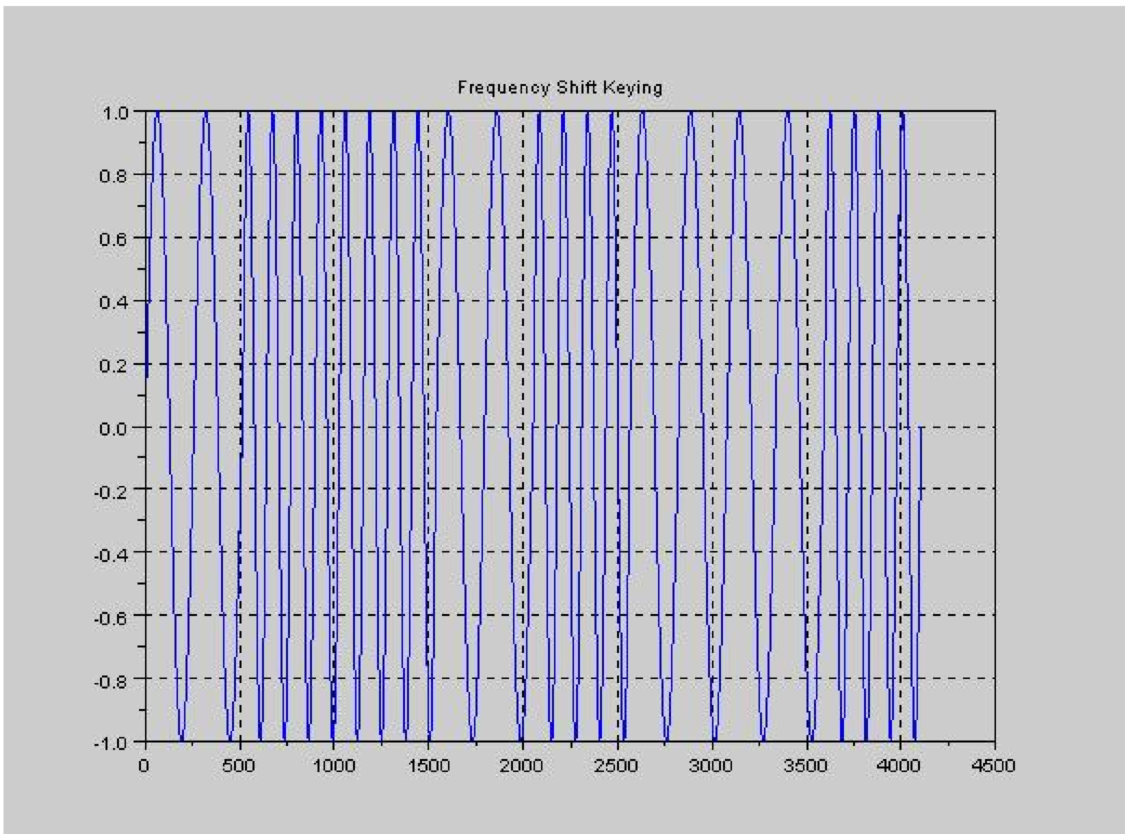

Figure 7.4: Figure7.1c

```
t2 = 0:0.01:2*Tb;
phi1 = cos(2*%pi*t1).* cos((%pi/(2*Tb))*t1);
phi2 = sin(2*%pi*t2).*sin((%pi/(2*Tb))*t2);
teta_0 = [0,%pi];
teta_tb = [%pi/2,-%pi/2];
S1 = [];
S2 = [];
for i = 1:M
  s1(i) = cos(teta_0(i));
  s2(i) = -sin(teta_tb(i));
  S1 = [S1 s1(i)*phi1];
  S2 = [S2 s2(1)*phi2];
end
for i = M:-1:1
  S1 = [S1 s1(i)*phi1];
  S2 = [S2 s2(2)*phi2];
end
Input_Sequence =[1,1,0,1,0,0,0];
S = [];
t = 0:0.01:1;
S = [S cos(0)*cos(2*%pi*t)-sin(%pi/2)*sin(2*%pi*t)];
S = [S cos(0)*cos(2*%pi*t)-sin(%pi/2)*sin(2*%pi*t)];
S = [S cos(%pi)*cos(2*%pi*t)-sin(%pi/2)*sin(2*%pi*t)];
S = [S cos(%pi)*cos(2*%pi*t)-sin(-%pi/2)*sin(2*%pi*t)];
S = [S cos(0)*cos(2*%pi*t)-sin(-%pi/2)*sin(2*%pi*t)];
S = [S cos(0)*cos(2*%pi*t)-sin(-%pi/2)*sin(2*%pi*t)];
S = [S cos(0)*cos(2*%pi*t)-sin(-%pi/2)*sin(2*%pi*t)];
y = [s1(1),s2(1);s1(2),s2(1);s1(2),s2(2);s1(1),s2(2)];
disp(y,'coordinates of message points')
figure
subplot(3,1,1)
a = gca();
```

```
a.x_location = "origin";
plot(S1)
title('Scaled time function s1*phi1(t)')
subplot(3,1,2)
a =gca();
a.x_location = "origin";
plot(S2)
title('Scaled time function s2*phi2(t)')
subplot(3,1,3)
a =gca();
a.x_location = "origin";
plot(S)
title('Obtained by adding s1*phi1(t)+s2*phi2(t) on a
    bit-by-bit basis')
```

Scilab code CF 7.2 Signal Space diagram for coherent BPSK

```
//Caption:Signal Space diagram for coherent BPSK
//Figure7.2 Signal Space Diagram for coherent BPSK
   system
clear
clc;
close;
M =2;
i = 1:M;
y = cos(2*%pi+(i-1)*%pi);
annot = dec2bin([length(y)-1:-1:0],log2(M));
disp(y,'coordinates of message points')
disp(annot,'Message points')
figure;
a =gca();
a.data_bounds = [-2,-2;2,2];
a.x_location = "origin";
a.y_location = "origin";
plot2d(real(y(1)),imag(y(1)),-9)
plot2d(real(y(2)),imag(y(2)),-5)
xlabel('
```

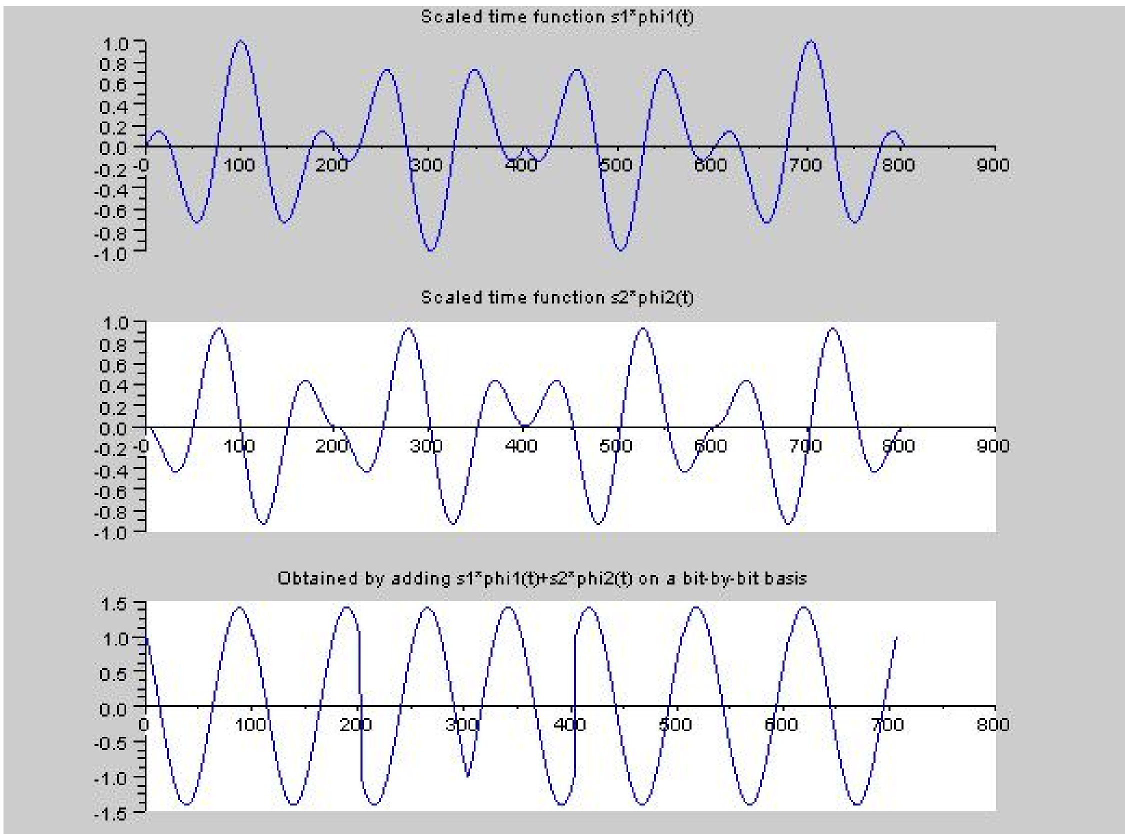

Figure 7.5: Example7.2

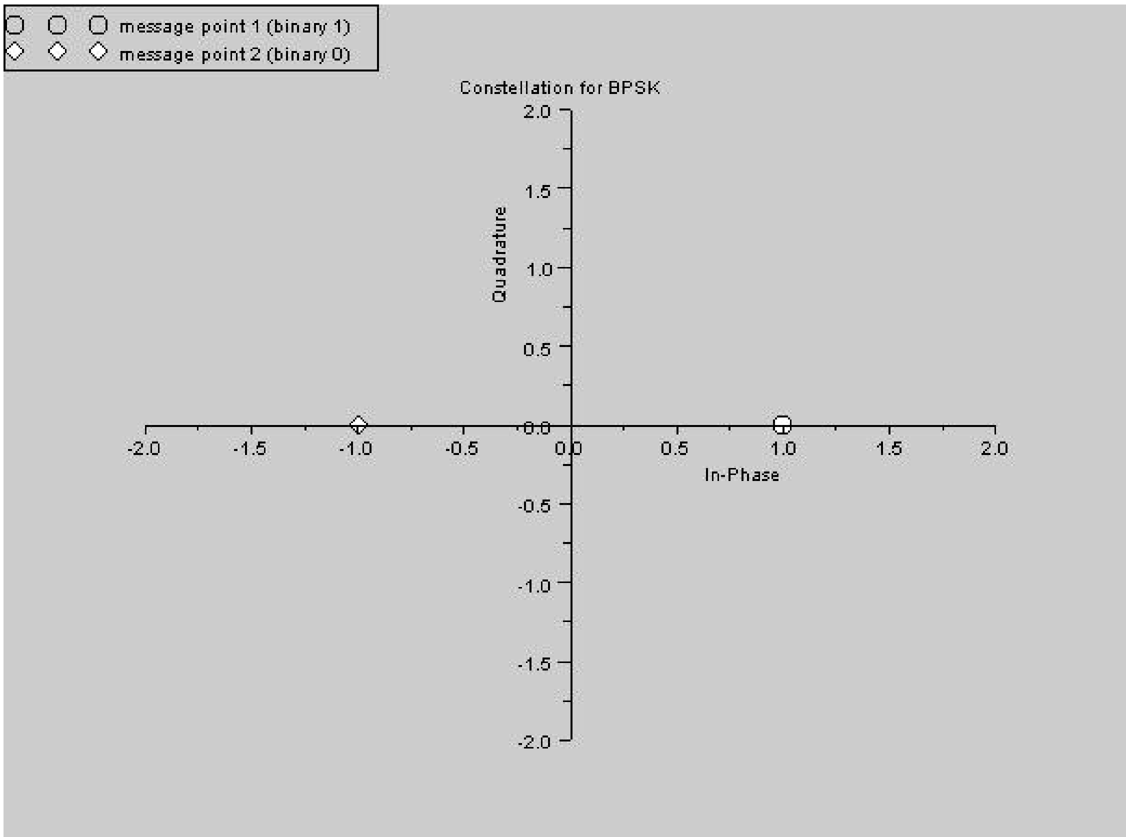

Figure 7.6: Figure7.2

```
    In-Phase');
ylabel('
    Quadrature');
title('Constellation for BPSK')
legend(['message point 1 (binary 1)';'message point
    2 (binary 0)'],5)
```

Scilab code Tab 7.3 Illustration the generation of DPSK signal

```
//Caption:Illustrating the generation of DPSK signal
```

```
//Table7.3 Generation of Differential Phase shift keying signal
clc;
bk = [1,0,0,1,0,0,1,1];//input digital sequence
for i = 1:length(bk)
  if(bk(i)==1)
    bk_not(i) =~1;
  else
    bk_not(i)= 1;
  end
end
dk_1(1) = 1&bk(1);  //initial value of differential encoded sequence
dk_1_not(1)=0&bk_not(1);
dk(1) = xor(dk_1(1),dk_1_not(1))//first bit of dpsk encoder
for i=2:length(bk)
  dk_1(i) = dk(i-1);
  dk_1_not(i) = ~dk(i-1);
  dk(i) = xor((dk_1(i)&bk(i)),(dk_1_not(i)&bk_not(i)));
end
for i =1:length(dk)
  if(dk(i)==1)
    dk_radians(i)=0;
  elseif(dk(i)==0)
    dk_radians(i)=%pi;
  end
end
disp('Table 7.3 Illustrating the Generation of DPSK Signal')
disp('------------------------------------------------------')
disp(bk,'(bk)')
bk_not = bk_not';
disp(bk_not,'(bk_not)')
dk = dk';
```

```
disp(dk,'Differentially encoded sequence (dk)')
dk_radians = dk_radians';
disp(dk_radians,'Transmitted phase in radians')
disp('
    ------------------------------------------------------
    ')
```

Scilab code CF 7.4 Signal Space diagram for coherent BFSK

```
//Caption:Signal Space diagram for coherent BFSK
//Figure7.4 Signal Space Diagram for coherent BFSK
    system
clear
clc;
close;
M =2;
y = [1,0;0,1];
annot = dec2bin([M-1:-1:0],log2(M));
disp(y,'coordinates of message points')
disp(annot,'Message points')
figure;
a =gca();
a.data_bounds = [-2,-2;2,2];
a.x_location = "origin";
a.y_location = "origin";
plot2d(y(1,1),y(1,2),-9)
plot2d(y(2,1),y(2,2),-5)
xlabel('

    In-Phase');
ylabel('

    Quadrature');
title('Constellation for BFSK')
legend(['message point 1 (binary 1)';'message point
    2 (binary 0)'],5)
```

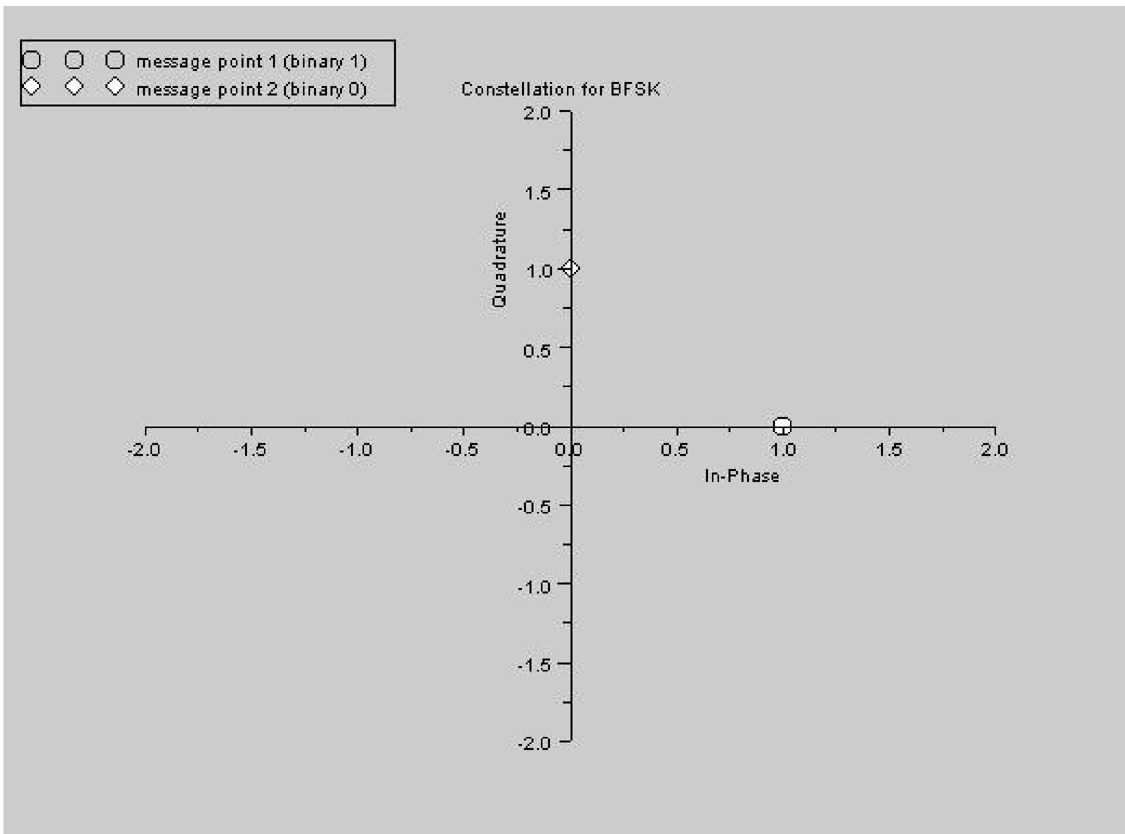

Figure 7.7: Figure 7.4

Scilab code CF 7.6 Signal space diagram for coherent QPSK waveform

```
//Caption:Signal space diagram for coherent QPSK waveform
//Figure7.6 Signal Space Diagram for coherent QPSK system
clear
clc;
close;
M =4;
i = 1:M;
y = cos((2*i-1)*%pi/4)-sin((2*i-1)*%pi/4)*%i;
annot = dec2bin([0:M-1],log2(M));
disp(y,'coordinates of message points')
disp(annot,'dibits value')
figure;
a =gca();
a.data_bounds = [-1,-1;1,1];
a.x_location = "origin";
a.y_location = "origin";
plot2d(real(y(1)),imag(y(1)),-2)
plot2d(real(y(2)),imag(y(2)),-4)
plot2d(real(y(3)),imag(y(3)),-5)
plot2d(real(y(4)),imag(y(4)),-9)
xlabel('                                        In-Phase');
ylabel('                                        Quadrature');
title('Constellation for QPSK')
legend(['message point 1 (dibit 10)';'message point 2 (dibit 00)';'message point 3 (dibit 01)';'message point 4 (dibit 11)'],5)
```

Scilab code Tab 7.6 Bndwidth efficiency of M ary PSK signals

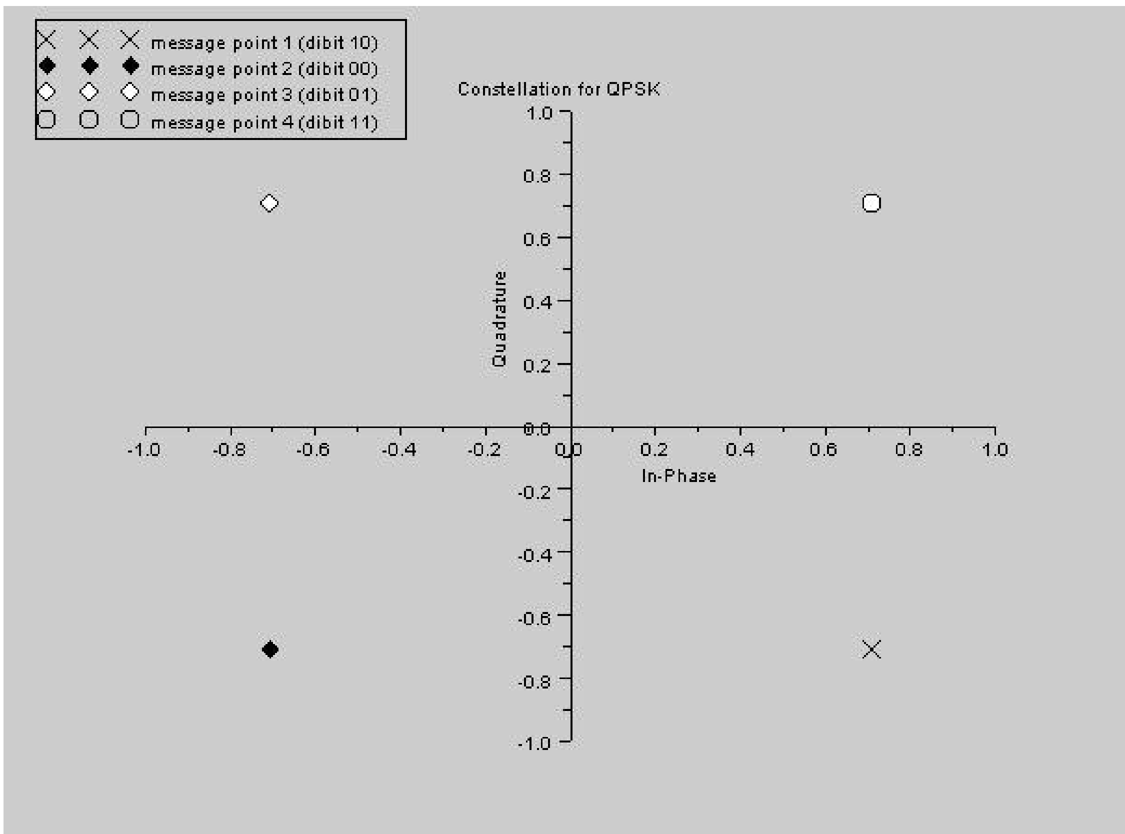

Figure 7.8: Figure7.6

```
//Caption:Bandwidth efficiency of M-ary PSK signals
//Table7.6: Bandwidth Efficiency of M-ary PSK
    signals
clear;
clc;
close;
M = [2,4,8,16,32,64];//M-ary
Ruo = log2(M)./2; //Bandwidth efficiency in bits/s/
    Hz
disp('Table 7.7 Bandwidth Efficiency of M-ary PSK
    signals')
disp('
    -----------------------------------------------------------
    ')
disp(M,'M')
disp('
    -----------------------------------------------------------
    ')
disp(Ruo,'r in bits/s/Hz')
disp('
    -----------------------------------------------------------
    ')
```

Scilab code Tab 7.7 Bandwidth efficiency of M ary FSK signals

```
//Caption:Bandwidth efficiency of M-ary FSK signals
//Table7.7: Bandwidth Efficiency of M-ary FSK
clear;
clc;
close;
M = [2,4,8,16,32,64];//M-ary
Ruo = 2*log2(M)./M; //Bandwidth efficiency in bits/s
    /Hz
//M = M';
//Ruo = Ruo';
disp('Table 7.7 Bandwidth Efficiency of M-ary FSK
    signals')
disp('
```

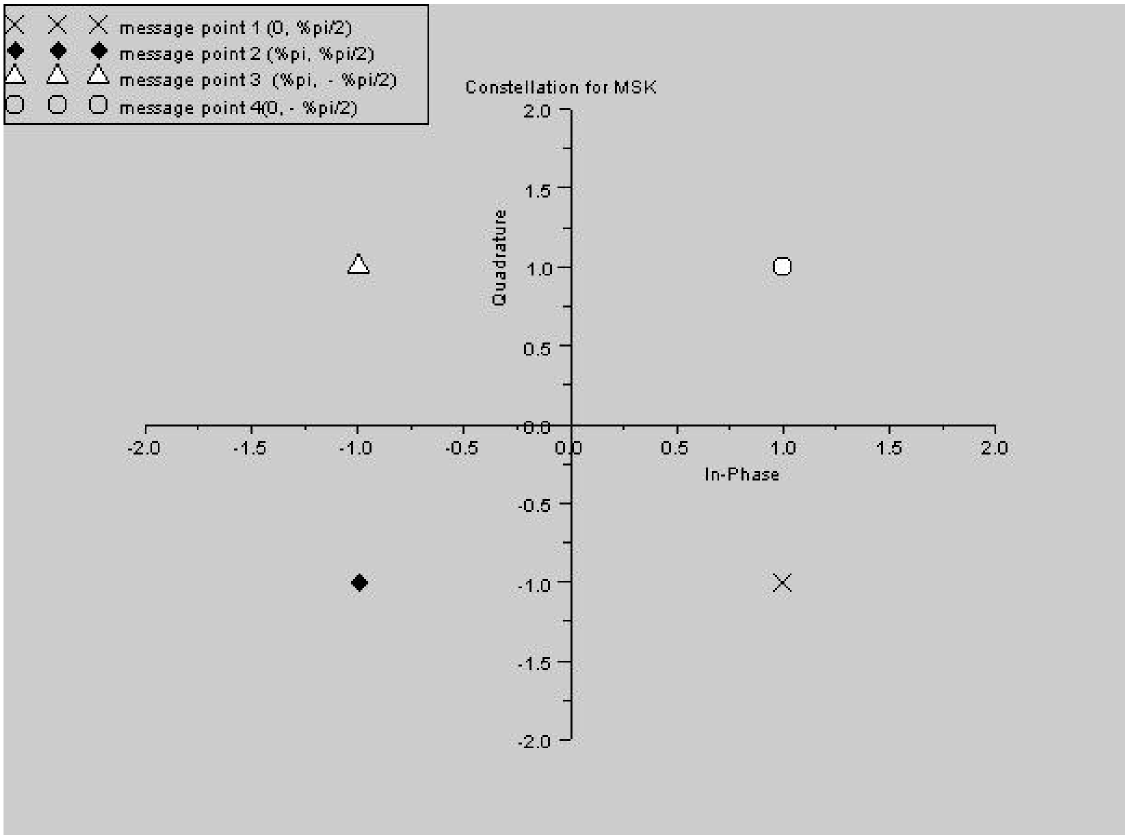

Figure 7.9: Figure7.12

```
    ------------------------------------------------------------
    ')
disp(M,'M')
disp('
    ------------------------------------------------------------
    ')
disp(Ruo,'r in bits/s/Hz')
disp('
    ------------------------------------------------------------
    ')
```

Scilab code CF 7.29 Power Spectra of BPSK and BFSK signals

```
//Caption:Power Spectra of BPSK and BFSK signals
//Figure7.29:Comparison of Power Spectral Densities of BPSK
//and BFSK
clc;
rb = input('Enter the bit rate=');
Eb = input('Enter the energy of the bit=');
f = 0:1/100:8/rb;
Tb = 1/rb;  //Bit duration
for i= 1:length(f)
   if(f(i)==(1/(2*Tb)))
     SB_FSK(i)=Eb/(2*Tb);
   else
     SB_FSK(i) = (8*Eb*(cos(%pi*f(i)*Tb)^2))/((%pi^2)*(((4*(Tb^2)*(f(i)^2))-1)^2));
   end
     SB_PSK(i)=2*Eb*(sinc_new(f(i)*Tb)^2);
end
a=gca();
plot(f*Tb,SB_FSK/(2*Eb))
plot(f*Tb,SB_PSK/(2*Eb))
poly1= a.children(1).children(1);
poly1.foreground = 6;
xlabel('Normalized Frequency ---->')
ylabel('Normalized Power Spectral Density--->')
title('PSK Vs FSK Power Spectra Comparison')
legend(['Frequency Shift Keying','Phase Shift Keying'])
xgrid(1)
//Result
//Enter the bit rate in bits per second:2
//Enter the Energy of bit:1
```

Scilab code CF 7.30 Power Spectra of QPSK and MSK signals.

```
//Caption:Power Spectra of QPSK and MSK signals
//Figure7.30:Comparison of QPSK and MSK Power
```

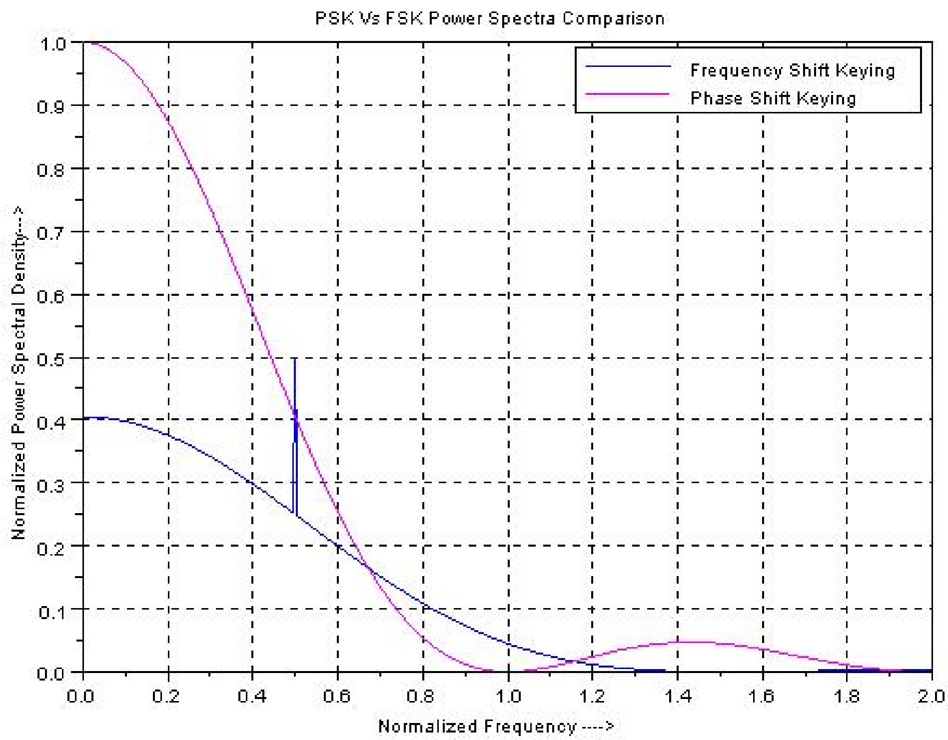

Figure 7.10: Figure7.29

```
   Spectrums
//clear;
//close;
//clc;
rb = input('Enter the bit rate in bits per second:')
   ;
Eb = input('Enter the Energy of bit:');
f = 0:1/(100*rb):(4/rb);
Tb = 1/rb; //bit duration in seconds
for i = 1:length(f)
  if(f(i)==0.5)
    SB_MSK(i) = 4*Eb*f(i);
  else
    SB_MSK(i) = (32*Eb/(%pi^2))*(cos(2*%pi*Tb*f(i))
       /((4*Tb*f(i))^2-1))^2;
  end
    SB_QPSK(i)= 4*Eb*sinc_new((2*Tb*f(i)))^2;
end
a = gca();
plot(f*Tb,SB_MSK/(4*Eb));
plot(f*Tb,SB_QPSK/(4*Eb));
poly1= a.children(1).children(1);
poly1.foreground = 3;
xlabel('Normalized Frequency ---->')
ylabel('Normalized Power Spectral Density--->')
title('QPSK Vs MSK Power Spectra Comparison')
legend(['Minimum Shift Keying','QPSK'])
xgrid(1)
//Result
//Enter the bit rate in bits per second:2
//Enter the Energy of bit:1
```

Scilab code CF 7.31 Power spectra of M-ary PSK signals

```
//Caption:Power spectra of M-ary PSK signals
//Figure7.31 Comparison of Power Spectral Densities
   of M-ary PSK signals
```

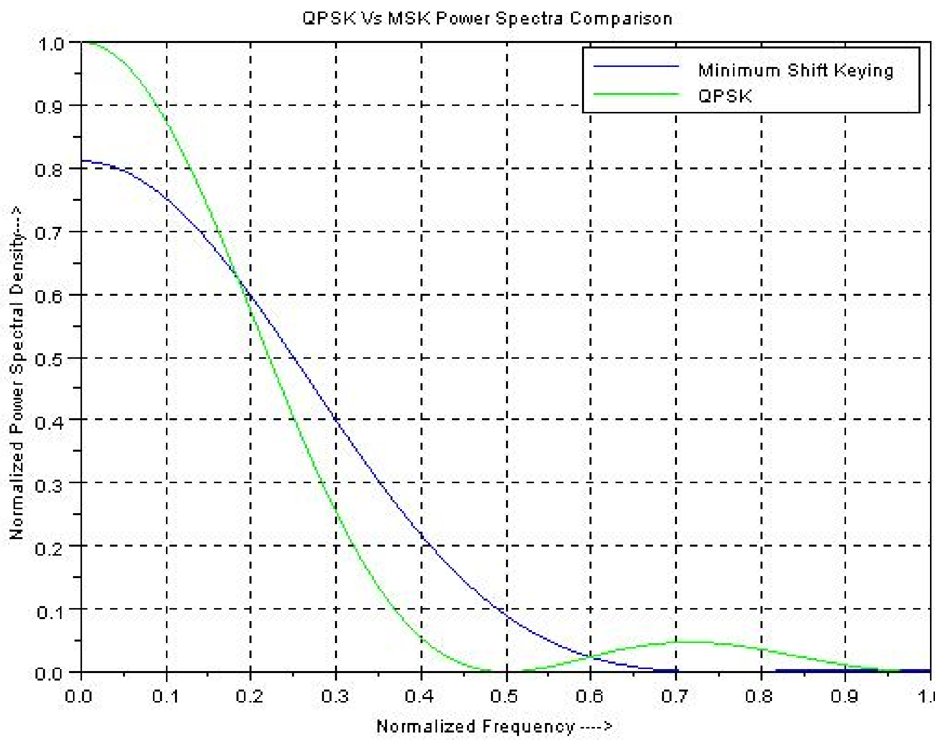

Figure 7.11: Figure7.30

```
rb = input('Enter the bit rate=');
Eb = input('Enter the energy of the bit=');
f = 0:1/100:rb;
Tb = 1/rb;  //Bit duration
M = [2,4,8];
for j = 1:length(M)
  for i= 1:length(f)
    SB_PSK(j,i)=2*Eb*(sinc_new(f(i)*Tb*log2(M(j)))
       ^2)*log2(M(j));
  end
end
a=gca();
plot2d(f*Tb,SB_PSK(1,:)/(2*Eb))
plot2d(f*Tb,SB_PSK(2,:)/(2*Eb),2)
plot2d(f*Tb,SB_PSK(3,:)/(2*Eb),5)
xlabel('Normalized Frequency ---->')
ylabel('Normalized Power Spectral Density--->')
title('Power Spectra of M-ary signals for M =2,4,8')
legend(['M=2','M=4','M=8'])
xgrid(1)
//Result
//Enter the bit rate in bits per second:2
//Enter the Energy of bit:1
```

Scilab code CF 7.41 Matched Filter output of rectangular pulse

```
//Caption:Matched Filter output of rectangular pulse
//Figure7.41
//Matched Filter Output
clear;
clc;
T =4;
a =2;
t = 0:T;
g = 2*ones(1,T+1);
h  =abs(convol(g,g));
for i = 1:length(h)
```

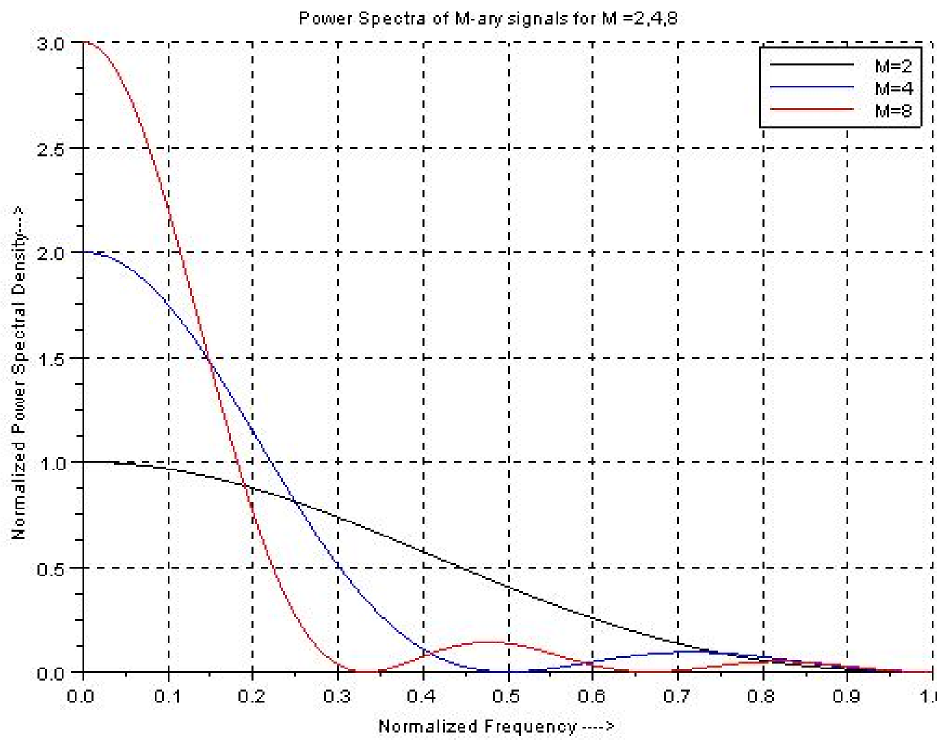

Figure 7.12: Figure7.31

```
  if(h(i)<0.01)
    h(i) =0;
  end
end
h = h-T;
t1 = 0:length(h)-1;
figure
a =gca();
a.data_bounds = [0,0;6,4];
plot2d(t,g,5)
xlabel('t--->')
ylabel('g(t)----->')
title('Rectangular pulse duration T = 4, a =2')
figure
plot2d(t1,h,6)
xlabel('t--->')
ylabel('Matched Filter output')
title('Output of filter matched to rectangular pulse
    g(t)')
```

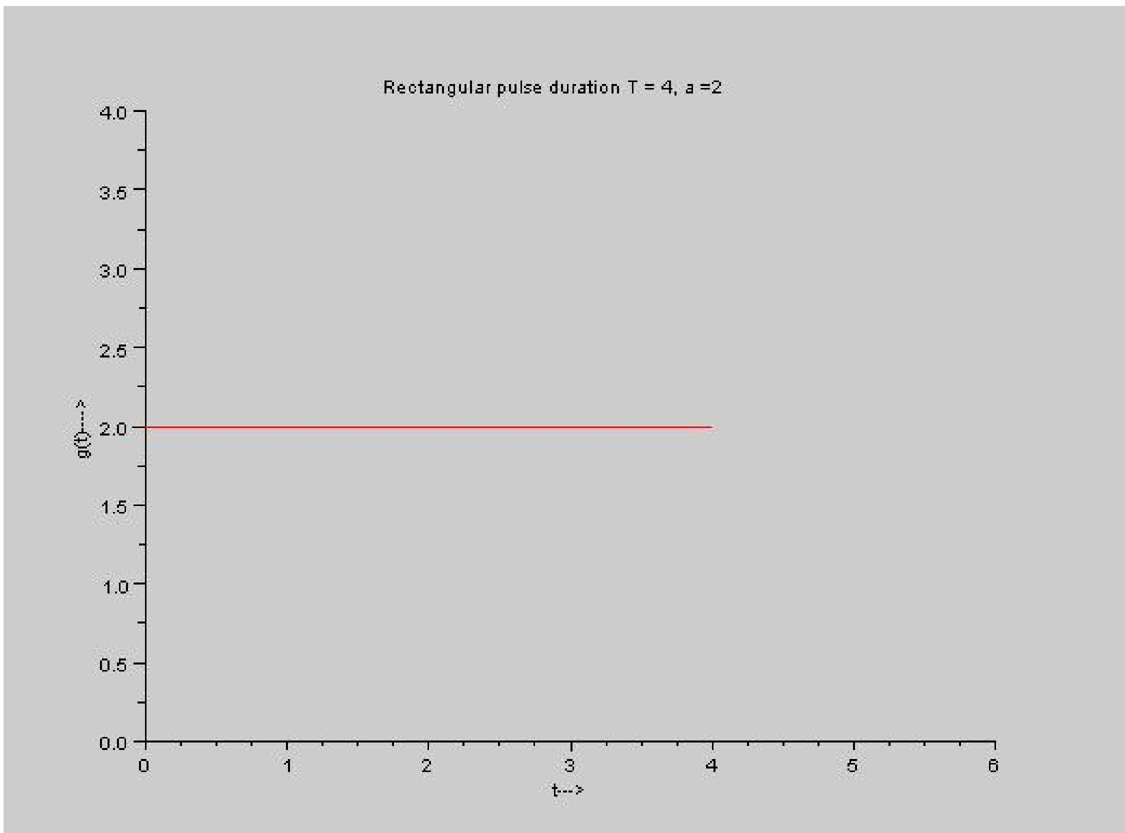

Figure 7.13: Figure7.41a

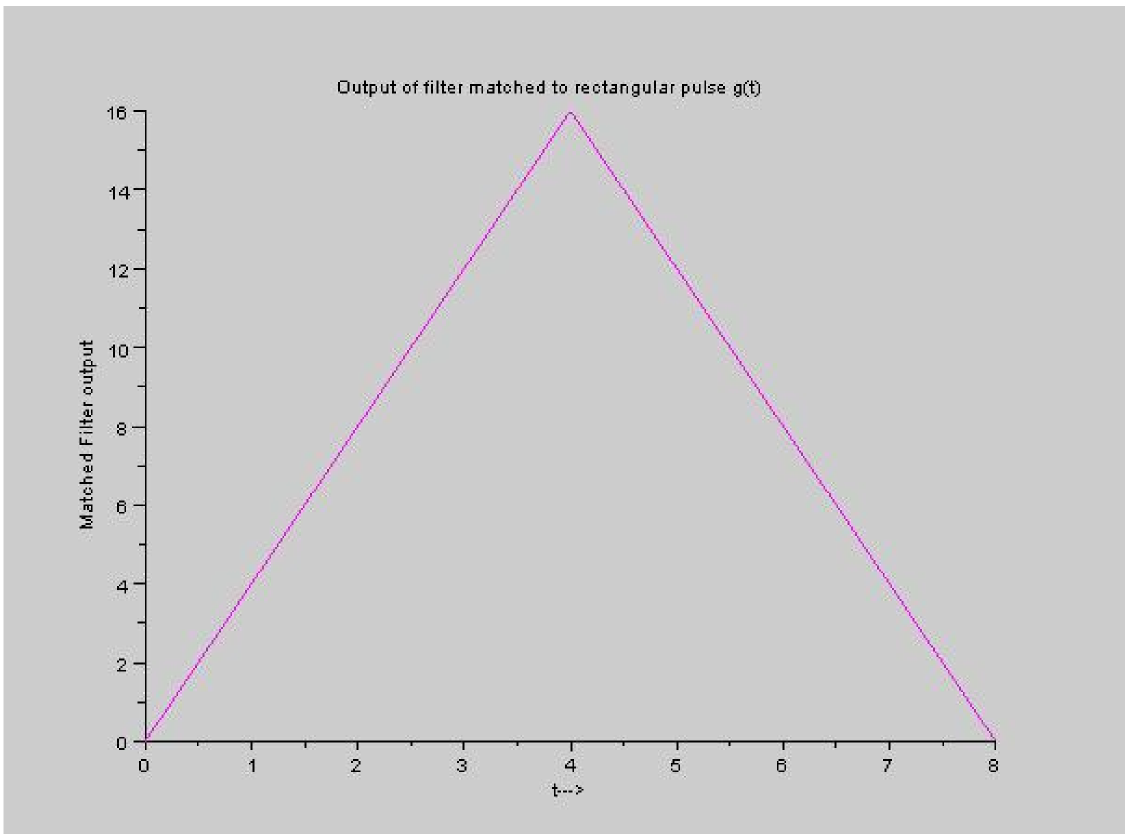

Figure 7.14: Figure7.41b

Chapter 8

Error-Control Coding

Scilab code Exa 8.1 Repetition Codes

```
//Caption:Repetition Codes
//Example8.1:Repetition Codes
clear;
clc;
n =5; //block of identical 'n' bits
k =1; //one bit
m = 1;// bit value = 1
I = eye(n-k,n-k);//Identity matrix
P = ones(1,n-k);//coefficient matrix
H = [I P'];//parity-check matrix
G = [P 1];//generator matrix
x = m.*G; //code word
disp(G,'generator matrix');
disp(H,'parity-check matrix');
disp(x,'code word for binary one input');
```

Scilab code Exa 8.2 Hamming Codes

```
//Caption:Hamming Codes
//Example8.2:Hamming codes
clear;
clc;
k = 4; //message bits length
```

```
n = 7; //block length
m = n-k;//Number of parity bits
I = eye(k,k); //identity matrix
disp(I,'identity matrix Ik')
P =[1,1,0;0,1,1;1,1,1;1,0,1];//coefficient matrix
disp(P,'coefficient matrix P')
G = [P I]; //generator matrix
disp(G,'generator matrix G')
H = [eye(k-1,k-1) P']://parity check matrix
disp(H,'parity chechk matrix H')
//message bits
m =
   [0,0,0,0;0,0,0,1;0,0,1,0;0,0,1,1;0,1,0,0;0,1,0,1;0,1,1,0;0,1,1,1;

//
C = m*G;
C = modulo(C,2);
disp(C,'Code words of (7,4) Hamming code')
```

Scilab code Exa 8.3 Hamming Codes Revisited

```
//Caption:Hamming Codes Revisited
//Example8.3:(7,4) Hamming Code Revisited
//message sequence = [1,0,0,1]
//D = poly(0,D);
clc;
D = poly(0,'D');
g = 1+D+0+D^3; //generator polynomial
m = (D^3)*(1+0+0+D^3); //message sequence
[r,q] = pdiv(m,g);
p = coeff(r);
disp(r,'remainder in polynomial form')
disp(p,'Parity bits are:')
G = [g;g*D;g*D^2;g*D^3];
G = coeff(G);
disp(G,'G')
G(3,:) = G(3,:)+G(1,:);
G(3,:) = modulo(G(3,:),2);
```

```
G(4,:) = G(1,:)+G(2,:)+G(4,:);
G(4,:) = modulo(G(4,:),2);
disp(G,'Generator Matrix G =')
h = 1+D^-1+D^-2+D^-4;
H_D = [D^4*h;D^5*h;D^6*h];
H_num =numer(H_D);
H = coeff(H_num);
H(1,:) =H(1,:)+H(3,:);
H(1,:) = modulo(H(1,:),2);
disp(H,'Partiy Check matrix H =')
```

Scilab code Exa 8.4 Encoder for the (7,4) Cyclic Hamming Code

```
//Caption:Encoder for the (7,4) Cyclic Hamming Code
//Example8.4:Encoder for the (7,4) Cyclic hamming
    code
//message sequence = [1,0,0,1]
//D = poly(0,D);
D = poly(0,'D');
g = 1+D+0+D^3; //generator polynomial
m = (D^3)*(1+0+0+D^3); //message sequence
[r,q] = pdiv(m,g);
p = coeff(r);
disp(r,'remainder in polynomial form')
disp(p,'Parity bits are:')
disp('Table 8.3 Contents of the Shift Register in
    the Encoder of fig8.7 for Message Sequence(1001)'
    )
disp('
    ----------------------------------------------------------------
    ')
disp('Shift                 Input                 Register
    Contents')
disp('
    ----------------------------------------------------------------
    ')
disp('1                     1                     1 1 0')
disp('2                     0                     0 1 1')
```

```
disp('3                    0                    1  1  1')
disp('4                    1                    0  1  1')
disp('-----------------------------------------------------------------')
```

Scilab code Exa 8.5 Syndrome calculator for the(7,4) Cyclic Hamming Code

```
//Caption:Syndrome calculator for the(7,4) Cyclic Hamming Code
//Example8.5: Syndrome calculator
//message sequence = [0,1,1,1,0,0,1]
clc;
D = poly(0,'D');
g = 1+D+0+D^3; //generator polynomial
C1 = 0+D+D^2+D^3+0+0+D^6;//error free codeword
C2 = 0+D+D^2+0+0+0+D^6;//middle bit is error
[r1,q1] = pdiv(C1,g);
S1 = coeff(r1);
S1 = modulo(S1,2);
disp(r1,'remainder in polynomial form')
disp(S1,'Syndrome bits for error free codeword are:')
[r2,q2] = pdiv(C2,g);
S2 = coeff(r2);
S2 = modulo(S2,2);
disp(r2,'remainder in polynomial form for errored codeword')
disp(S2,'Syndrome bits for errored  codeword are:')
```

Scilab code Exa 8.6 Reed-Solomon Codes

```
//Caption:Reed-Solomon Codes
//Example8.6: Reed-Solomon Codes
//Single-error-correcting RS code with a 2-bit byte
clc;
m =2; //m-bit symbol
```

```
k = 1^2; //number of message bits
t =1; //single bit error correction
n = 2^m-1; //code word length in 2-bit byte
p = n-k; //parity bits length in  2-bit byte
r = k/n; //code rate
disp(n,'n')
disp(p,'n-k')
disp(r,'Code rate:r = k/n =')
disp(2*t,'It can correct any error upto =')
```

Scilab code Exa 8.7 Convolutional Encoding - Time domain approach

```
//Caption:Convolutional Encoding - Time domain
   approach
//Example8.7:Convolutional Code Generation
//Time Domain Approach
close;
clc;
g1 = input('Enter the input Top Adder Sequence:=')
g2 = input('Enter the input Bottom Adder Sequence:='
   )
m = input('Enter the message sequence:=')
x1 = round(convol(g1,m));
x2 = round(convol(g2,m));
x1 = modulo(x1,2);
x2 = modulo(x2,2);
N = length(x1);
for i =1:length(x1)
  x(i,:) =[x1(N-i+1),x2(N-i+1)];
end
x = string(x)
disp(x)
//Result
//Enter the input Top Adder Sequence:=[1,1,1]
//Enter the input Bottom Adder Sequence:=[1,0,1]
//Enter the message sequence:=[1,1,0,0,1]
//x =
//!1  1  !
```

```
//!        !
//!1   0   !
//!        !
//!1   1   !
//!        !
//!1   1   !
//!        !
//!0   1   !
//!        !
//!0   1   !
//!        !
//!1   1   !
```

Scilab code Exa 8.8 Convolutional Encoding Transform domain approach

```
//Caption:Convolutional Encoding   Transform domain approach
//Example8.8:Convolutional code - Transform domain approach
clc;
D = poly(0,'D');
g1D = 1+D+D^2; //generator polynomial 1
g2D = 1+D^2; //generator polynomial 2
mD = 1+0+0+D^3+D^4; //message sequence polynomial representation
x1D = g1D*mD; //top output polynomial
x2D = g2D*mD; //bottom output polynomial
x1 = coeff(x1D);
x2 = coeff(x2D);
disp(modulo(x1,2),'top output sequence')
disp(modulo(x2,2),'bottom output sequence')
//Result
//top output sequence
//     1.     1.     1.     1.     0.     0.     1.
//
// bottom output sequence
//     1.     0.     1.     1.     1.     1.     1.
```

Scilab code Exa 8.11 Fano metric for binary symmetric channel using convolutional code

```
//Caption:Fano metric for binary symmetric channel
    using convolutional code
//Example8.11: Convolutional code for binary
    symmetric channel
clc;
r = 1/2; //code rate
n =2; //number of bits
pe = 0.04; //transition probility
p = 1-pe;// probability of correct reception
gama_1 = 2*log2(p)+2*(1-r); //branch metric for
    correct reception
gamma_2 = log2(pe*p)+1; //branch metric for any one
    correct recption
gamma_3 = 2*log2(pe)+1; //branch metric for no
    correct reception
disp(gama_1,'branch metric for correct reception')
disp(gamma_2,'branch metric for any one correct
    recption')
disp(gamma_3,'branch metric for no correct reception
    ')
//branch metric for correct reception
//      0.8822126
//  branch metric for any one correct recption
//    - 3.7027499
//  branch metric for no correct reception
//    - 8.2877124
```

Chapter 9

Spread-Spectrum Modulation

Scilab code Exa 9.1 PN sequence generation

```
//Caption:PN sequence generation
//Example9.1 and Figure9.1: Maximum-length sequence generator
//Program to generate Maximum Length Pseudo Noise Sequence
//Period of PN Sequence N = 7
clc;
//Assign Initial value for PN generator
x0= 1;
x1= 0;
x2 =0;
x3 =0;
N = input('Enter the period of the signal')
for i =1:N
  x3 =x2;
  x2 =x1;
  x1 = x0;
  x0 =xor(x1,x3);
  disp(i,'The PN sequence at step')
  x = [x1 x2 x3];
  disp(x,'x=')
end
m = [7,8,9,10,11,12,13,17,19];
```

```
N = 2^m-1;
disp('Table 9.1 Range of PN Sequence lengths')
disp('----------------------------------------------------------')
disp('Length of shift register (m)')
disp(m)
disp('PN sequence Length (N)')
disp(N)
disp('----------------------------------------------------------')
//RESULTEnter the period of the signal 7
//  The PN sequence at step   1.
//  x=        1.     0.     0.
//  The PN sequence at step   2.
//  x=        1.     1.     0.
//  The PN sequence at step   3.
//  x=        1.     1.     1.
//  The PN sequence at step   4.
//  x=        0.     1.     1.
//  The PN sequence at step   5.
//  x=        1.     0.     1.
//  The PN sequence at step   6.
//  x=        0.     1.     0.
//  The PN sequence at step   7.
//  x=        0.     0.     1.
```

Scilab code Exa 9.2 Maximum length sequence property

```
//Caption:Maximum length sequence property
//Example9.2 and Figure 9.2: Maximum-length sequence
//Period of PN Sequence N = 7
//Properites of maximum-length sequence
clc;
//Assign Initial value for PN generator
x0= 1;
x1= 0;
```

```
x2 =0;
x3 =0;
N = input('Enter the period of the signal')
one_count = 0;
zero_count = 0;
for i =1:N
  x3 =x2;
  x2 =x1;
  x1 = x0;
  x0 =xor(x1,x3);
  disp(i,'The PN sequence at step')
  x = [x1 x2 x3];
  disp(x,'x=')
  C(i) = x3;
  if(C(i)==1)
    C_level(i)=1;
    one_count = one_count+1;
  elseif(C(i)==0)
    C_level(i)=-1;
    zero_count = zero_count+1;
  end
end
disp(C,'Output Sequence')//refer equation 9.4
disp(C_level,'Output Sequence levels')//refer equation 9.5
if(zero_count < one_count)
  disp(one_count,'Number of 1s in the given PN sequence')
  disp(zero_count,'Number of 0s in the given PN sequence')
  disp('Property 1 (Balance property) is satisified')
end
Rc_tuo = corr(C_level,N);
t = 1:2*length(C_level);
//
figure
a =gca();
```

```
a.x_location = "origin";
plot2d(t,[C_level; C_level])
xlabel('                              t')
title('Waveform of maximum-length sequence [0 0 1 1 1 0 1 0 0 1 1 1 0 1]')
//
figure
a =gca();
a.x_location ="origin";
a.y_location ="origin";
plot2d([-length(Rc_tuo)+1:-1,0:length(Rc_tuo)-1],[Rc_tuo($:-1:2),Rc_tuo],5)
xlabel('

    tuo')
ylabel('

    Rc(tuo)')
title('Autocorrelation of maximum-length sequence')
```

Scilab code Exa 9.3 Processing gain, PN sequence length, Jamming margin in dB

```
//Caption:Processing gain, PN sequence length, Jamming margin in dB
//Example9.3: Processing gain and Jamming Margin
clear;
clc;
close;
Tb = 4.095*10^-3;//Information bit duration
Tc = 1*10^-6;//PN chip duration
PG = Tb/Tc;//Processing gain
disp(PG,'The processing gain is:')
N = PG; //PN sequence length
m = log2(N+1);//feedback shift register length
disp(N,'The required PN sequence is:')
disp(m,'The feedback shift register length:')
```

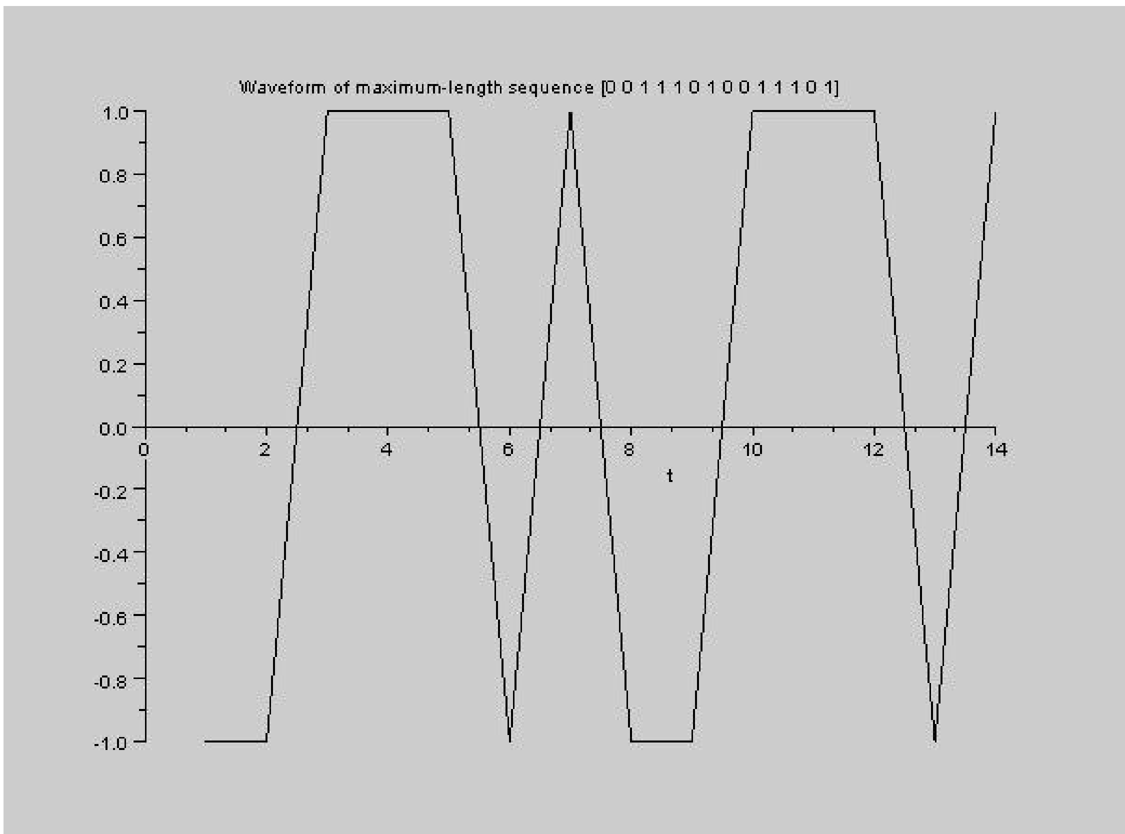

Figure 9.1: Example9.2a

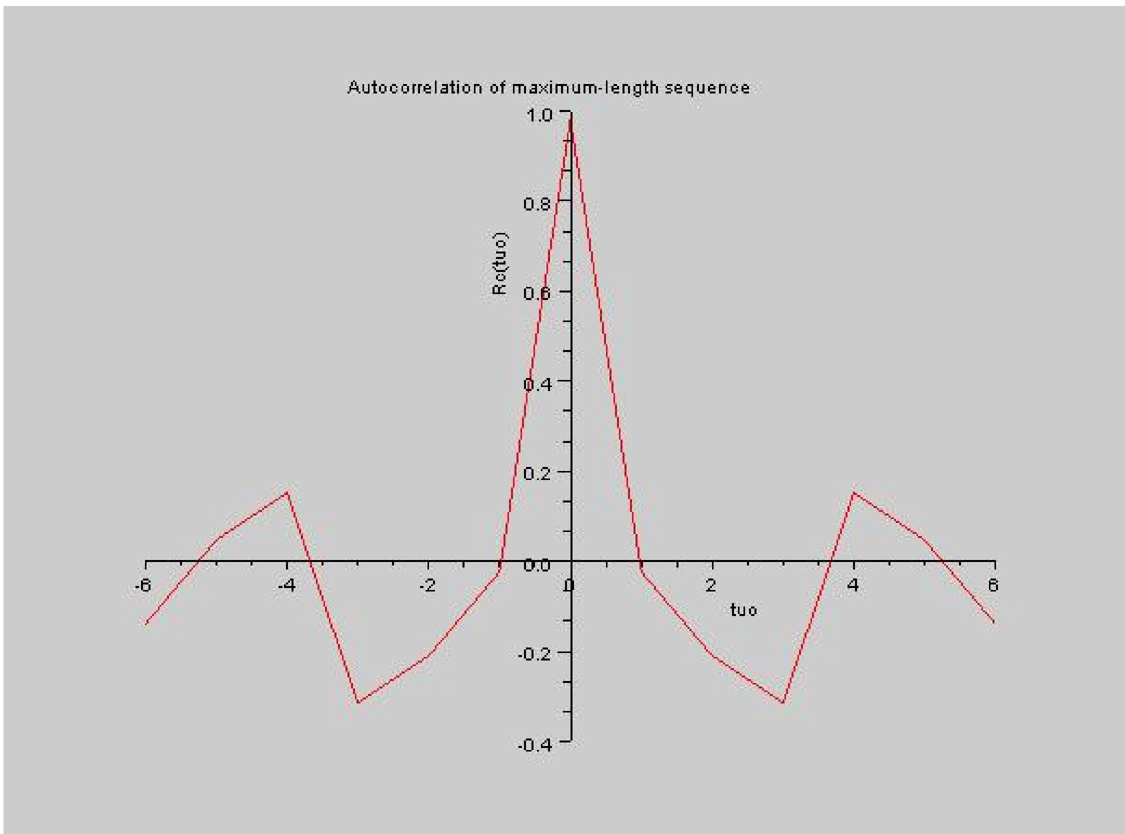

Figure 9.2: Example9.2b

```
Eb_No = 10;//Energy to noise density ratio
J_P = PG/Eb_No;//Jamming Margin
disp(10*log10(J_P),'Jamming Margin in dB:')
//Result
//The processing gain is:  4095.
//The required PN sequence is: 4095.
//The feedback shift register length: 12.
//Jamming Margin in dB: 26.122539
```

Scilab code Exa 9.4Example9.5 Slow and Fast Frequency hopping: FH/MFSK

```
//Caption:Slow and Fast Frequency hopping: FH/MFSK
//Example9.4 and Example9.5: Parameters of FH/MFSK signal
//Slow and Fast Frequency Hopping
clear;
close;
clc;
K =2; //number of bits per symbol
M = 2^K; //Number of MFSK tones
N = 2^M-1;//Period of the PN sequence
k = 3; //length of PN sequence per hop
disp(K,'number of bits per symbol K =')
disp(M,'Number of MFSK tones M=')
disp(N,'Period of the PN sequence N =')
disp(k,'length of PN sequence per hop k =')
disp(2^k,'Total number of frequency hops =')
//Result
//number of bits per symbol K = 2.
//Number of MFSK tones M = 4.
//Period of the PN sequence N = 15.
//length of PN sequence per hop k = 3.
//Total number of frequency hops =  8.
```

Scilab code Fig 9.4Figure9.6 Direct Sequence Spread Coherent BPSK

```
//Caption:Direct Sequence Spread Coherent BPSK
```

```
//Figure 9.4:Generation of waveforms in DS/BPSK
   spread spectrum transmitter
clear;
close;
clc;
t = 0:13;
N = 7;
wt = 0:0.01:1;
bt = [1*ones(1,N) -1*ones(1,N)];
ct = [0,0,1,1,1,0,1,0,0,1,1,1,0,1];
ct_polar = [-1,-1,1,1,1,-1,1,-1,-1,1,1,1,-1,1];
mt = bt.*ct_polar;
Carrier = 2*sin(wt*2*%pi);
st = [];
for i = 1:length(mt)
  st = [st mt(i)*Carrier];
end
//
figure
subplot(3,1,1)
a =gca();
a.x_location = "origin";
a.y_location = "origin";
a.data_bounds = [0,-2;20,2];
plot2d2(t,bt,5)
xlabel('

   t')
title('Data b(t)')
subplot(3,1,2)
a =gca();
a.x_location = "origin";
a.y_location = "origin";
a.data_bounds = [0,-2;20,2];
plot2d2(t,ct_polar,5)
xlabel('

   t')
```

```
title('Spreading code c(t)')
subplot(3,1,3)
a =gca();
a.x_location = "origin";
a.y_location = "origin";
a.data_bounds = [0,-2;20,2];
plot2d2(t,mt,5)
xlabel('

   t')
title('Product Signal m(t)')
//
figure
subplot(3,1,1)
a =gca();
a.x_location = "origin";
a.y_location = "origin";
a.data_bounds = [0,-2;20,2];
plot2d2(t,mt,5)
xlabel('

   t')
title('Product Signal m(t)')
subplot(3,1,2)
a =gca();
a.x_location = "origin";
a.y_location = "origin";
a.data_bounds = [0,-2;20,2];
plot(Carrier)
xlabel('

   t')
title('Carrier Signal')
subplot(3,1,3)
a =gca();
a.x_location = "origin";
a.y_location = "origin";
a.data_bounds = [0,-2;20,2];
```

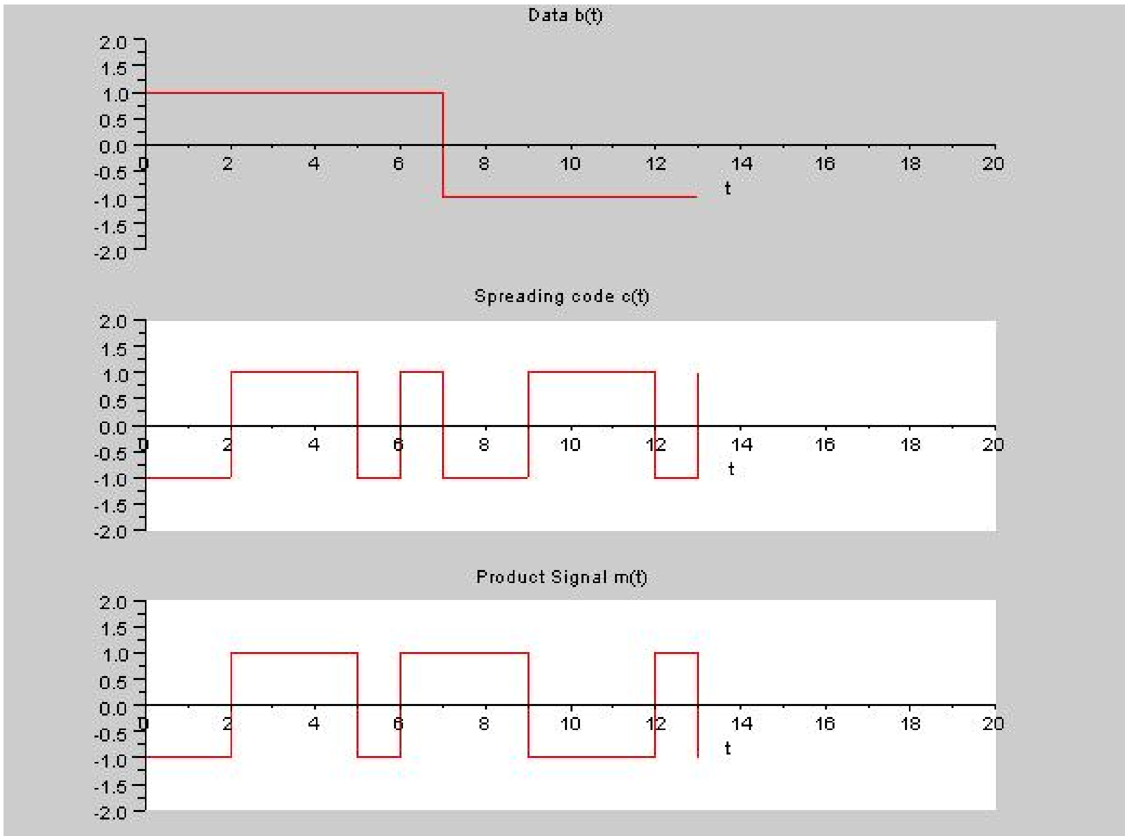

Figure 9.3: Figure9.6a

```
plot(st)
xlabel('

    t')
title('DS/BPSK signal s(t)')
//
```

Scilab code ARC 1 Alaw

```
function [Cx,Xmax] = Alaw(x,A)
  //Non-linear Quantization
  //A-law: A-law nonlinear quantization
```

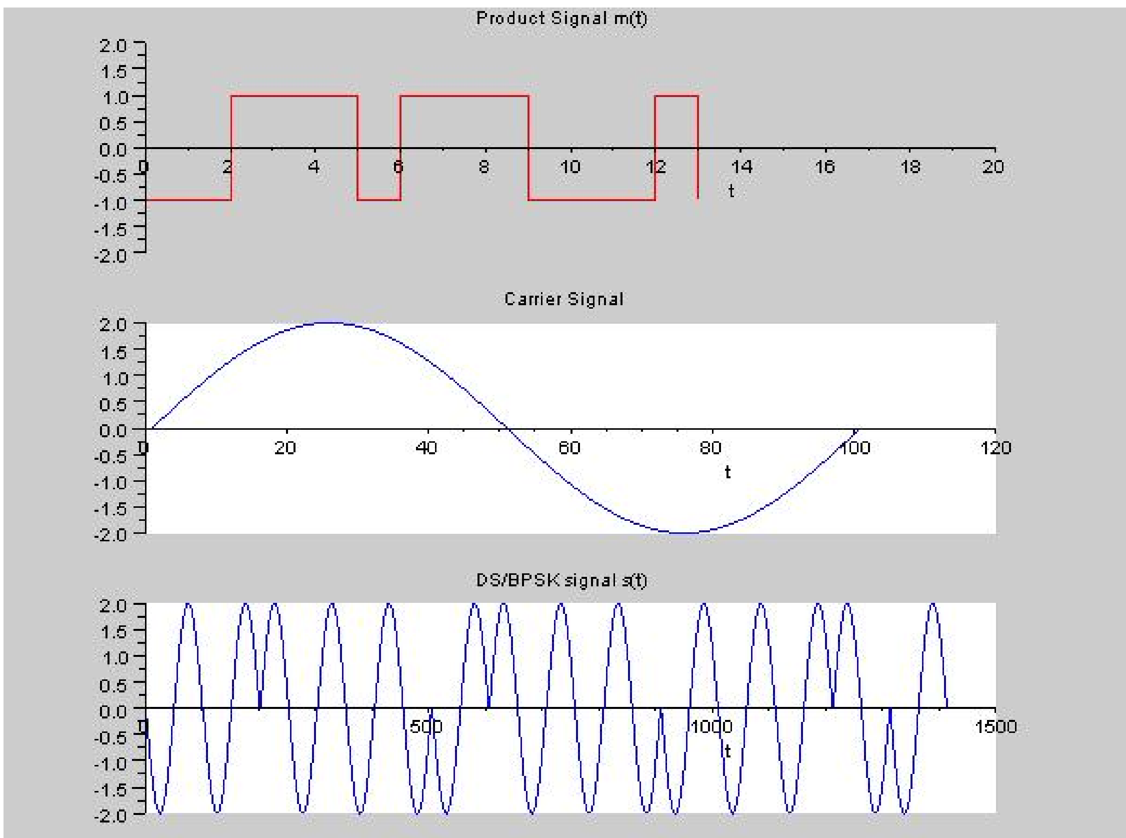

Figure 9.4: Figure9.6b

Figure9.6b

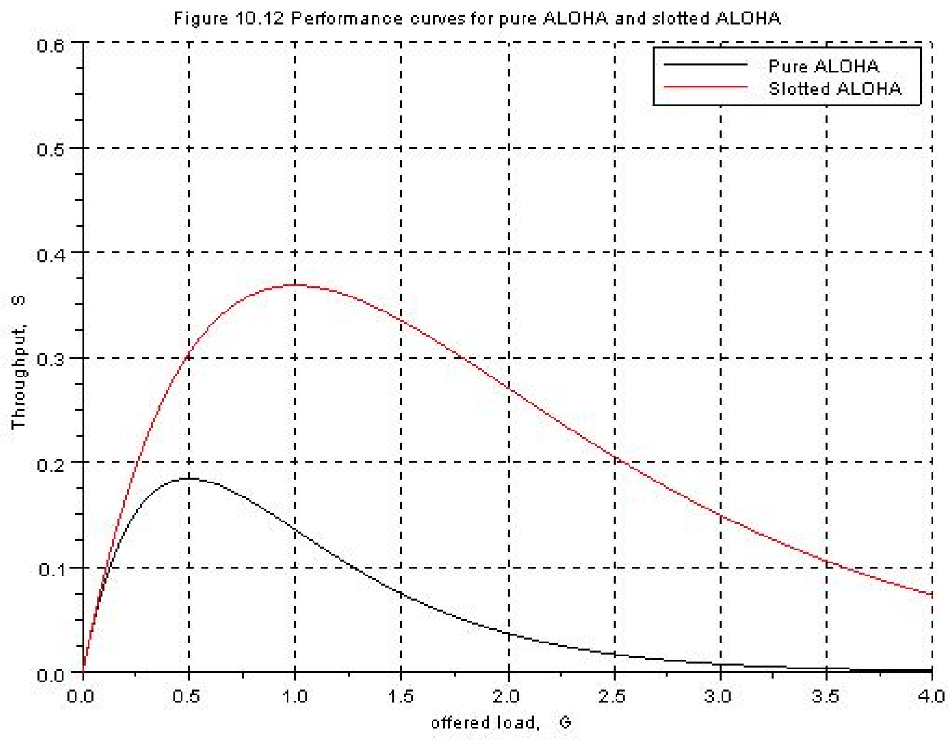

Figure 9.5: Figure10.12

```
  //x = input vector
  //Cx = A-law compressor output
  //Xmax = maximum of input vector x
  Xmax  = max(abs(x));
  for i = 1:length(x)
    if(x(i)/Xmax < = 1/A)
       Cx(i) = A*abs(x(i)/Xmax)./(1+log(A));
    elseif(x(i)/Xmax > 1/A)
      Cx(i) = (1+log(A*abs(x(i)/Xmax)))./(1+log(A));
    end
  end
  Cx = Cx/Xmax; //normalization of output vector
  Cx = Cx';
endfunction
```

Scilab code ARC 2 auto correlation

```
function [Rxx] = auto_correlation(x)
//Autocorrelation of a given Input Sequence
//Finding out the period of the signal using autocorrelation technique

L = length(x);
h = zeros(1,L);
for i = 1:L
  h(L-i+1) = x(i);
end
N = 2*L-1;
Rxx = zeros(1,N);
for i = L+1:N
   h(i) = 0;
end
for i = L+1:N
    x(i) = 0;
end
for n = 1:N
  for k = 1:N
    if(n >= k)
```

```
      Rxx(n) = Rxx(n)+x(n-k+1)*h(k);
    end
  end
end
disp('Auto Correlation Result is')
Rxx
disp('Center Value is the Maximum of autocorrelation result')
[m,n] = max(Rxx)
disp('Period of the given signal using Auto Correlation Sequence')
n
endfunction
```

Scilab code ARC 3 Convolutional Coding

```
//Caption:Convolutional Code Generation
//Time Domain Approach
close;
clc;
g1 = input('Enter the input Top Adder Sequence:=')
g2 = input('Enter the input Bottom Adder Sequence:=')
m = input('Enter the message sequence:=')
x1 = round(convol(g1,m));
x2 = round(convol(g2,m));
x1 = modulo(x1,2);
x2 = modulo(x2,2);
N = length(x1);
for i =1:length(x1)
  x(i,:) =[x1(N-i+1),x2(N-i+1)];
end
x = string(x)
//Result
//Enter the input Top Adder Sequence:=[1,1,1]
//Enter the input Bottom Adder Sequence:=[1,0,1]
//Enter the message sequence:=[1,1,0,0,1]
//x =
```

```
//!1   1   !
//!         !
//!1   0   !
//!         !
//!1   1   !
//!         !
//!1   1   !
//!         !
//!0   1   !
//!         !
//!0   1   !
//!         !
//!1   1   !
```

Scilab code ARC 4 Hamming Distance

```
//Caption: Hamming Weight and Hamming Distance
//H(7,4)
//Code Word Length = 7, Message Word length = 4,
   Parity bits =3
//clear;
close;
clc;
//Getting Code Words
code1 = input('Enter the first code word');
code2 =  input('Enter the second code word');
Hamming_Distance = 0;
for i = 1:length(code1)
  Hamming_Distance =Hamming_Distance+xor(code1(i),
     code2(i));
end
disp(Hamming_Distance,'Hamming Distance')
//Result
//Enter the first code word [0,1,1,1,0,0,1]
//Enter the second code word[1,1,0,0,1,0,1]
//Hamming Distance       4.
```

Scilab code ARC 5 Hamming Encode

```
//Caption:Hamming Encoding
//H(7,4)
//Code Word Length = 7, Message Word length = 4, Parity bits =3
//clear;
close;
clc;
//Getting Message Word
m3 = input('Enter the 1 bit(MSb) of message word');
m2 = input('Enter the 2 bit of message word');
m1 = input('Enter the 3 bit of message word');
m0 = input('Enter the 4 bit(LSb) of message word');
//Generating Parity bits
for i = 1:(2^4)
  b2(i) = xor(m0(i),xor(m3(i),m1(i)));
  b1(i) = xor(m1(i),xor(m2(i),m3(i)));
  b0(i) = xor(m0(i),xor(m1(i),m2(i)));
  m(i,:) = [m3(i) m2(i) m1(i) m0(i)];
  b(i,:) = [b2(i) b1(i) b0(i)];
end
C = [b m];
disp('-----------------------------------------------------------')
for i = 1:2^4
  disp(i)
  disp(m(i,:),'Message Word')
  disp(b(i,:),'Parity Bits')
  disp(C(i,:),'CodeWord')
  disp("   ");
  disp("   ");
end
disp('-----------------------------------------------------------')
//disp(m b C)
//Result
//Enter the 1 bit(MSb) of message word
```

```
[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1];
//Enter the 2 bit of message word [0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1];
//Enter the 3 bit of message word [0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1];
//Enter the 4 bit(LSb) of message word [0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1];
```

Scilab code ARC 5 invmulaw

```
function x =  invmulaw(y,mu)
  //Non-linear Quantization
  //invmulaw: inverse mulaw nonlinear quantization
  //x = output vector
  //y = input vector (using mulaw nonlinear comression)
  x = (((1+mu).^(abs(y))-1)./mu)
endfunction
```

Scilab code ARC 6 PCM Encoding

```
function [c] = PCM_Encoding(x,L,en_code)
  //Encoding: Converting Quantized decimal sample values in to binary
   //x = input sequence
   //L = number of qunatization levels
   //en_code = normalized input sequence
n = log2(L);
c = zeros(length(x),n);
for i = 1:length(x)
  for j = n:-1:0
    if(fix(en_code(i)/(2^j))==1)
      c(i,(n-j)) =1;
      en_code(i) = en_code(i)-2^j;
    end
  end
end
disp(c)
```

Scilab code ARC 7 PCM Transmission

```
//Caption:PCM Transmission (includes functions:
    uniform_pcm.sce, PCM_encoding.sce)
//This program is a sample program for Pulse Code
    Modulation transmission
//step 1: The given analog signal converted into
    quantized sample value
//step 2: Then the quantized sample value converted
    into binary value
clc;
close;
t = 0:0.001:1;
x  = sin(2*%pi*t);
L = 16;
//Step 1
[SQNR,xq,en_code] = uniform_pcm(x,L);
//Step 2
c = PCM_Encoding(x,L,en_code);
a =gca();
a.x_location ="origin";
a.y_location ="origin";
plot2d2(t*2*%pi,x);
plot2d2(t*2*%pi,xq,5);
title('Quantization of Sampled analog signal')
legend(['Analog signal','Quantized Signal'])
```

Scilab code ARC 8 sinc new

```
function [y]=sinc_new(x)
i=find(x==0);
x(i)= 1;      // From LS: don't need this is /0
    warning is off
y = sin(%pi*x)./(%pi*x);
y(i) = 1;
endfunction
```

Scilab code ARC 9 uniform pcm

```
function [SQNR,xq,en_code] = uniform_pcm(x,L)
  //x = input sequence
  //L = number of qunatization levels
 xmax = max(abs(x));
xq = x/xmax;
en_code = xq;
d = 2/L;
q = d*[0:L-1];
q = q-((L-1)/2)*d;
for i = 1:L
    xq(find(((q(i)-d/2)<= xq)&(xq<=(q(i)+d/2))))=...
    q(i).*ones(1,length(find(((q(i)-d/2)<=xq)&(xq<=(
       q(i)+d/2)))));
    en_code(find(xq == q(i)))= (i-1).*ones(1,length(
       find(xq == q(i))));
end
  xq = xq*xmax;
  SQNR = 20*log10(norm(x)/norm(x-xq));
endfunction
```

Scilab code ARC 10 xor

```
function [value] = xor(A,B)
  if(A==B)
    value = 0;
  else
    value = 1;
  end
endfunction
```
