

Scilab Code for  
Digital Communication,  
by Simon Haykin <sup>1</sup>

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

<sup>1</sup>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 downloaded from the website [www.scilab.in](http://www.scilab.in)

# Book Details

**Authors:** Simon Haykins

**Title:** Digital Communication

**Publisher:** Wiley 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	4
1 Introduction	2
2 Fundamental Limit on Performance	5
3 Detection and Estimation	14
4 Sampling Process	27
5 Waveform Coding Techniques	31
6 Baseband Shaping for Data Transmission	39
7 Digital Modulation Techniques	59
8 Error-Control Coding	87
9 Spread-Spectrum Modulation	94

# List of Scilab Codes

CF 1.2	Digital Representation of Analog signal . . . . .	2
Exa 2.1	Entropy of Binary Memoryless source . . . . .	5
Exa 2.2	Second order Extension of Discrete Memoryless Source	5
Exa 2.3	Entropy, Average length, Variance of Huffman Encoding	7
Exa 2.4	Entropy, Average length, Variance of Huffman Encoding	8
Exa 2.5	Binary Symmetric Channel . . . . .	9
Exa 2.6	Channel Capacity of a Binary Symmetric Channel . .	10
Exa 2.7	Significance of the Channel Coding theorem . . . . .	10
Exa 3.1	Orthonormal basis for given set of signals . . . . .	14
Exa 3.2	M ARY Signaling . . . . .	16
Exa 3.3	Matched Filter output for RF pulse . . . . .	19
Exa 3.4	Matched Filter output for Noise-like signal . . . . .	20
Exa 3.6	Linear Predictor of Order one . . . . .	22
CF 3.29	Implementation of LMS Adaptive Filter algorithm . .	24
Exa 4.1	Bound on Aliasing error for Time-shifted sinc pulse . .	27
Exa 4.3	Equalizier to compensate Aperture effect . . . . .	28
Exa 5.1	Average Transmitted Power for PCM . . . . .	31
Exa 5.2	Comparision of M-ary PCM with ideal system (Channel Capacity Theorem) . . . . .	31
Exa 5.3	Signal-to-Quantization Noise Ratio of PCM . . . . .	32
Exa 5.5	Output Signal-to-Noise ratio for Sinusoidal Modulation	34
CF 5.13a	(a) u-Law companding . . . . .	36
CF 5.13b	(b) A-law companding . . . . .	36
Exa 6.1	Bandwidth Requirements of the T1 carrier . . . . .	39
CF 6.1a	(a) Nonreturn-to-zero unipolar format . . . . .	40
CF 6.1b	(b) Nonreturn-to-zero polar format . . . . .	40
CF 6.1c	(c) Nonreturn-to-zero bipolar format . . . . .	42
Exa 6.2	Duobinary Encoding . . . . .	44

Exa 6.3	Generation of bipolar output for duobinary coder . . .	47
CF 6.4	Power Spectra of different binary data formats . . . .	48
CF 6.6b	(b) Ideal solution for zero ISI . . . . .	49
CF 6.7b	(b) Practical solution: Raised Cosine . . . . .	51
CF 6.9	Frequency response of duobinary conversion filter . . .	53
CF 6.15	Frequency response of modified duobinary conversion filter . . . . .	55
Exa 7.1	QPSK Waveform . . . . .	59
CF 7.1	Waveform of Different Digital Modulation techniques .	61
Exa 7.2	MSK waveforms . . . . .	63
CF 7.2	Signal Space diagram for coherent BPSK . . . . .	68
Tab 7.3	Illustration the generation of DPSK signal . . . . .	70
CF 7.4	Signal Space diagram for coherent BFSK . . . . .	72
CF 7.6	Signal space diagram for coherent QPSK waveform . .	74
Tab 7.6	Bandwidth efficiency of M ary PSK signals . . . . .	74
Tab 7.7	Bandwidth efficiency of M ary FSK signals . . . . .	76
CF 7.29	Power Spectra of BPSK and BFSK signals . . . . .	77
CF 7.30	Power Spectra of QPSK and MSK signals . . . . .	78
CF 7.31	Power spectra of M-ary PSK signals . . . . .	80
CF 7.41	Matched Filter output of rectangular pulse . . . . .	82
Exa 8.1	Repetition Codes . . . . .	87
Exa 8.2	Hamming Codes . . . . .	87
Exa 8.3	Hamming Codes Revisited . . . . .	88
Exa 8.4	Encoder for the (7,4) Cyclic Hamming Code . . . . .	89
Exa 8.5	Syndrome calculator for the(7,4) Cyclic Hamming Code	90
Exa 8.6	Reed-Solomon Codes . . . . .	90
Exa 8.7	Convolutional Encoding - Time domain approach . . .	91
Exa 8.8	Convolutional Encoding Transform domain approach	92
Exa 8.11	Fano metric for binary symmetric channel using convo- lutional code . . . . .	93
Exa 9.1	PN sequence generation . . . . .	94
Exa 9.2	Maximum length sequence property . . . . .	95
Exa 9.3	Processing gain, PN sequence length, Jamming margin in dB . . . . .	97
Exa 9.4	Simple Fast Frequency hopping: FH/MFSK . . . .	100
Fig 9.4	Direct Sequence Spread Coherent BPSK . . . . .	100
ARC 1	Alaw . . . . .	103
ARC 2	auto correlation . . . . .	106

ARC 3	Convolutional Coding . . . . .	107
ARC 4	Hamming Distance . . . . .	108
ARC 5	Hamming Encode . . . . .	108
ARC 5	invmulaw . . . . .	110
ARC 6	PCM Encoding . . . . .	110
ARC 7	PCM Transmission . . . . .	111
ARC 8	sinc new . . . . .	111
ARC 9	uniform pcm . . . . .	112
ARC 10	xor . . . . .	112

# List of Figures

1.1	Figure1.2a . . . . .	3
1.2	Figure1.2b . . . . .	4
2.1	Example2.1 . . . . .	6
2.2	Example2.6 . . . . .	11
2.3	Example2.7 . . . . .	13
3.1	Example3.1a . . . . .	16
3.2	Example3.1b . . . . .	17
3.3	Example3.2 . . . . .	19
3.4	Example3.3 . . . . .	21
3.5	Example3.4 . . . . .	23
3.6	Figure3.29 . . . . .	26
4.1	Example4.1 . . . . .	28
4.2	Example4.3 . . . . .	30
5.1	Example5.2 . . . . .	33
5.2	Figure5.13a . . . . .	37
5.3	Figure5.13b . . . . .	38
6.1	Figure6.1a . . . . .	41
6.2	Figure6.1b . . . . .	43
6.3	Figure6.1c . . . . .	45
6.4	Figure6.4 . . . . .	50
6.5	Figure6.6 . . . . .	52
6.6	Figure6.7 . . . . .	54
6.7	Figure6.9 . . . . .	56
6.8	Figure6.15 . . . . .	58



7.1	Example7.1	61
7.2	Figure7.1a	64
7.3	Figure7.1b	65
7.4	Figure7.1c	66
7.5	Example7.2	69
7.6	Figure7.2	70
7.7	Figure 7.4	73
7.8	Figure7.6	75
7.9	Figure7.12	77
7.10	Figure7.29	79
7.11	Figure7.30	81
7.12	Figure7.31	83
7.13	Figure7.41a	85
7.14	Figure7.41b	86
9.1	Example9.2a	98
9.2	Example9.2b	99
9.3	Figure9.6a	103
9.4	Figure9.6b	104
9.5	Figure10.12	105

# Chapter 1

## Introduction

Scilab code CF 1.2 Digital Representation of Analog signal

```
1 //Caption: Digital Representation of Analog signal
2 //Figure 1.2: Analog to Digital Conversion
3 clear;
4 close;
5 clc;
6 t = -1:0.01:1;
7 x = 2*sin((%pi/2)*t);
8 dig_data = [0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,1,0,1]
9 //
10 figure
11 a=gca();
12 a.x_location = "origin";
13 a.y_location = "origin";
14 a.data_bounds = [-2,-3;2,3]
15 plot(t,x)
16 plot2d3('gnn',0.5,sqrt(2),-9)
17 plot2d3('gnn',-0.5,-sqrt(2),-9)
18 plot2d3('gnn',1,2,-9)
19 plot2d3('gnn',-1,-2,-9)
20 xlabel('
    Time')
21 ylabel('
```

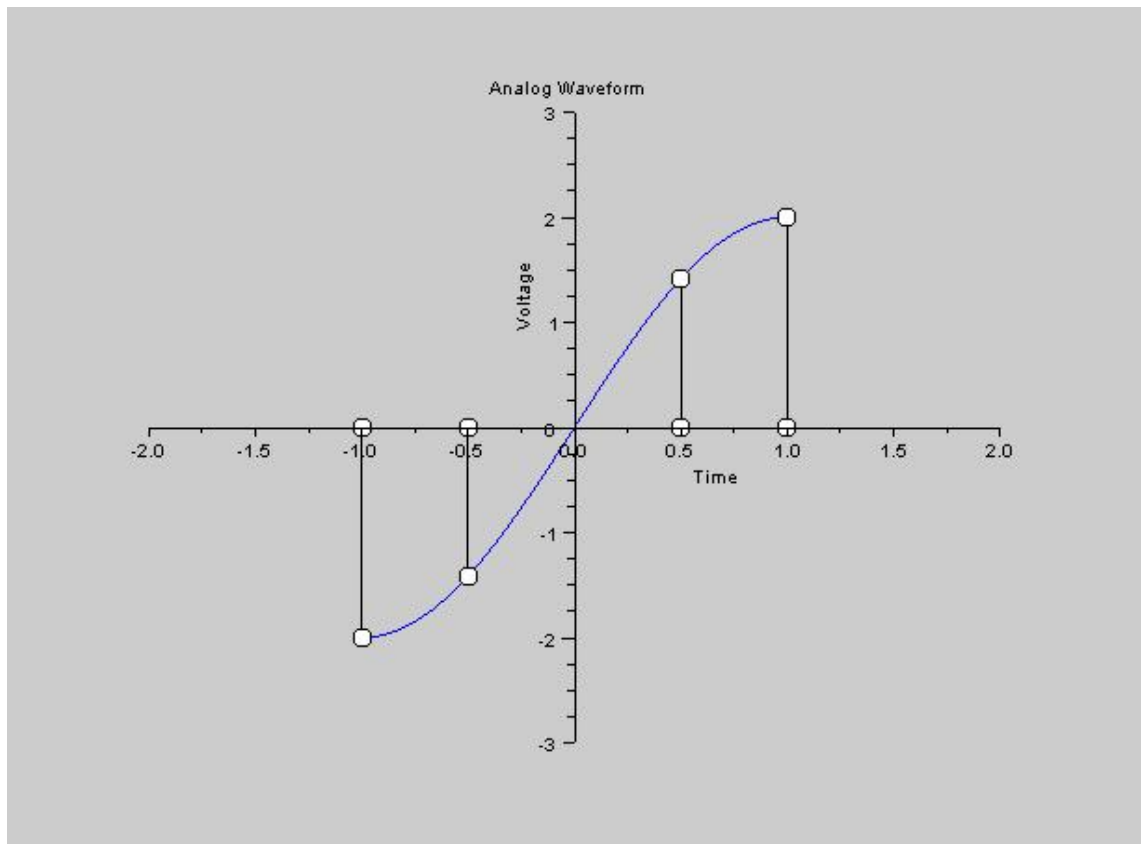


Figure 1.1: Figure1.2a

```

    Voltage')
22 title('Analog Waveform')
23 //
24 figure
25 a = gca();
26 a.data_bounds = [0,0;21,5];
27 plot2d2([1:length(dig_data)],dig_data,5)
28 title('Digital Representation')

```

---

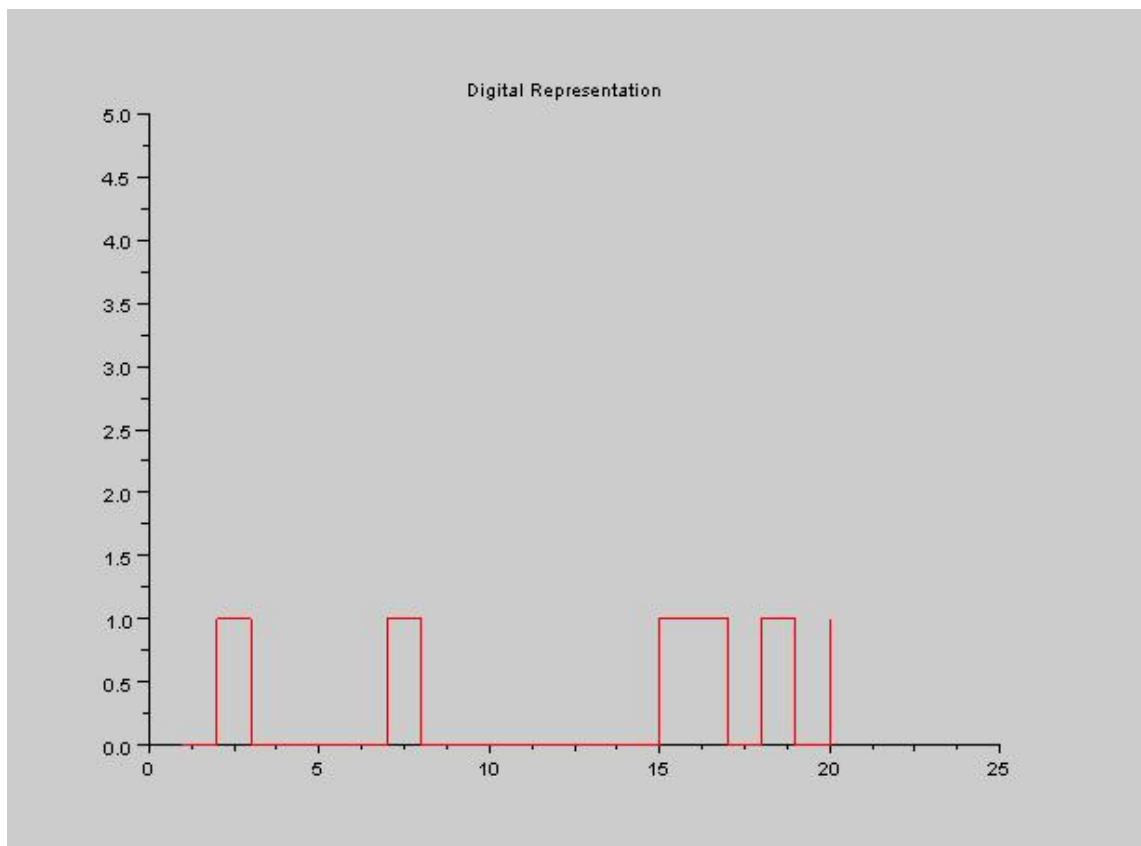


Figure 1.2: Figure1.2b

## Chapter 2

# Fundamental Limit on Performance

Scilab code Exa 2.1 Entropy of Binary Memoryless source

```
1 //Caption:Entropy of Binary Memoryless source
2 //Example 2.1: Entropy of Binary Memoryless Source
3 //page 18
4 clear;
5 close;
6 clc;
7 Po = 0:0.01:1;
8 H_Po = zeros(1,length(Po));
9 for i = 2:length(Po)-1
10     H_Po(i) = -Po(i)*log2(Po(i))-(1-Po(i))*log2(1-Po(i)
11         ));
12 end
13 //plot
14 plot2d(Po,H_Po)
15 xlabel('Symbol Probability , Po')
16 ylabel('H(Po)')
17 title('Entropy function H(Po)')
18 plot2d3('ggn',0.5,1)
```

---

Scilab code Exa 2.2 Second order Extension of Discrete Memoryless Source

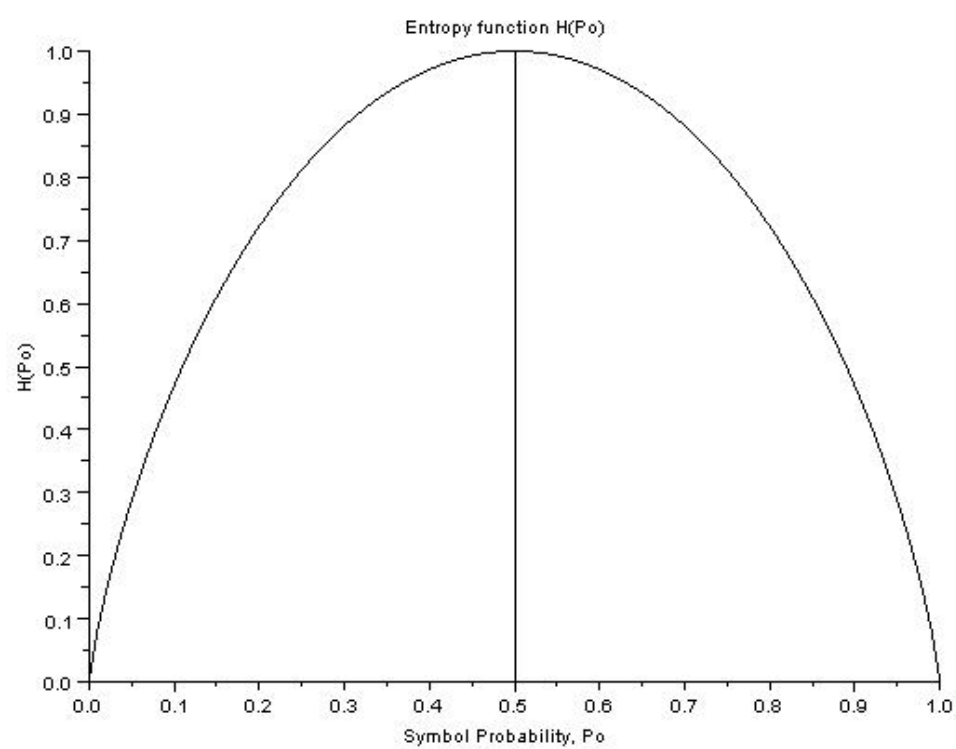


Figure 2.1: Example2.1

```

1 //caption:Second order Extension of Discrete
  Memoryless Source
2 //Example 2.2:Entropy of Discrete Memoryless source
3 //page 19
4 clear;
5 clc;
6 P0 = 1/4; //probability of source alphabet S0
7 P1 = 1/4; //probability of source alphabet S1
8 P2 = 1/2; //probability of source alphabet S2
9 H_Ruo = P0*log2(1/P0)+P1*log2(1/P1)+P2*log2(1/P2);
10 disp('Entropy of Discrete Memoryless Source')
11 disp('bits',H_Ruo)
12 //Second order Extension of discrete Memoryless
  source
13 P_sigma = [P0*P0,P0*P1,P0*P2,P1*P0,P1*P1,P1*P2,P2*P0
  ,P2*P1,P2*P2];
14 disp('Table 2.1 Alphabet Particulars of Second-order
  Extension of a Discrete Memoryless Source')
15 disp('
  -----
  ')
16 disp('Sequence of Symbols of ruo2:')
17 disp('  S0*S0      S0*S1      S0*S2      S1*S0      S1*
  S1      S1*S2      S2*S0      S2*S1      S2*S2')
18 disp(P_sigma,'Probability p(sigma), i =0,1.....8')
19 disp('
  -----
  ')
20 disp(' ')
21 H_Ruo_Square =0;
22 for i = 1:length(P_sigma)
23   H_Ruo_Square = H_Ruo_Square+P_sigma(i)*log2(1/
  P_sigma(i));
24 end
25 disp('bits', H_Ruo_Square,'H(Ruo_Square)=')
26 disp('H(Ruo_Square) = 2*H(Ruo)')

```

---

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

```

1 //Caption:Entropy , Average length , Variance of
  Huffman Encoding
2 //Example 2.3: Huffman Encoding: Calculation of
3 // (a)Average code-word length 'L'
4 //(b)Entropy 'H'
5 clear;
6 clc;
7 P0 = 0.4; //probability of codeword '00'
8 L0 = 2;   //length of codeword S0
9 P1 = 0.2; //probability of codeword '10'
10 L1 = 2;  //length of codeword S1
11 P2 = 0.2; //probability of codeword '11'
12 L2 = 2;  //length of codeword S2
13 P3 = 0.1; //probability of codeword '010'
14 L3 = 3;  //length of codeword S3
15 P4 = 0.1; //probability of codeword '011'
16 L4 = 3;  //length of codeword S4
17 L = P0*L0+P1*L1+P2*L2+P3*L3+P4*L4;
18 H_Ruo = P0*log2(1/P0)+P1*log2(1/P1)+P2*log2(1/P2)+P3
    *log2(1/P3)+P4*log2(1/P4);
19 disp('bits',L,'Average code-word Length L')
20 disp('bits',H_Ruo,'Entropy of Huffman coding result
    H')
21 disp('percent',((L-H_Ruo)/H_Ruo)*100,'Average code-
    word length L exceeds the entropy H(Ruo) by only'
    )
22 sigma_1 = P0*(L0-L)^2+P1*(L1-L)^2+P2*(L2-L)^2+P3*(L3
    -L)^2+P4*(L4-L)^2;
23 disp(sigma_1,'Varinace of Huffman code')

```

---

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

```

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

```



```

2 //Example2.4: Illustrating nonuniquess of the
   Huffman Encoding
3 // Calculation of (a)Average code-word length 'L' (b
   )Entropy 'H'
4 clear;
5 clc;
6 P0 = 0.4; //probability of codeword '1'
7 L0 = 1;   //length of codeword S0
8 P1 = 0.2; //probability of codeword '01'
9 L1 = 2;   //length of codeword S1
10 P2 = 0.2; //probability of codeword '000'
11 L2 = 3;   //length of codeword S2
12 P3 = 0.1; //probability of codeword '0010'
13 L3 = 4;   //length of codeword S3
14 P4 = 0.1; //probability of codeword '0011'
15 L4 = 4;   //length of codeword S4
16 L = P0*L0+P1*L1+P2*L2+P3*L3+P4*L4;
17 H_Ruo = P0*log2(1/P0)+P1*log2(1/P1)+P2*log2(1/P2)+P3
   *log2(1/P3)+P4*log2(1/P4);
18 disp('bits',L,'Average code-word Length L')
19 disp('bits',H_Ruo,'Entropy of Huffman coding result
   H')
20 sigma_2 = P0*(L0-L)^2+P1*(L1-L)^2+P2*(L2-L)^2+P3*(L3
   -L)^2+P4*(L4-L)^2;
21 disp(sigma_2,'Varinace of Huffman code')

```

---

#### Scilab code Exa 2.5 Binary Symmetric Channel

```

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

```

```

9 disp('Transition probability')
10 disp(pe,'probability of 0 receiving if a 1 is sent =
    probability of 1 receiving if a 0 is sent=')

```

---

**Scilab code Exa 2.6** Channel Capacity of a Binary Symmetric Channel

```

1 //Caption:Channel Capacity of a Binary Symmetric
  Channel
2 //Example2.6:Channel Capacity of Binary Symmetri
  Channel
3 clear;
4 close;
5 clc;
6 p = 0:0.01:0.5;
7 for i =1:length(p)
8     if(i~=1)
9         C(i) = 1+p(i)*log2(p(i))+(1-p(i))*log2((1-p(i)))
10        ;
11    elseif(i==1)
12        C(i) =1;
13    elseif(i==length(p))
14        C(i)=0;
15    end
16 end
17 plot2d(p,C,5)
18 xlabel('Transition Probability , p')
19 ylabel('Channel Capacity , C')
20 title('Figure 2.10 Variation of channel capacity of
    a binary symmetric channel with transition
    probability p')

```

---

**Scilab code Exa 2.7** Significance of the Channel Coding theorem

```

1 //Caption:Significance of the Channel Coding theorem
2 //Example2.7: Significance of the channel coding
  theorem
3 //Average Probability of Error of Repetition Code

```

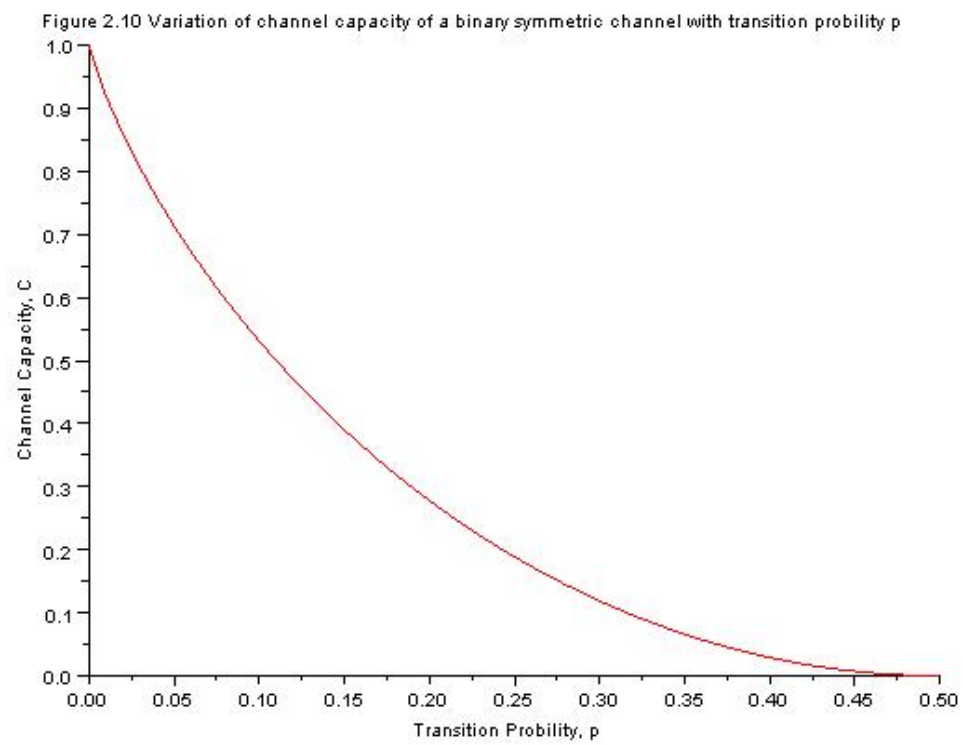


Figure 2.2: Example2.6

```

4  clear;
5  clc;
6  close;
7  p =10^-2;
8  pe_1 =p; //Average Probility of error for code rate
    r = 1
9  pe_3 = 3*p^2*(1-p)+p^3;//probility of error for code
    rate r =1/3
10 pe_5 = 10*p^3*(1-p)^2+5*p^4*(1-p)+p^5;//error for
    code rate r =1/5
11 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
12 r = [1,1/3,1/5,1/7];
13 pe = [pe_1,pe_3,pe_5,pe_7];
14 a=gca();
15 a.data_bounds=[0,0;1,0.01];
16 plot2d(r,pe,5)
17 xlabel('Code rate , r')
18 ylabel('Average Probability of error , Pe')
19 title('Figure 2.12 Illustrating significance of the
    channel coding theorem')
20 legend('Repetition codes')
21 xgrid(1)
22 disp('Table 2.3 Average Probility of Error for
    Repetition Code')
23 disp('
    -----
    ')
24 disp(r,'Code Rate , r =1/n',pe,'Average Probility of
    Error , Pe')
25 disp('
    -----
    ')

```

---

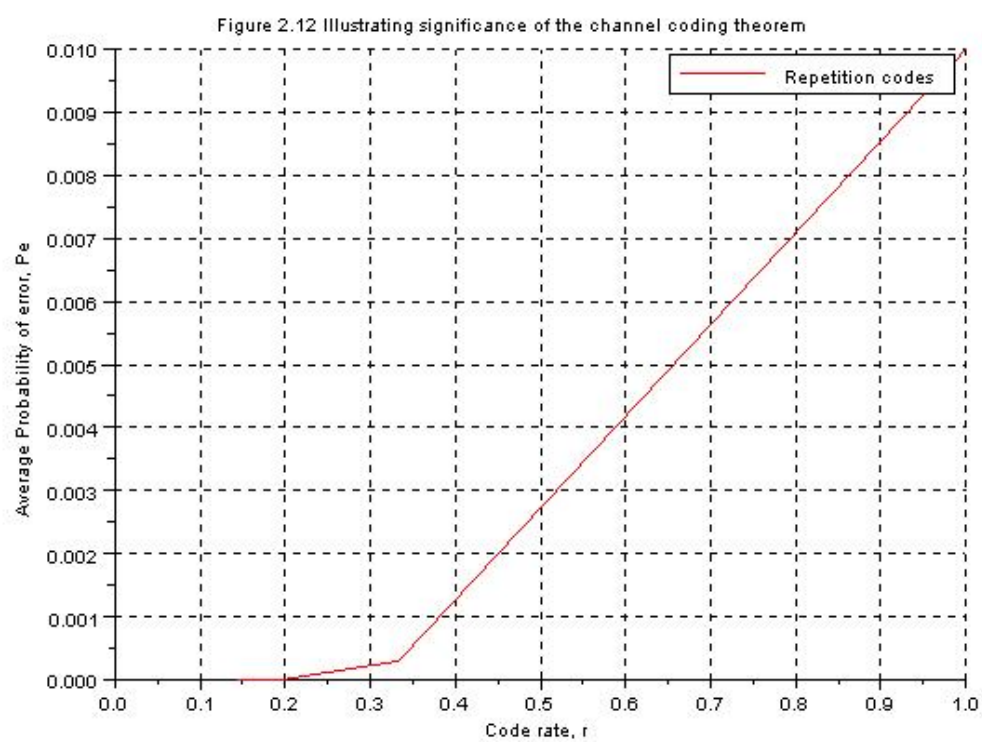


Figure 2.3: Example2.7

## Chapter 3

# Detection and Estimation

**Scilab code Exa 3.1** Orthonormal basis for given set of signals

```
1 //Caption:Orthonormal basis for given set of signals
2 //Example3.1:Finding orthonormal basis for the given
  signals
3 //using Gram-Schmidt orthogonalization procedure
4 clear;
5 close;
6 clc;
7 T = 1;
8 t1 = 0:0.01:T/3;
9 t2 = 0:0.01:2*T/3;
10 t3 = T/3:0.01:T;
11 t4 = 0:0.01:T;
12 s1t = [0,ones(1,length(t1)-2),0];
13 s2t = [0,ones(1,length(t2)-2),0];
14 s3t = [0,ones(1,length(t3)-2),0];
15 s4t = [0,ones(1,length(t4)-2),0];
16 t5 = 0:0.01:T/3;
17 phi1t = sqrt(3/T)*[0,ones(1,length(t5)-2),0];
18 t6 =T/3:0.01:2*T/3;
19 phi2t = sqrt(3/T)*[0,ones(1,length(t6)-2),0];
20 t7 = 2*T/3:0.01:T;
21 phi3t = sqrt(3/T)*[0,ones(1,length(t7)-2),0];
22 //
```

```

23 figure
24 title('Figure3.4(a) Set of signals to be
    orthonormalized')
25 subplot(4,1,1)
26 a = gca();
27 a.data_bounds = [0,0;2,2];
28 plot2d2(t1,s1t,5)
29 xlabel('t')
30 ylabel('s1(t)')
31 subplot(4,1,2)
32 a = gca();
33 a.data_bounds = [0,0;2,2];
34 plot2d2(t2,s2t,5)
35 xlabel('t')
36 ylabel('s2(t)')
37 subplot(4,1,3)
38 a = gca();
39 a.data_bounds = [0,0;2,2];
40 plot2d2(t3,s3t,5)
41 xlabel('t')
42 ylabel('s3(t)')
43 subplot(4,1,4)
44 a = gca();
45 a.data_bounds = [0,0;2,2];
46 plot2d2(t4,s4t,5)
47 xlabel('t')
48 ylabel('s4(t)')
49 //
50 figure
51 title('Figure3.4(b) The resulting set of orthonormal
    functions')
52 subplot(3,1,1)
53 a = gca();
54 a.data_bounds = [0,0;2,4];
55 plot2d2(t5,phi1t,5)
56 xlabel('t')
57 ylabel('phi1(t)')
58 subplot(3,1,2)

```

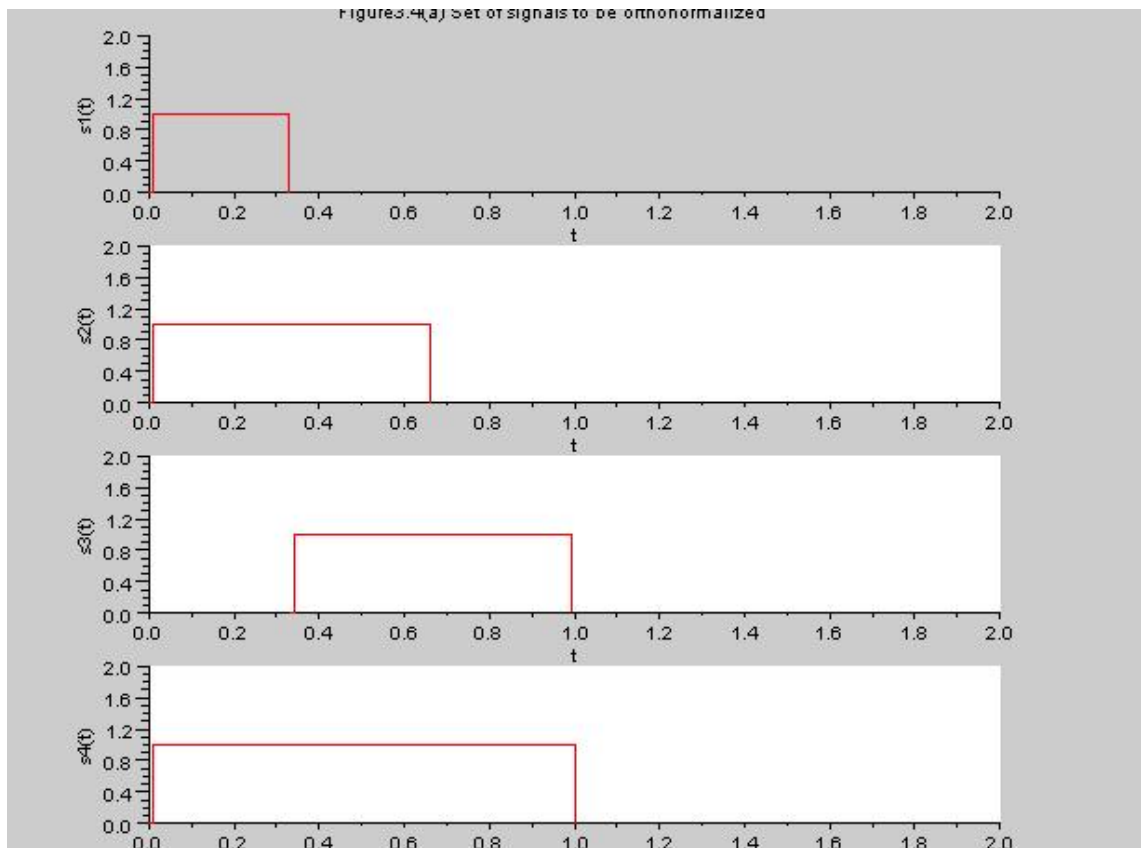


Figure 3.1: Example3.1a

```

59 a = gca();
60 a.data_bounds = [0,0;2,4];
61 plot2d2(t6,phi2t,5)
62 xlabel('t')
63 ylabel('phi2(t)')
64 subplot(3,1,3)
65 a = gca();
66 a.data_bounds = [0,0;2,4];
67 plot2d2(t7,phi3t,5)
68 xlabel('t')
69 ylabel('phi3(t)')

```

---



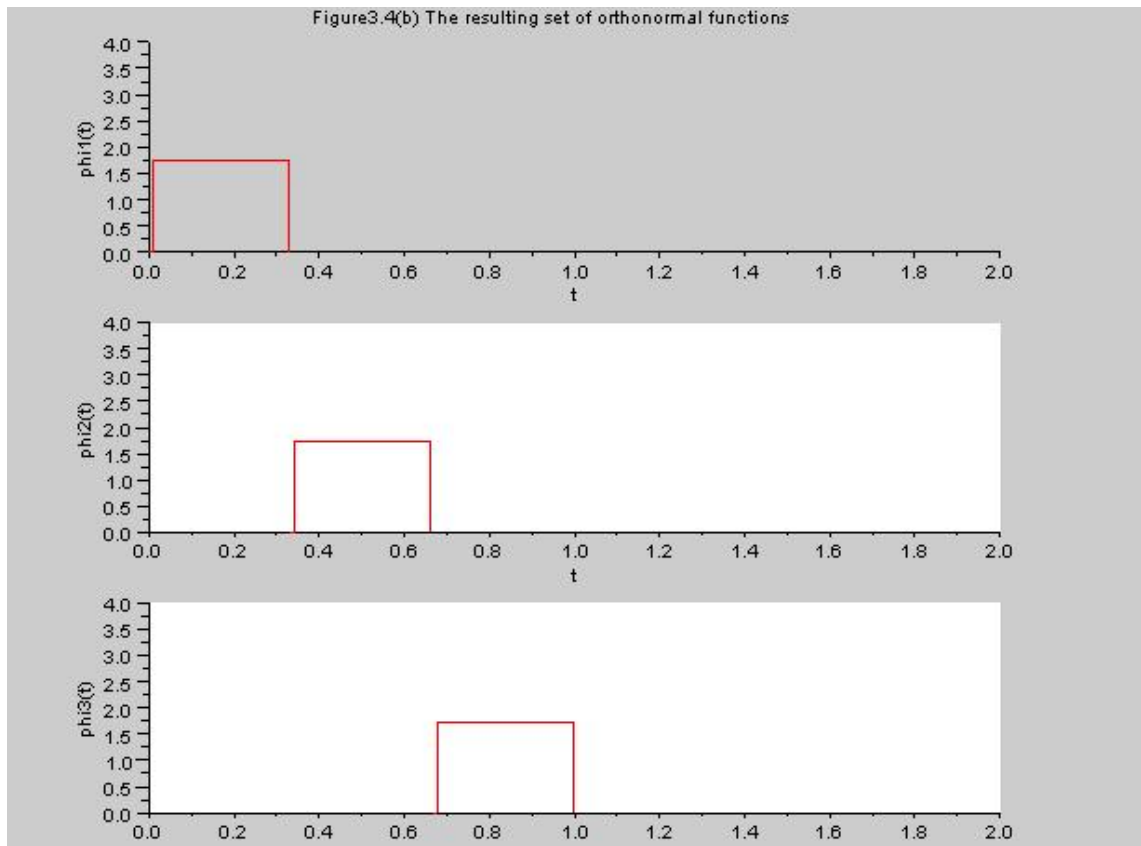


Figure 3.2: Example3.1b

### Scilab code Exa 3.2 M-ARY Signaling

```

1 //Caption:M-ARY Signaling
2 //Example3.2:M-ARY SIGNALING
3 //Signal constellation and Representation of dibits
4 clear;
5 close;
6 clc;
7 a =1; //amplitude =1
8 T =1; //Symbol duration in seconds
9 //Four message points
10 Si1 = [(-3/2)*a*sqrt(T), (-1/2)*a*sqrt(T), (3/2)*a*
        sqrt(T), (1/2)*a*sqrt(T)];
11 a =gca();
12 a.data_bounds = [-2,-0.5;2,0.5]
13 plot2d(Si1,[0,0,0,0],-10)
14 xlabel('phi1(t)')
15 title('Figure 3.8 (a) Signal constellation')
16 xgrid(1)
17 disp('Figure 3.8 (b). Representation of transmitted
        dibits')
18 disp('Loc. of meg. point | (-3/2)asqrt(T) | (-1/2)asqrt(
        T) | (3/2)asqrt(T) | (1/2)asqrt(T)')
19 disp('
        -----
        ')
20 disp('Transmitted dibit |          00          |          01
        |          11          |          10')
21 disp('')
22 disp('')
23 disp('Figure 3.8 (c). Decision intervals for
        received dibits')
24 disp('Received dibit          |          00          |          01
        |          11          |          10')
25 disp('
        -----
        ')
26 disp('Interval on phi1(t) | x1 < -a.sqrt(T) | -a.sqrt(

```

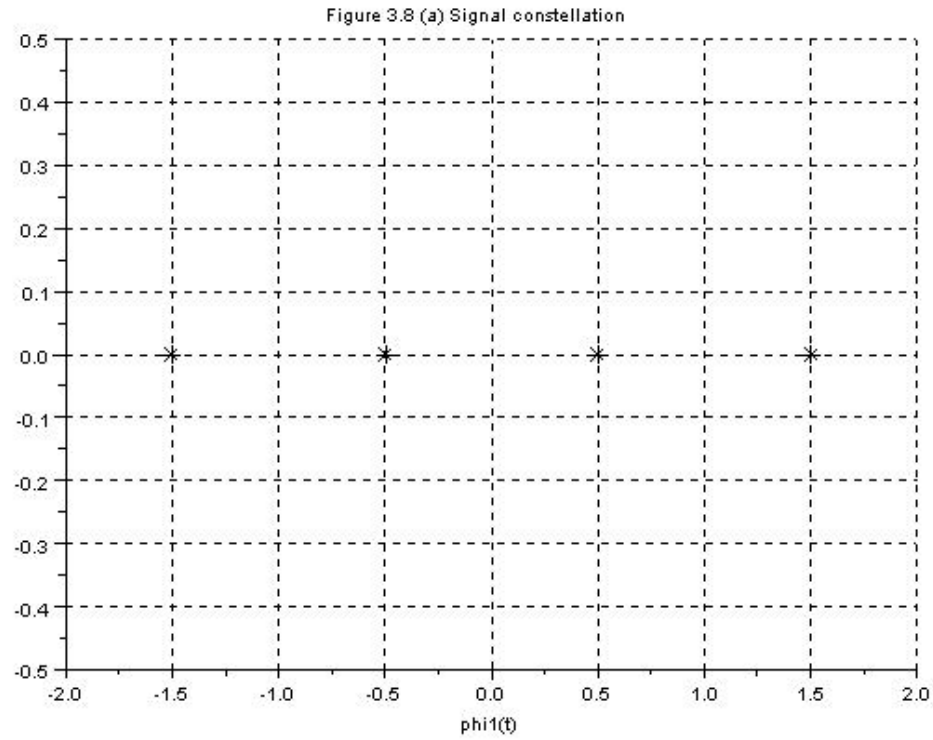


Figure 3.3: Example3.2

$$T) < x_1 < 0 \mid 0 < x_1 < a \cdot \sqrt{T} \mid a \cdot \sqrt{T} < x_1')$$


---

**Scilab code Exa 3.3** Matched Filter output for RF pulse

```

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

```

9  phit = sqrt(2/T)*cos(2*%pi*fc*t1);
10 hopt = phit;
11 phiot = convol(phit,hopt);
12 phiot = phiot/max(phiot);
13 t2 = 0:0.01:2*T;
14 subplot(2,1,1)
15 a =gca();
16 a.x_location = "origin";
17 a.y_location = "origin";
18 a.data_bounds = [0,-1;1,1];
19 plot2d(t1,phit);
20 xlabel('

        t ')
21 ylabel('

        phi(t)')
22 title('Figure 3.13 (a) RF pulse input')
23 subplot(2,1,2)
24 a =gca();
25 a.x_location = "origin";
26 a.y_location = "origin";
27 a.data_bounds = [0,-1;1,1];
28 plot2d(t2,phiot);
29 xlabel('

        t ')
30 ylabel('

        phi0(t)')
31 title('Figure 3.13 (b) Matched Filter output')

```

---

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

```

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

```

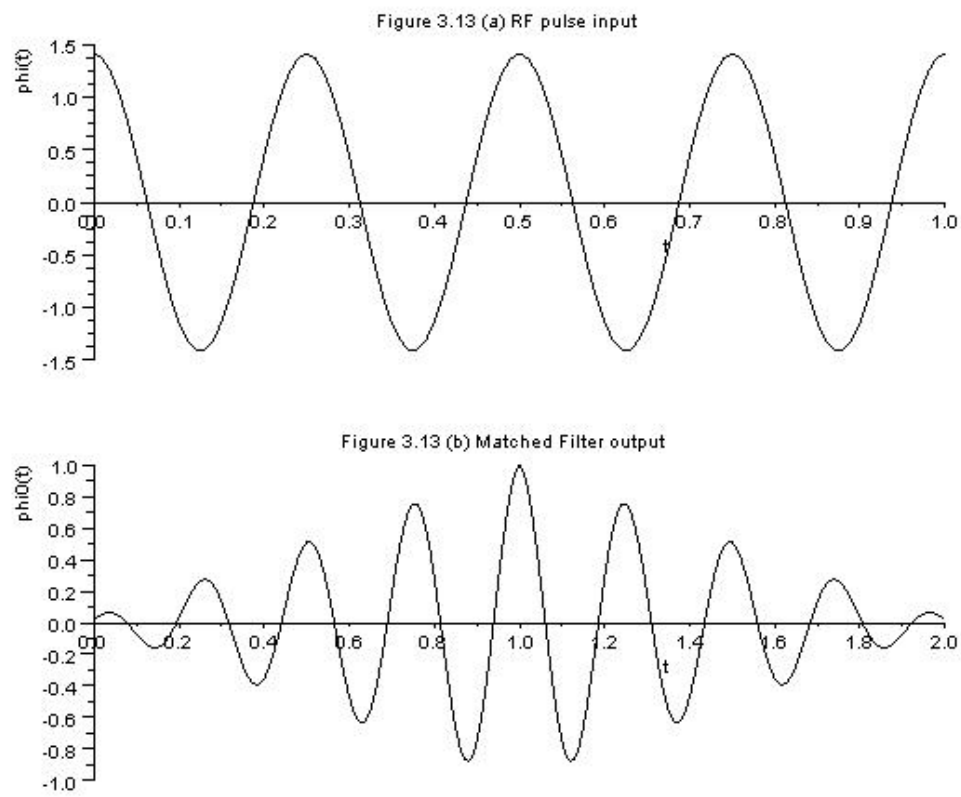


Figure 3.4: Example3.3

```

    input
3  clear;
4  close;
5  clc;
6  phit = 0.1*rand(1,10,'uniform');
7  hopt = phit;
8  phi0t = convol(phit,hopt);
9  phi0t = phi0t/max(phi0t);
10 subplot(2,1,1)
11 a =gca();
12 a.x_location = "origin";
13 a.y_location = "origin";
14 a.data_bounds = [0,-1;1,1];
15 plot2d([1:length(phit)],phit);
16 xlabel('

    t ')
17 ylabel('

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

    t ')
26 ylabel('

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

```

---

**Scilab code Exa 3.6** Linear Predictor of Order one

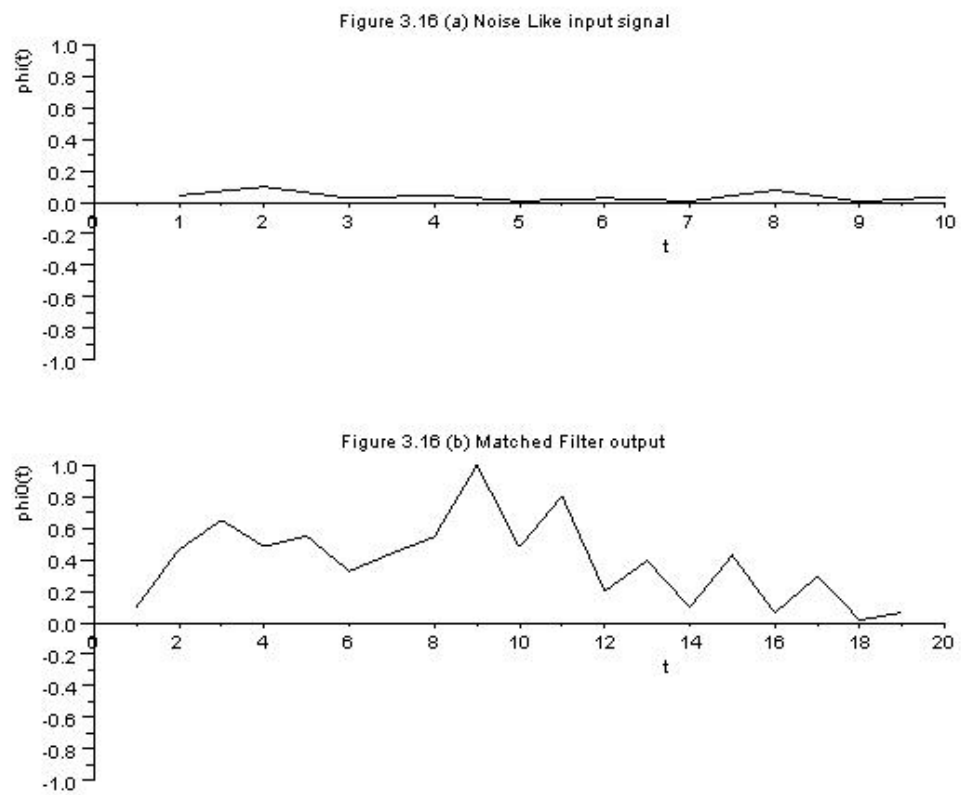


Figure 3.5: Example3.4

```

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

```

---

**Scilab code CF 3.29** Implementation of LMS Adaptive Filter algorithm

```

1 //Implementation of LMS ADAPTIVE FILTER
2 //For noise cancellation application
3 clear;
4 clc;
5 close;
6 order = 18;
7 t =0:0.01:1;
8 x = sin(2*%pi*5*t);
9 noise =rand(1,length(x));
10 x_n = x+noise;
11 ref_noise = noise*rand(10);
12 w = zeros(order,1);
13 mu = 0.01*(sum(x.^2)/length(x));
14 N = length(x);
15 for k =1:1010
16     for i = 1:N-order-1
17         buffer = ref_noise(i:i+order-1);
18         desired(i) = x_n(i)-buffer*w;

```



```
19     w = w+(buffer*mu*desired(i))';
20     end
21 end
22 subplot(4,1,1)
23 plot2d(t,x)
24 title('Original Input Signal')
25 subplot(4,1,2)
26 plot2d(t,noise,2)
27 title('random noise')
28 subplot(4,1,3)
29 plot2d(t,x_n,5)
30 title('Signal+noise')
31 subplot(4,1,4)
32 plot(desired)
33 title('noise removed signal')
```

---

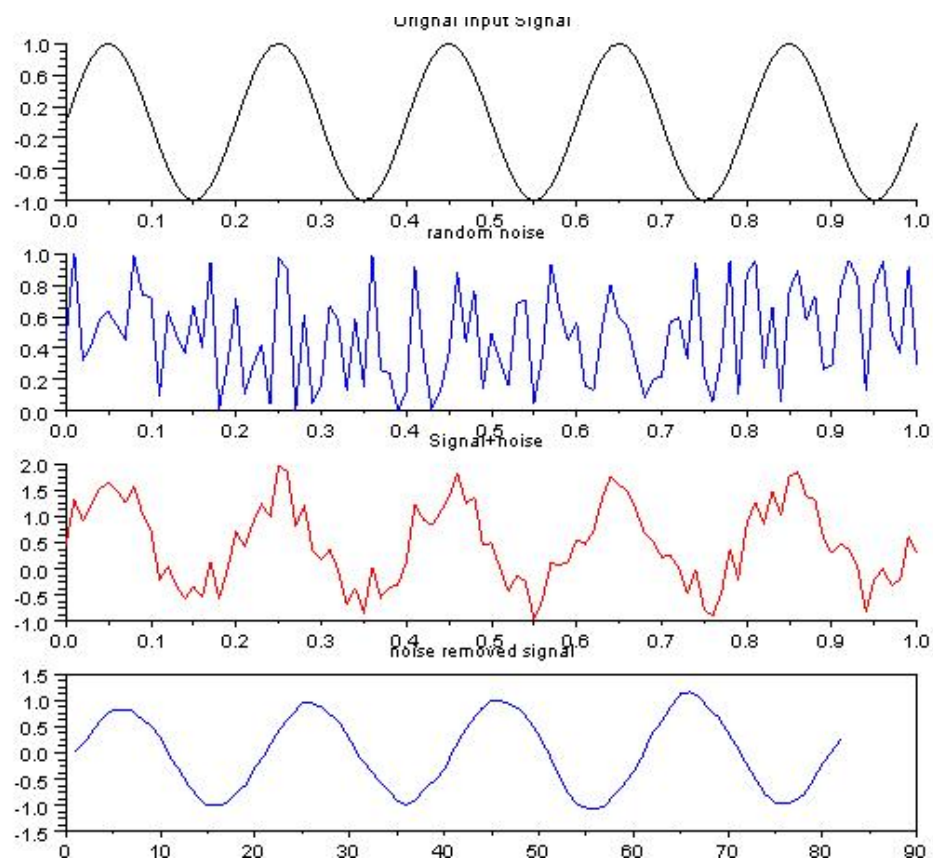


Figure 3.6: Figure3.29

# Chapter 4

## Sampling Process

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

```
1 //Caption:Bound on Aliasing error for Time-shifted
   sinc pulse
2 //Example4.1:Maximum bound on aliasing error for
   sinc pulse
3 clc;
4 close;
5 t = -1.5:0.01:2.5;
6 g = 2*sinc_new(2*t-1);
7 disp(max(g),'Aliasing error cannot exceed max|g(t)|'
   )
8 f = -1:0.01:1;
9 G = [0,0,0,0,ones(1,length(f)),0,0,0,0];
10 f1 = -1.04:0.01:1.04;
11 subplot(2,1,1)
12 a=gca();
13 a.data_bounds =[-3,-1;2,2];
14 a.x_location = "origin"
15 a.y_location = "origin"
16 plot2d(t,g)
17 xlabel('                                t')
18 ylabel('                                g(t)')
19 title('Figure 4.8 (a) Sinc pulse g(t)')
20 subplot(2,1,2)
```

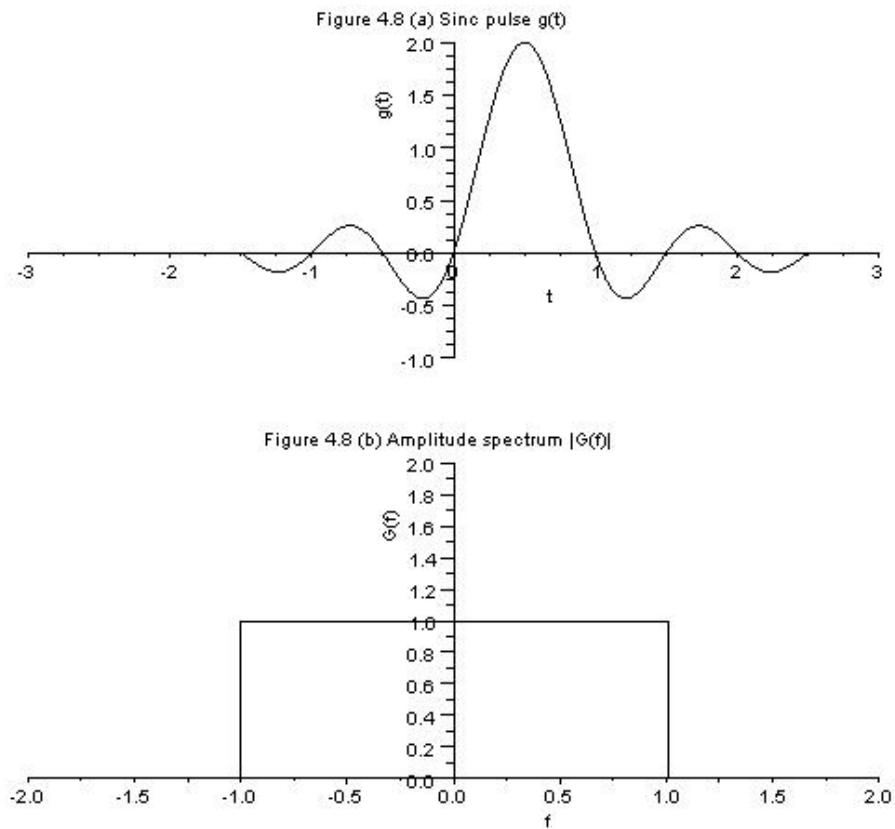


Figure 4.1: Example4.1

```

21 a=gca();
22 a.data_bounds =[-2,0;2,2];
23 a.x_location = "origin"
24 a.y_location = "origin"
25 plot2d2(f1,G)
26 xlabel('f')
27 ylabel('G(f)')
28 title('Figure 4.8 (b) Amplitude spectrum |G(f)|')

```

---

**Scilab code Exa 4.3** Equalizer to compensate Aperture effect

```

1 //Caption:Equalizer to compensate Aperture effect

```

```

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

```

---

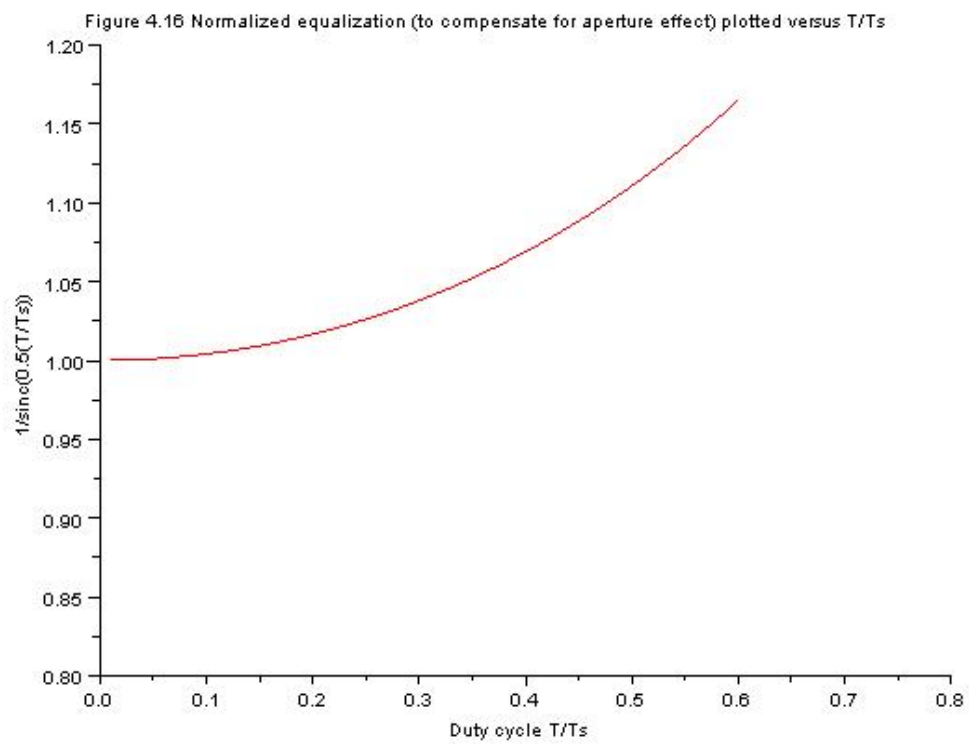


Figure 4.2: Example4.3

## Chapter 5

# Waveform Coding Techniques

**Scilab code Exa 5.1** Average Transmitted Power for PCM

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

---

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

```

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

```

---

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

```

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

```



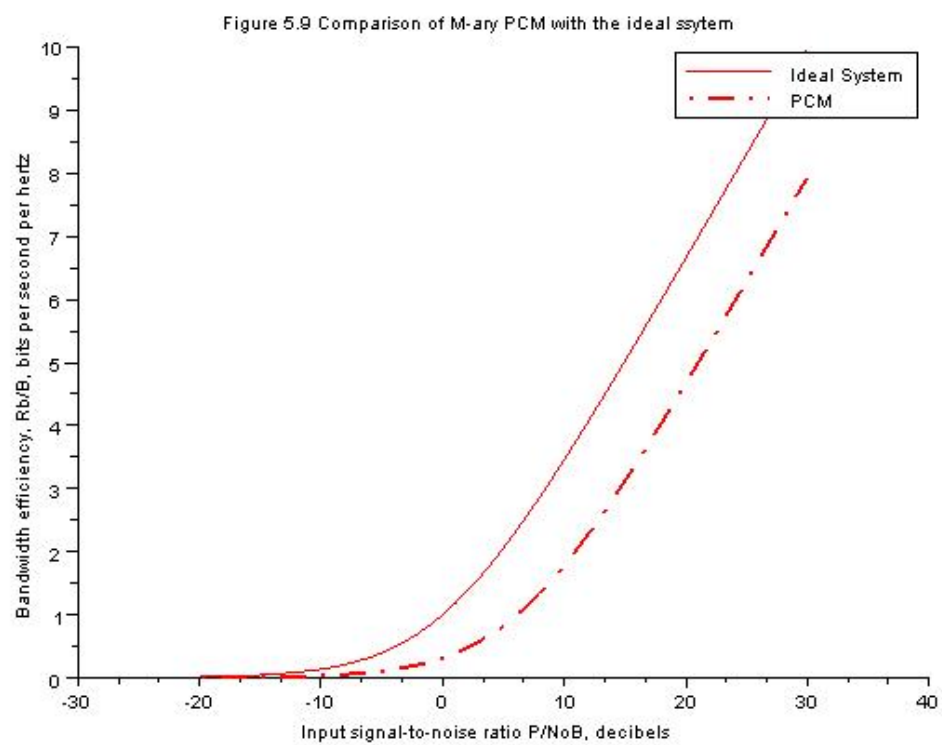


Figure 5.1: Example5.2

```

5  clc;
6  n = input('Enter no. of bits used to encode:')
7  W = input('Enter the message signal bandwidth in Hz: '
            )
8  B = n*W;
9  disp(B, 'Channel width in Hz:')
10 SNRo = 6*n - 7.2;
11 disp(SNRo, 'Output Signal to noise ratio in dB:')
12 //Result 1 if n = 8 bits
13 //Enter no. of bits used to encode: 8
14 //Enter the message signal bandwidth in Hz: 4000
15 //Channel width in Hz: 32000.
16 //Output Signal to noise ratio in dB: 40.8
17 ////////////////////////////////////////////
18 //Result 2 if n = 9 bits
19 //Enter no. of bits used to encode:9
20 //Enter the message signal bandwidth in Hz:4000
21 //Channel width in Hz: 36000.
22 //Output Signal to noise ratio in dB: 46.8
23 ////////////////////////////////////////////
24 //Conclusion: comparing result 1 with result 2 if
    number of bits increased by 1
25 //corresponding output signal to noise in PCM
    increased by 6 dB.

```

---

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

```

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

```

```

7  f0 = input('Enter the frequency of sinusoidal signal
            in Hz:');
8  fs = input('Enter the sampling frequency in samples
            per seconds:');
9  Ts = 1/fs; //Sampling interval
10 delta = 2*pi*f0*a0*Ts; //Step size to avoid slope
            overload
11 Pmax = (a0^2)/2; //maximum permissible output power
12 sigma_Q = (delta^2)/3; //Quantization error or noise
            variance
13 W = f0; //Maximum message bandwidth
14 N = W*Ts*sigma_Q; //Average output noise power
15 SNRo = Pmax/N; // Maximum output signal-to-noise
            ratio
16 SNRo_dB = 10*log10(SNRo);
17 disp(SNRo_dB, 'Maximum output signal-to-nosie in dB
            for Delta Modualtion:');
18 //Result 1 for fs = 8000 Hertz
19 //Enter the amplitude of sinusoidal signal:1
20 //Enter the frequency of sinusoidal signal in Hz
            :4000
21 //Enter the sampling frequency in samples per
            seconds:8000
22 //Maximum output signal-to-nosie in dB for Delta
            Modualtion:-5.1717849
23 //
            //////////////////////////////////////
24 //Result 2 for fs = 16000 Hertz
25 //Enter the amplitude of sinusoidal signal:1
26 //Enter the frequency of sinusoidal signal in Hz
            :4000
27 //Enter the sampling frequency in samples per
            seconds:16000
28 //Maximum output signal-to-nosie in dB for Delta
            Modualtion:3.859115
29 //
            //////////////////////////////////////

```

```

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

```

---

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

```

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

```

---

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

```

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

```

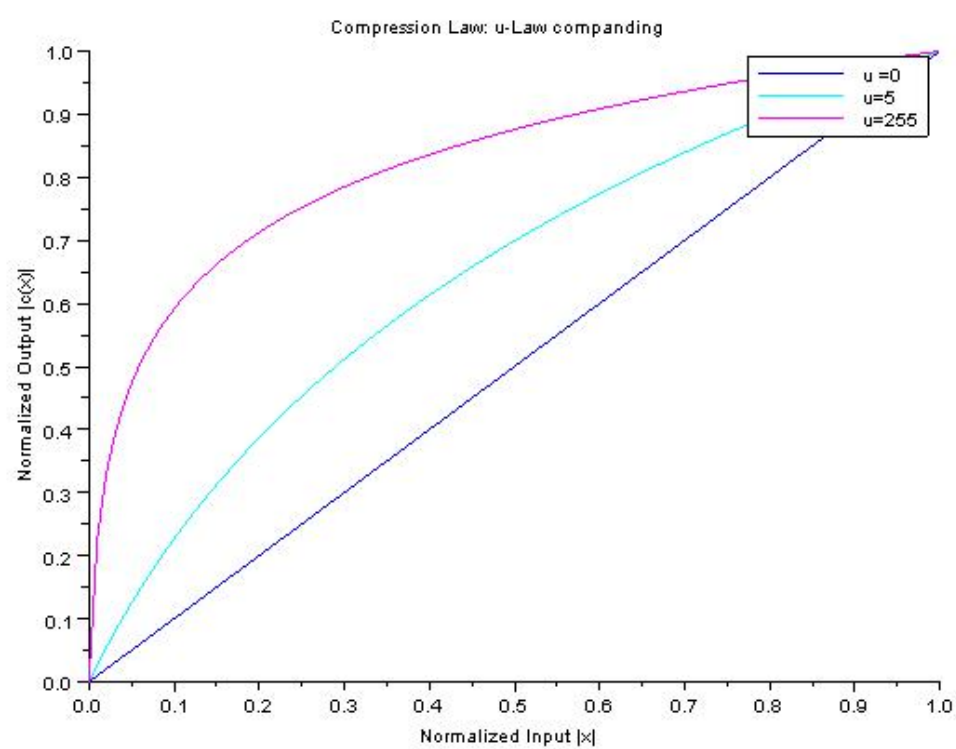


Figure 5.2: Figure5.13a

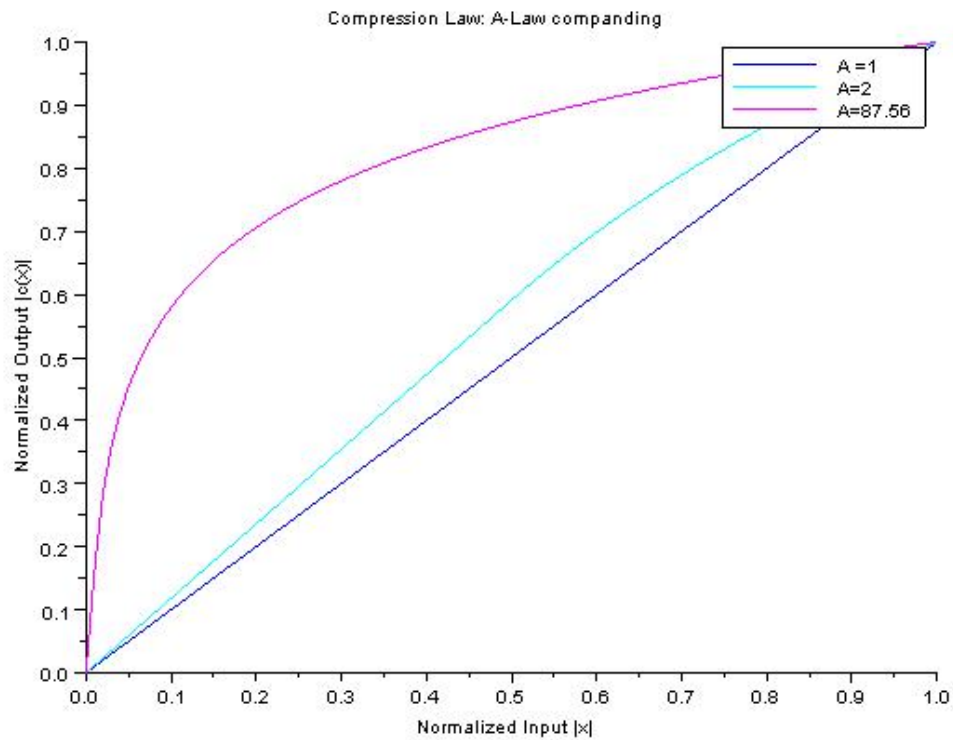


Figure 5.3: Figure5.13b

```

9    [Cx(i,:),Xmax(i)] = Alaw(x,A(i));
10 end
11 plot2d(x/Xmax(1),Cx(1,:),2)
12 plot2d(x/Xmax(2),Cx(2,:),4)
13 plot2d(x/Xmax(3),Cx(3,:),6)
14 xtitle('Compression Law: A-Law companding','
        Normalized Input |x|','Normalized Output |c(x)|')
    ;
15 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

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

in Hz:1545595.1

---

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

```
1 //Caption:Nonreturn-to-zero unipolar format
2 //Figure 6.1(a): Discrete PAM Signals Generation
3 //[1]. Unipolar NRZ
4 //page 235
5 clear;
6 close;
7 clc;
8 x = [0 1 0 0 0 1 0 0 1 1];
9 binary_zero = [0 0 0 0 0 0 0 0 0 0];
10 binary_one = [1 1 1 1 1 1 1 1 1 1];
11 L = length(x);
12 L1 = length(binary_zero);
13 total_duration = L*L;
14 //plotting
15 a = gca();
16 a.data_bounds = [0 -2;L*L1 2];
17 for i =1:L
18     if(x(i)==0)
19         plot([i*L-L+1:i*L],binary_zero);
20         poly1= a.children(1).children(1);
21         poly1.thickness =3;
22     else
23         plot([i*L-L+1:i*L],binary_one);
24         poly1= a.children(1).children(1);
25         poly1.thickness =3;
26     end
27 end
28 xgrid(1)
29 title('Unipolar NRZ')
```

---

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

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



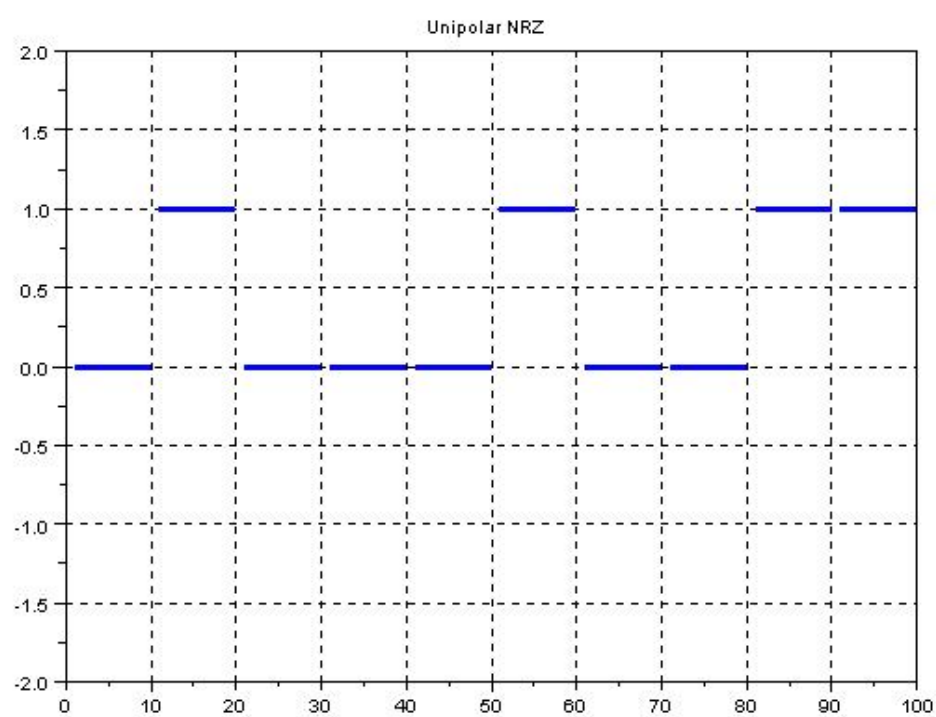


Figure 6.1: Figure6.1a

```

2 //Figure 6.1(b): Discrete PAM Signals Generation
3 // [2].Polar NRZ
4 //page 235
5 clear;
6 close;
7 clc;
8 x = [0 1 0 0 0 1 0 0 1 1];
9 binary_negative = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1];
10 binary_positive = [1 1 1 1 1 1 1 1 1 1];
11 L = length(x);
12 L1 = length(binary_negative);
13 total_duration = L*L1;
14 //plotting
15 a =gca();
16 a.data_bounds =[0 -2;L*L1 2];
17 for i =1:L
18     if(x(i)==0)
19         plot([i*L-L+1:i*L],binary_negative);
20         poly1= a.children(1).children(1);
21         poly1.thickness =3;
22     else
23         plot([i*L-L+1:i*L],binary_positive);
24         poly1= a.children(1).children(1);
25         poly1.thickness =3;
26     end
27 end
28 xgrid(1)
29 title('Polar NRZ')

```

---

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

```

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

```

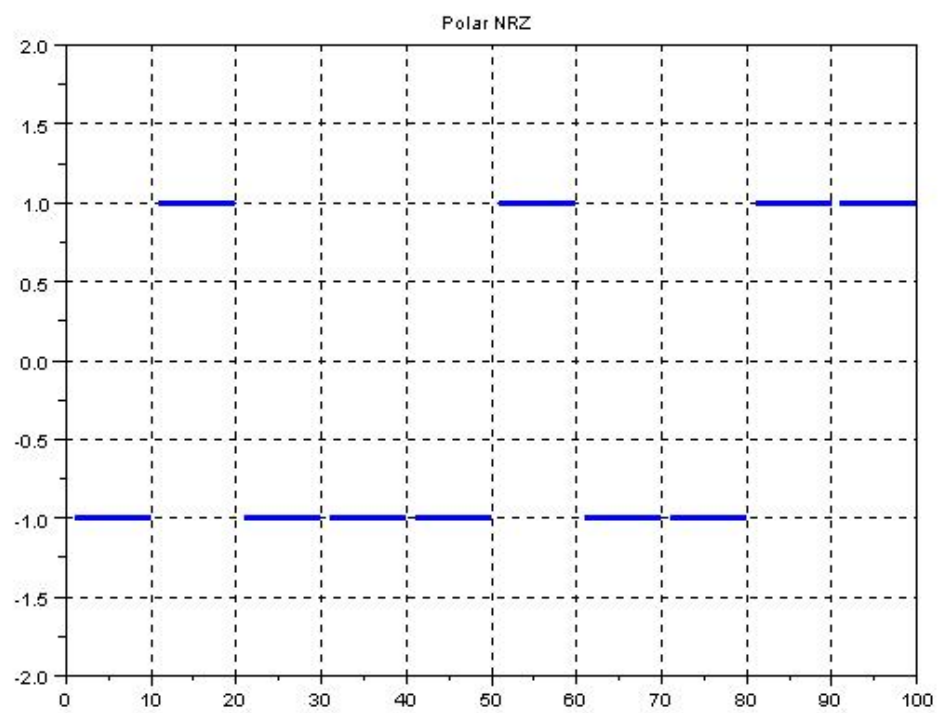


Figure 6.2: Figure6.1b

```

7  clc;
8  x = [0 1 1 0 0 1 0 0 1 1];
9  binary_negative = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1];
10 binary_zero = [0 0 0 0 0 0 0 0 0 0];
11 binary_positive = [1 1 1 1 1 1 1 1 1 1];
12 L = length(x);
13 L1 = length(binary_negative);
14 total_duration = L*L1;
15 //plotting
16 a = gca();
17 a.data_bounds = [0 -2;L*L1 2];
18 for i = 1:L
19     if(x(i)==0)
20         plot([i*L-L+1:i*L],binary_zero);
21         poly1= a.children(1).children(1);
22         poly1.thickness =3;
23     elseif((x(i)==1)&(x(i-1)~=1))
24         plot([i*L-L+1:i*L],binary_positive);
25         poly1= a.children(1).children(1);
26         poly1.thickness =3;
27     else
28         plot([i*L-L+1:i*L],binary_negative);
29         poly1= a.children(1).children(1);
30         poly1.thickness =3;
31     end
32 end
33 xgrid(1)
34 title('BiPolar NRZ')

```

---

#### Scilab code Exa 6.2 Duobinary Encoding

```

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

```

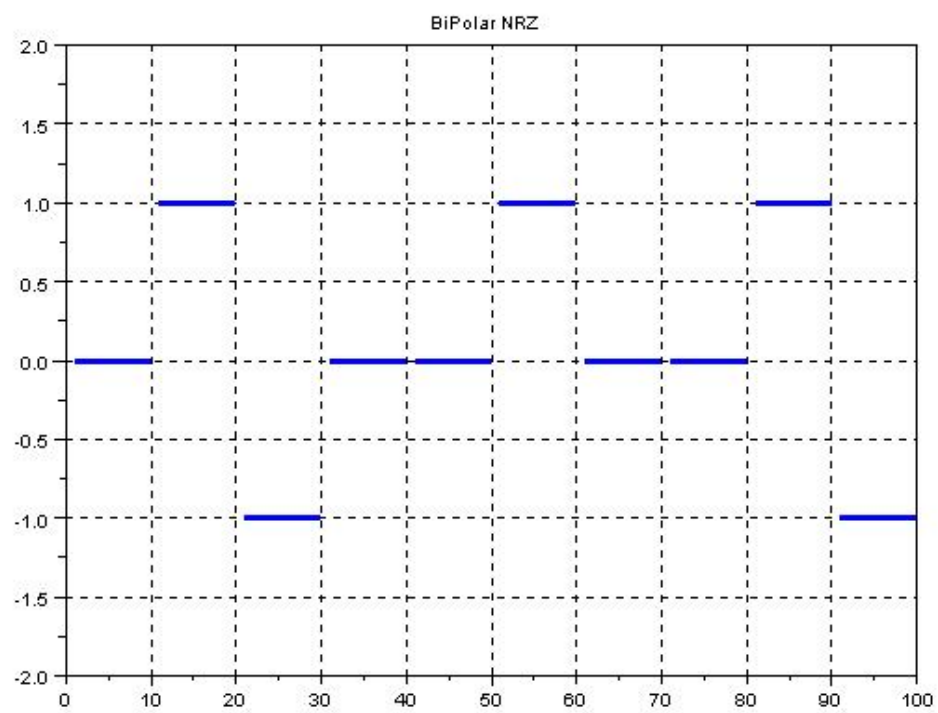


Figure 6.3: Figure6.1c

```

6 a(1) = xor(1,b(1));
7 if(a(1)==1)
8     a_volts(1) = 1;
9 end
10 for k =2:length(b)
11     a(k) = xor(a(k-1),b(k));
12     if(a(k)==1)
13         a_volts(k)=1;
14     else
15         a_volts(k)=-1;
16     end
17 end
18 a = a';
19 a_volts = a_volts';
20 disp(a,'Precoder output in binary form:')
21 disp(a_volts,'Precoder output in volts:')
22 //Duobinary coder output in volts
23 c(1) = 1+ a_volts(1);
24 for k =2:length(a)
25     c(k) = a_volts(k-1)+a_volts(k);
26 end
27 c = c';
28 disp(c,'Duobinary coder output in volts:')
29 //Duobinary decoder output by applying decision
    rule
30 for k =1:length(c)
31     if(abs(c(k))>1)
32         b_r(k) = 0;
33     else
34         b_r(k) = 1;
35     end
36 end
37 b_r = b_r';
38 disp(b_r,'Recovered original sequence at detector
    oupupt:')
39 //Result
40 //Precoder output in binary form:
41 //

```

```

42 // 1.    1.    0.    0.    1.    0.    0.
43 //
44 // Precoder output in volts:
45 //
46 // 1.    1.  - 1.  - 1.    1.  - 1.  - 1.
47 //
48 // Duobinary coder output in volts:
49 //
50 // 2.    2.    0.  - 2.    0.    0.  - 2.
51 //
52 // Recovered original sequence at detector ouput:
53 //
54 // 0.    0.    1.    0.    1.    1.    0.

```

---

**Scilab code Exa 6.3** Generation of bipolar output for duobinary coder

```

1 //Caption:Generation of bipolar output for duobinary
  coder
2 //Example6.3:Operation of Circuit in figure 6.13
3 //for generating bipolar format
4 //page 256 and page 257
5 //Refer Table 6.4
6 clc;
7 x = [0,1,1,0,1,0,0,0,1,1]; //input binary sequence:
  precoder input
8 y(1) = 1;
9 for k =2:length(x)+1
10   y(k) = xor(x(k-1),y(k-1));
11 end
12 y_delay = y(1:$-1);
13 y = y';
14 y_delay = y_delay';
15 disp(y,'Modulo-2 adder output:')
16 disp(y_delay,'Delay element output:')
17 for k = 1:length(y_delay)
18   z(k) = y(k+1)-y_delay(k);
19 end
20 z = z';

```

```

21 disp(z,'differential encoder bipolar output in volts
    :')
22 //Result
23 //Modulo-2 adder output:
24 //      1.      1.      0.      1.      1.      0.      0.      0.
        0.      1.      0.
25 // Delay element output:
26 //      1.      1.      0.      1.      1.      0.      0.      0.
        0.      1.
27 // differential encoder bipolar output in volts:
28 //      0. - 1.      1.      0. - 1.      0.      0.      0.
        1. - 1.

```

---

#### Scilab code CF 6.4 Power Spectra of different binary data formats

```

1 //Caption:Power Spectra of different binary data
  formats
2 //Figure 6.4: Power Spectral Densities of
3 //Different Line Coding Techniques
4 //[1].NRZ Polar Format [2].NRZ Bipolar format
5 //[3].NRZ Unipolar format [4]. Manchester format
6 //Page 241
7 close;
8 clc;
9 //[1]. NRZ Polar format
10 a = input('Enter the Amplitude value:');
11 fb = input('Enter the bit rate:');
12 Tb = 1/fb; //bit duration
13 f = 0:1/(100*Tb):2/Tb;
14 for i = 1:length(f)
15     Sxxf_NRZ_P(i) = (a^2)*Tb*(sinc_new(f(i)*Tb)^2);
16     Sxxf_NRZ_BP(i) = (a^2)*Tb*((sinc_new(f(i)*Tb))^2)
        *((sin(%pi*f(i)*Tb))^2);
17     if (i==1)
18         Sxxf_NRZ_UP(i) = (a^2)*(Tb/4)*((sinc_new(f(i)*Tb
            ))^2)+(a^2)/4;
19     else

```



```

20      Sxxf_NRZ_UP(i) = (a^2)*(Tb/4)*((sinc_new(f(i)*Tb
        ))^2);
21      end
22      Sxxf_Manch(i) = (a^2)*Tb*(sinc_new(f(i)*Tb/2)^2)*
        sin(%pi*f(i)*Tb/2)^2);
23      end
24      //Plotting
25      a = gca();
26      plot2d(f,Sxxf_NRZ_P)
27      poly1= a.children(1).children(1);
28      poly1.thickness = 2; // the tickness of a curve.
29      plot2d(f,Sxxf_NRZ_BP,2)
30      poly1= a.children(1).children(1);
31      poly1.thickness = 2; // the tickness of a curve.
32      plot2d(f,Sxxf_NRZ_UP,5)
33      poly1= a.children(1).children(1);
34      poly1.thickness = 2; // the tickness of a curve.
35      plot2d(f,Sxxf_Manch,9)
36      poly1= a.children(1).children(1);
37      poly1.thickness = 2; // the tickness of a curve.
38      xlabel('f*Tb————>')
39      ylabel('Sxx(f)————>')
40      title('Power Spectral Densities of Different Line
        Codinig Techniques')
41      xgrid(1)
42      legend(['NRZ Polar Format','NRZ Bipolar format','NRZ
        Unipolar format','Manchester format']);
43      //Result
44      //Enter the Amplitude value:1
45      //Enter the bit rate:1

```

---

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

```

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

```

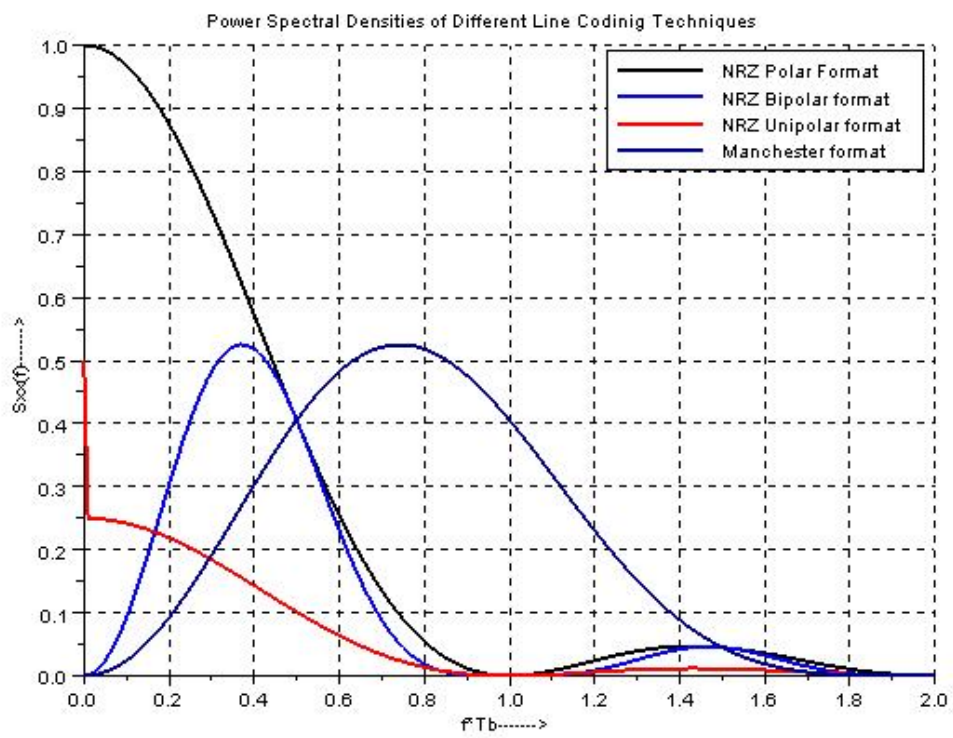


Figure 6.4: Figure6.4

```

4 //page 249
5 rb = input('Enter the bit rate:');
6 Bo = rb/2;
7 t = -3:1/100:3;
8 x = sinc_new(2*Bo*t);
9 plot(t,x)
10 xlabel('t————>');
11 ylabel('p(t)————>');
12 title('SINC Pulse for zero ISI')
13 xgrid(1)
14 //Result
15 //Enter the bit rate:1

```

---

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

```

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

```

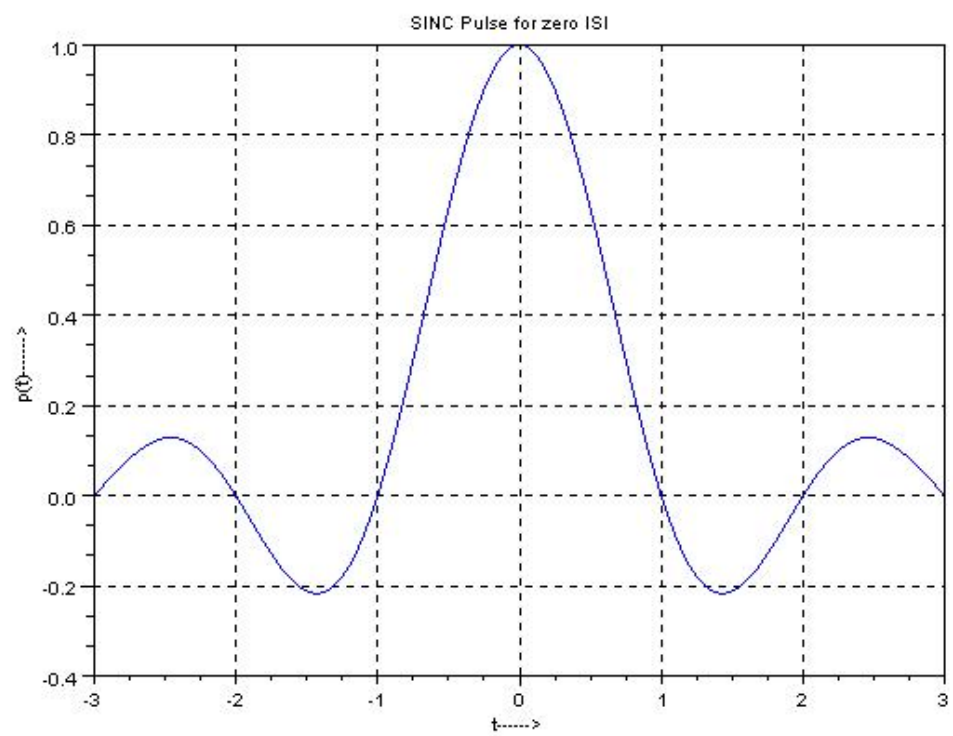


Figure 6.5: Figure6.6

```

21     end
22 end
23 Alpha = Alpha+0.5;
24 end
25 a =gca();
26 plot2d(t,p(1,:))
27 plot2d(t,p(2,:))
28 poly1= a.children(1).children(1);
29 poly1.foreground=2;
30 plot2d(t,p(3,:))
31 poly2= a.children(1).children(1);
32 poly2.foreground=4;
33 poly2.line_style = 3;
34 xlabel('t/Tb————>');
35 ylabel('p(t)————>');
36 title('RAISED COSINE SPECTRUM – Practical Solution
        for ISI')
37 legend(['ROlloff Factor =0', 'ROlloff Factor =0.5', '
        ROlloff Factor =1'])
38 xgrid(1)
39 //Result
40 //Enter the bit rate:1

```

---

**Scilab code CF 6.9** Frequency response of duobinary conversion filter

```

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

```

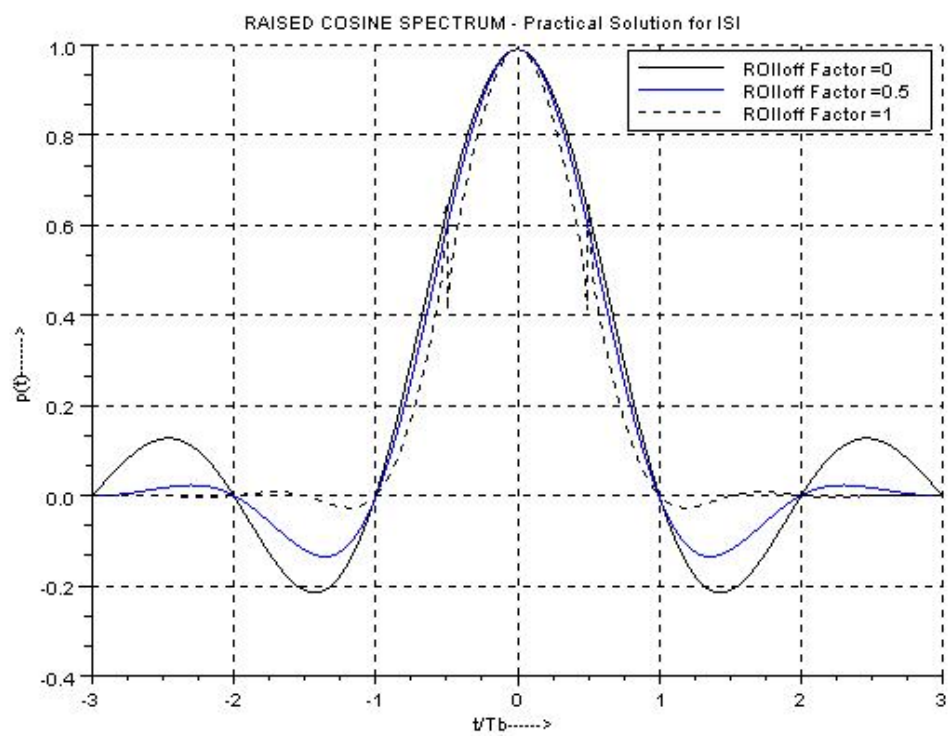


Figure 6.6: Figure6.7

```

11 f = -rb/2:1/100:rb/2;
12 Amplitude_Response = abs(2*cos(%pi*f.*Tb));
13 Phase_Response = -(%pi*f.*Tb);
14 subplot(2,1,1)
15 a=gca();
16 a.x_location = "origin";
17 a.y_location = "origin";
18 plot2d(f,Amplitude_Response,2)
19 poly1= a.children(1).children(1);
20 poly1.thickness = 2; // the tickness of a curve.
21 xlabel('Frequency f——>')
22 ylabel('|H(f)| ——>')
23 title('Amplitude Repsonse of Duobinary Singaling')
24 subplot(2,1,2)
25 a=gca();
26 a.x_location = "origin";
27 a.y_location = "origin";
28 plot2d(f,Phase_Response,5)
29 poly1= a.children(1).children(1);
30 poly1.thickness = 2; // the tickness of a curve.
31 xlabel('
    Frequency f——>')
32 ylabel('
    <H(f) ——>')
33 title('Phase Repsonse of Duobinary Singaling')
34 //Result
35 //Enter the bit rate=8

```

---

**Scilab code CF 6.15** Frequency response of modified duobinary conversion filter

```

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

```

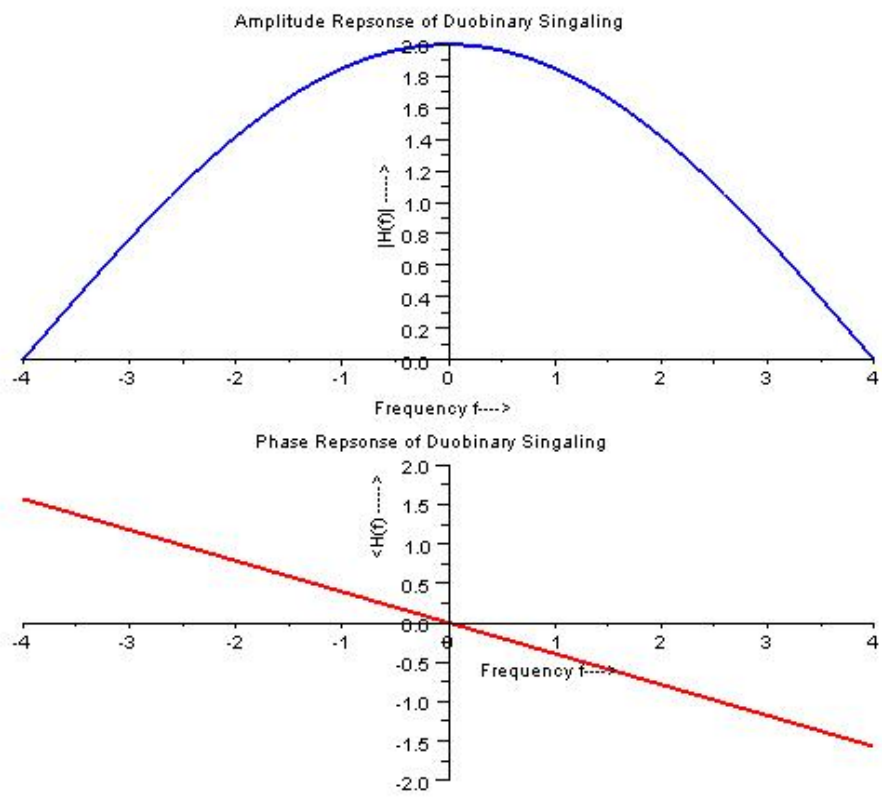


Figure 6.7: Figure6.9



```

5 //page 259
6 clear;
7 close;
8 clc;
9 rb = input('Enter the bit rate=');
10 Tb =1/rb; //Bit duration
11 f = -rb/2:1/100:rb/2;
12 Amplitude_Response = abs(2*sin(2*pi*f.*Tb));
13 Phase_Response = -(2*pi*f.*Tb);
14 subplot(2,1,1)
15 a=gca();
16 a.x_location = "origin";
17 a.y_location = "origin";
18 plot2d(f,Amplitude_Response,2)
19 poly1= a.children(1).children(1);
20 poly1.thickness = 2; // the tickness of a curve.
21 xlabel('Frequency f————>')
22 ylabel('|H(f)| ———>')
23 title('Amplitude Repsonse of Modified Duobinary
        Singaling ')
24 xgrid(1)
25 subplot(2,1,2)
26 a=gca();
27 a.x_location = "origin";
28 a.y_location = "origin";
29 plot2d(f,Phase_Response,5)
30 poly1= a.children(1).children(1);
31 poly1.thickness = 2; // the tickness of a curve.
32 xlabel('
        Frequency f————>')
33 ylabel('
        <H(f) ———>')
34 title('Phase Repsonse of Modified Duobinary
        Singaling ')
35 xgrid(1)
36 //Result
37 //Enter the bit rate=8

```

---

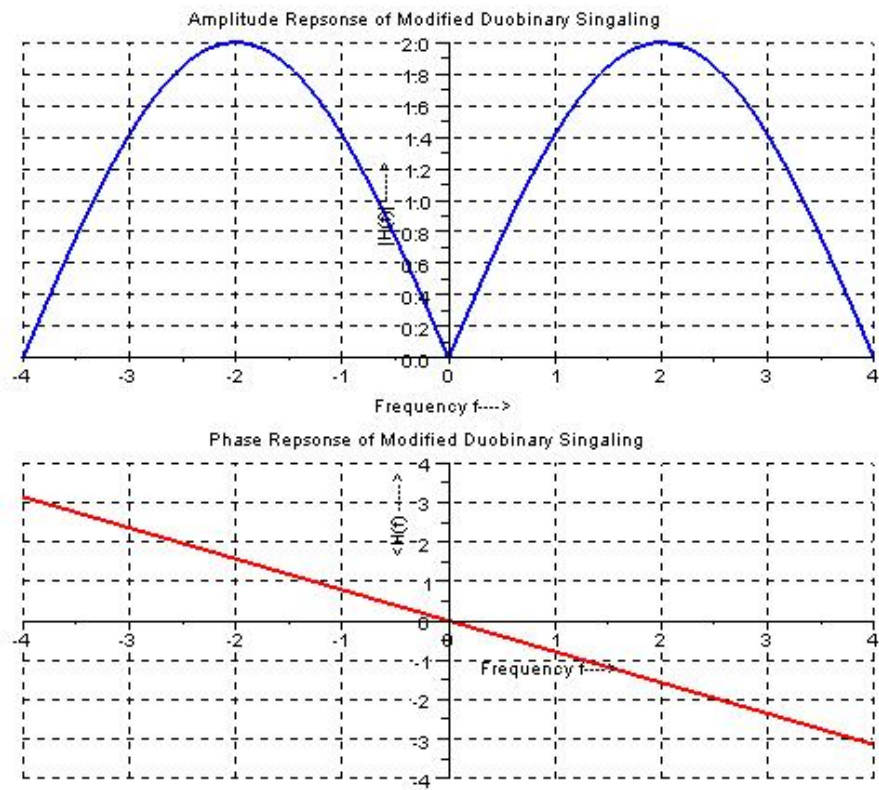


Figure 6.8: Figure6.15

# Chapter 7

## Digital Modulation Techniques

Scilab code Exa 7.1 QPSK Waveform

```
1 //Caption: Waveforms of Different Digital Modulation
  techniques
2 //Example7.1 Signal Space Diagram for coherent QPSK
  system
3 clear;
4 clc;
5 close;
6 M =4;
7 i = 1:M;
8 t = 0:0.001:1;
9 for i = 1:M
10     s1(i,:) = cos(2*%pi*2*t)*cos((2*i-1)*%pi/4);
11     s2(i,:) = -sin(2*%pi*2*t)*sin((2*i-1)*%pi/4);
12 end
13 S1 =[];
14 S2 = [];
15 S = [];
16 Input_Sequence =[0,1,1,0,1,0,0,0];
17 m = [3,1,1,2];
18 for i =1:length(m)
19     S1 = [S1 s1(m(i),:)];
20     S2 = [S2 s2(m(i),:)];
21 end
```

```

22 S = S1+S2;
23 figure
24 subplot(3,1,1)
25 a =gca();
26 a.x_location = "origin";
27 plot(S1)
28 title('Binary PSK wave of Odd-numbered bits of input
        sequence')
29 subplot(3,1,2)
30 a =gca();
31 a.x_location = "origin";
32 plot(S2)
33 title('Binary PSK wave of Even-numbered bits of
        input sequence')
34 subplot(3,1,3)
35 a =gca();
36 a.x_location = "origin";
37 plot(S)
38 title('QPSK waveform')
39 // -sin((2*i-1)*%pi/4)*%i;
40 // annot = dec2bin([0:length(y)-1],log2(M));
41 // disp(y,'coordinates of message points')
42 // disp(annot,'dibits value')
43 // figure;
44 // a =gca();
45 // a.data_bounds = [-1,-1;1,1];
46 // a.x_location = "origin";
47 // a.y_location = "origin";
48 // plot2d(real(y(1)),imag(y(1)),-2)
49 // plot2d(real(y(2)),imag(y(2)),-4)
50 // plot2d(real(y(3)),imag(y(3)),-5)
51 // plot2d(real(y(4)),imag(y(4)),-9)
52 // xlabel('
        Phase ');
53 // ylabel('
        Quadrature ');

```

In—

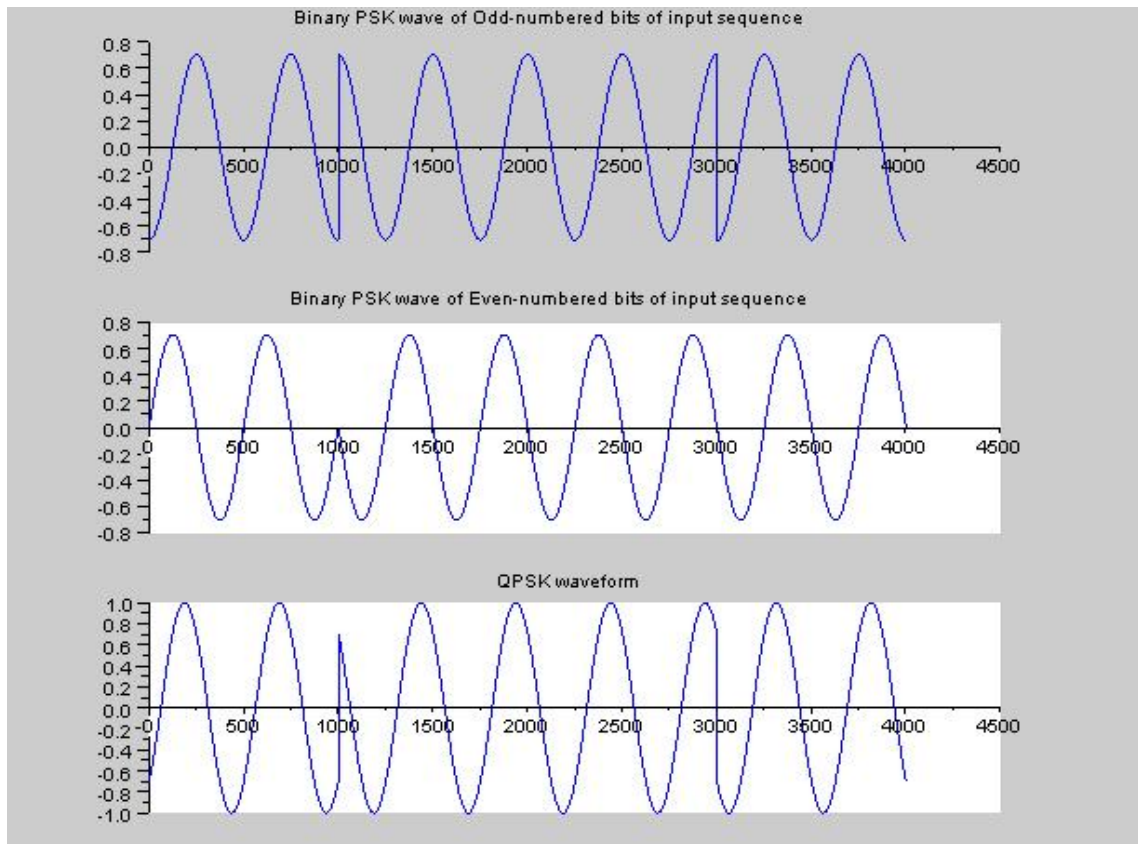


Figure 7.1: Example7.1

```

54 //title('Constellation for QPSK')
55 //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

```

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

```

```

5  clear;
6  clc;
7  close;
8  f = input('Enter the Analog Carrier Frequency in Hz'
           );
9  t = 0:1/512:1;
10 x = sin(2*%pi*f*t);
11 I = input('Enter the digital binary data');
12 //Generation of ASK Waveform
13 Xask = [];
14 for n = 1:length(I)
15     if ((I(n)==1)&(n==1))
16         Xask = [x,Xask];
17     elseif ((I(n)==0)&(n==1))
18         Xask = [zeros(1,length(x)),Xask];
19     elseif ((I(n)==1)&(n~=1))
20         Xask = [Xask,x];
21     elseif ((I(n)==0)&(n~=1))
22         Xask = [Xask,zeros(1,length(x))];
23     end
24 end
25 //Generation of FSK Waveform
26 Xfsk = [];
27 x1 = sin(2*%pi*f*t);
28 x2 = sin(2*%pi*(2*f)*t);
29 for n = 1:length(I)
30     if (I(n)==1)
31         Xfsk = [Xfsk,x2];
32     elseif (I(n)~=1)
33         Xfsk = [Xfsk,x1];
34     end
35 end
36 //Generation of PSK Waveform
37 Xpsk = [];
38 x1 = sin(2*%pi*f*t);
39 x2 = -sin(2*%pi*f*t);
40 for n = 1:length(I)
41     if (I(n)==1)

```

```

42     Xpsk = [Xpsk,x1];
43     elseif (I(n)~=1)
44         Xpsk = [Xpsk,x2];
45     end
46 end
47 figure
48 plot(t,x)
49 xtitle('Analog Carrier Signal for Digital Modulation
        ')
50 xgrid
51 figure
52 plot(Xask)
53 xtitle('Amplitude Shift Keying')
54 xgrid
55 figure
56 plot(Xfsk)
57 xtitle('Frequency Shift Keying')
58 xgrid
59 figure
60 plot(Xpsk)
61 xtitle('Phase Shift Keying')
62 xgrid
63 //Example
64 //Enter the Analog Carrier Frequency 2
65 //Enter the digital binary data[0,1,1,0,1,0,0,1]

```

---

#### Scilab code Exa 7.2 MSK waveforms

```

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

```

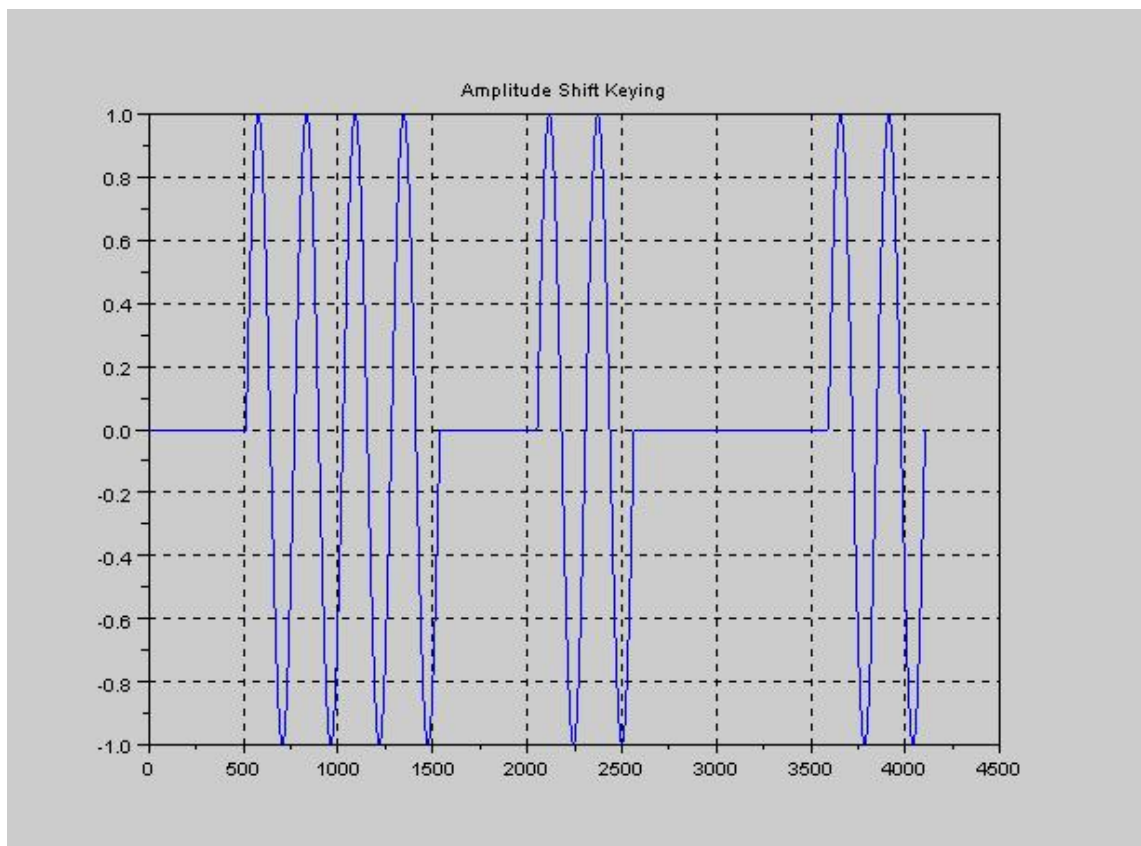


Figure 7.2: Figure7.1a



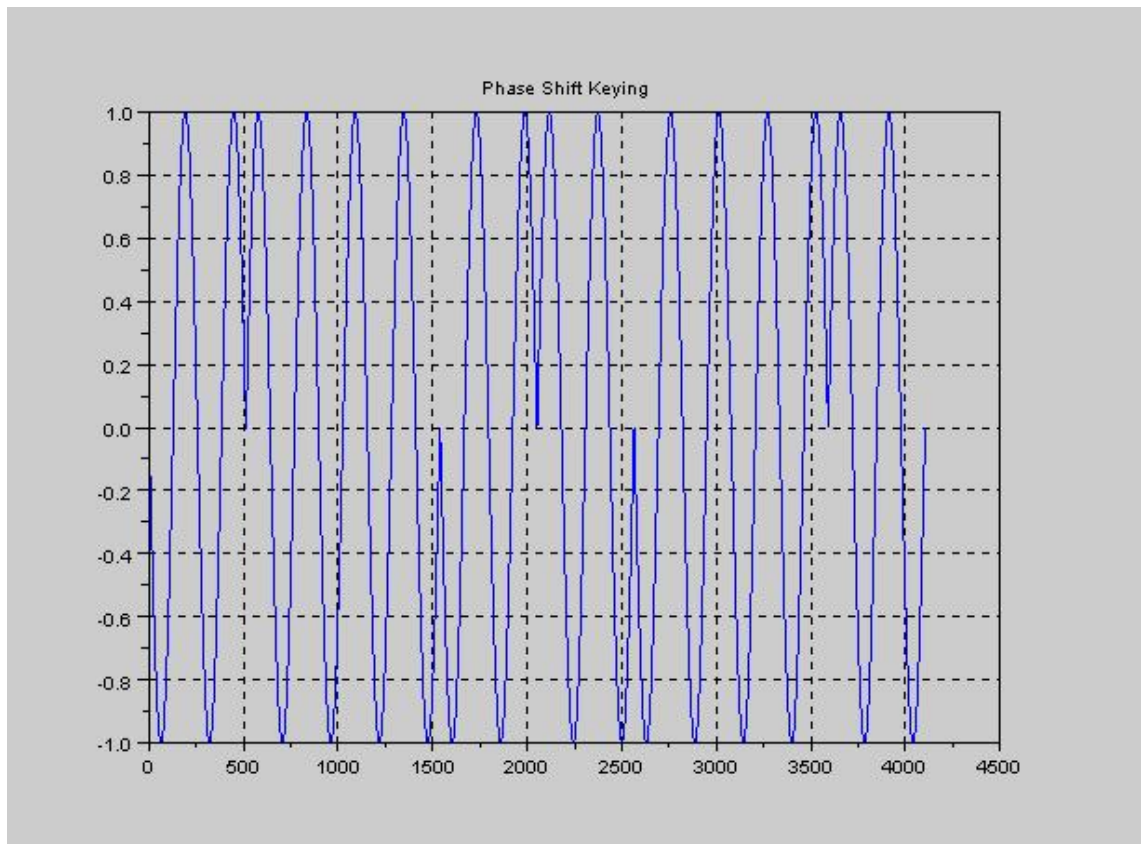


Figure 7.3: Figure7.1b

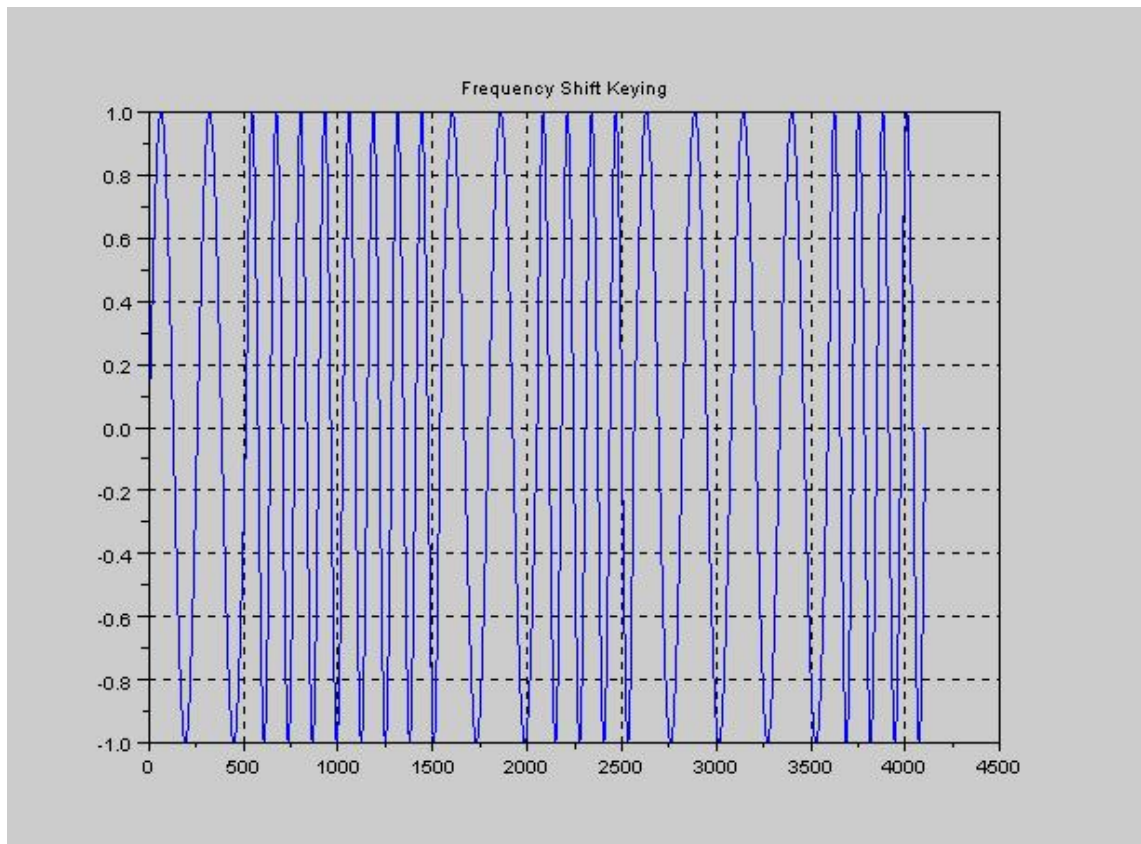


Figure 7.4: Figure7.1c

```

10 t2 = 0:0.01:2*Tb;
11 phi1 = cos(2*%pi*t1).* cos((%pi/(2*Tb))*t1);
12 phi2 = sin(2*%pi*t2).*sin((%pi/(2*Tb))*t2);
13 teta_0 = [0,%pi];
14 teta_tb = [%pi/2,-%pi/2];
15 S1 = [];
16 S2 = [];
17 for i = 1:M
18     s1(i) = cos(teta_0(i));
19     s2(i) = -sin(teta_tb(i));
20     S1 = [S1 s1(i)*phi1];
21     S2 = [S2 s2(1)*phi2];
22 end
23 for i = M:-1:1
24     S1 = [S1 s1(i)*phi1];
25     S2 = [S2 s2(2)*phi2];
26 end
27 Input_Sequence = [1,1,0,1,0,0,0];
28 S = [];
29 t = 0:0.01:1;
30 S = [S cos(0)*cos(2*%pi*t)-sin(%pi/2)*sin(2*%pi*t)];
31 S = [S cos(0)*cos(2*%pi*t)-sin(%pi/2)*sin(2*%pi*t)];
32 S = [S cos(%pi)*cos(2*%pi*t)-sin(%pi/2)*sin(2*%pi*t)
    ];
33 S = [S cos(%pi)*cos(2*%pi*t)-sin(-%pi/2)*sin(2*%pi*t)
    ];
34 S = [S cos(0)*cos(2*%pi*t)-sin(-%pi/2)*sin(2*%pi*t)
    ];
35 S = [S cos(0)*cos(2*%pi*t)-sin(-%pi/2)*sin(2*%pi*t)
    ];
36 S = [S cos(0)*cos(2*%pi*t)-sin(-%pi/2)*sin(2*%pi*t)
    ];
37 y = [s1(1),s2(1);s1(2),s2(1);s1(2),s2(2);s1(1),s2(2)
    ];
38 disp(y,'coordinates of message points')
39 figure
40 subplot(3,1,1)
41 a = gca();

```

```

42 a.x_location = "origin";
43 plot(S1)
44 title('Scaled time function s1*phi1(t)')
45 subplot(3,1,2)
46 a = gca();
47 a.x_location = "origin";
48 plot(S2)
49 title('Scaled time function s2*phi2(t)')
50 subplot(3,1,3)
51 a = gca();
52 a.x_location = "origin";
53 plot(S)
54 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

```

1 //Caption:Signal Space diagram for coherent BPSK
2 //Figure7.2 Signal Space Diagram for coherent BPSK
  system
3 clear
4 clc;
5 close;
6 M =2;
7 i = 1:M;
8 y = cos(2*%pi+(i-1)*%pi);
9 annot = dec2bin([length(y)-1:-1:0],log2(M));
10 disp(y,'coordinates of message points')
11 disp(annot,'Message points')
12 figure;
13 a = gca();
14 a.data_bounds = [-2,-2;2,2];
15 a.x_location = "origin";
16 a.y_location = "origin";
17 plot2d(real(y(1)),imag(y(1)),-9)
18 plot2d(real(y(2)),imag(y(2)),-5)
19 xlabel('

```

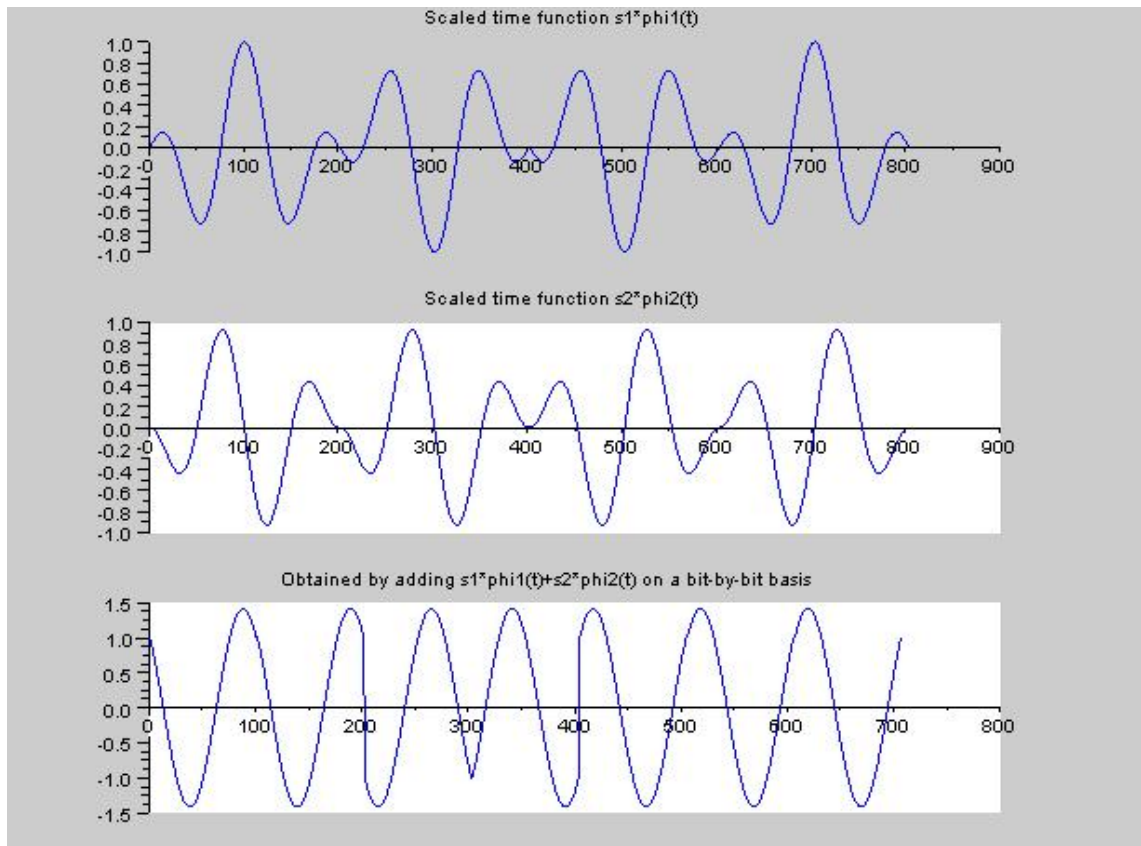


Figure 7.5: Example7.2

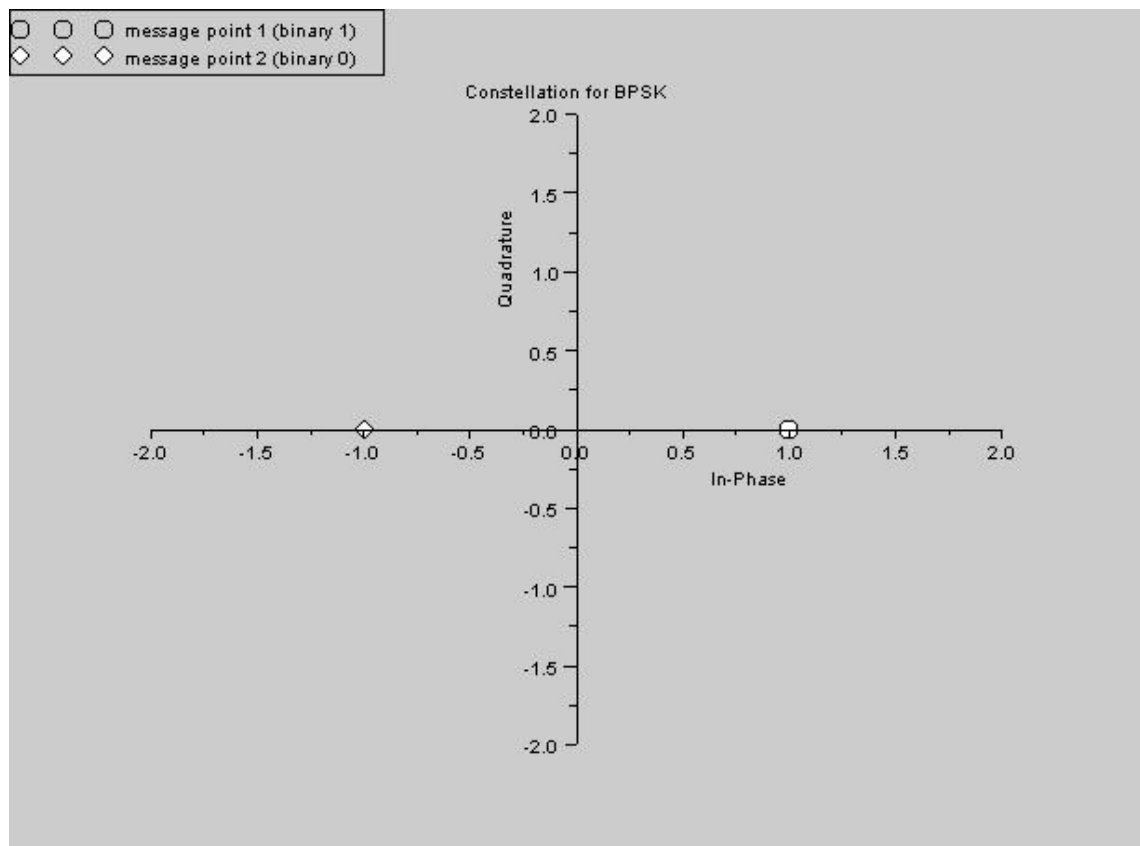


Figure 7.6: Figure7.2

```

    In-Phase');
20 ylabel('

    Quadrature');
21 title('Constellation for BPSK')
22 legend(['message point 1 (binary 1)'; 'message point
    2 (binary 0)'],5)

```

---

**Scilab code Tab 7.3** Illustration the generation of DPSK signal

```

1 //Caption: Illustrating the generation of DPSK signal

```

```

2 //Table7.3 Generation of Differential Phase shift
   keying signal
3 clc;
4 bk = [1,0,0,1,0,0,1,1]; //input digital sequence
5 for i = 1:length(bk)
6     if(bk(i)==1)
7         bk_not(i) = ~1;
8     else
9         bk_not(i)= 1;
10    end
11 end
12 dk_1(1) = 1&bk(1); //initial value of differential
   encoded sequence
13 dk_1_not(1)=0&bk_not(1);
14 dk(1) = xor(dk_1(1),dk_1_not(1)) //first bit of dpsk
   encoder
15 for i=2:length(bk)
16     dk_1(i) = dk(i-1);
17     dk_1_not(i) = ~dk(i-1);
18     dk(i) = xor((dk_1(i)&bk(i)),(dk_1_not(i)&bk_not(i)
        ));
19 end
20 for i =1:length(dk)
21     if(dk(i)==1)
22         dk_radians(i)=0;
23     elseif(dk(i)==0)
24         dk_radians(i)=%pi;
25     end
26 end
27 disp('Table 7.3 Illustrating the Generation of DPSK
   Signal')
28 disp('
   -----
   ')
29 disp(bk, '(bk)')
30 bk_not = bk_not';
31 disp(bk_not, '(bk_not)')
32 dk = dk';

```

```

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

---

#### Scilab code CF 7.4 Signal Space diagram for coherent BFSK

```

1 //Caption:Signal Space diagram for coherent BFSK
2 //Figure7.4 Signal Space Diagram for coherent BFSK
  system
3 clear
4 clc;
5 close;
6 M =2;
7 y = [1,0;0,1];
8 annot = dec2bin([M-1:-1:0],log2(M));
9 disp(y, 'coordinates of message points')
10 disp(annot, 'Message points')
11 figure;
12 a =gca();
13 a.data_bounds = [-2,-2;2,2];
14 a.x_location = "origin";
15 a.y_location = "origin";
16 plot2d(y(1,1),y(1,2),-9)
17 plot2d(y(2,1),y(2,2),-5)
18 xlabel('
      In-Phase');
19 ylabel('
      Quadrature');
20 title('Constellation for BFSK')
21 legend(['message point 1 (binary 1)'; 'message point
      2 (binary 0)'],5)
```

---



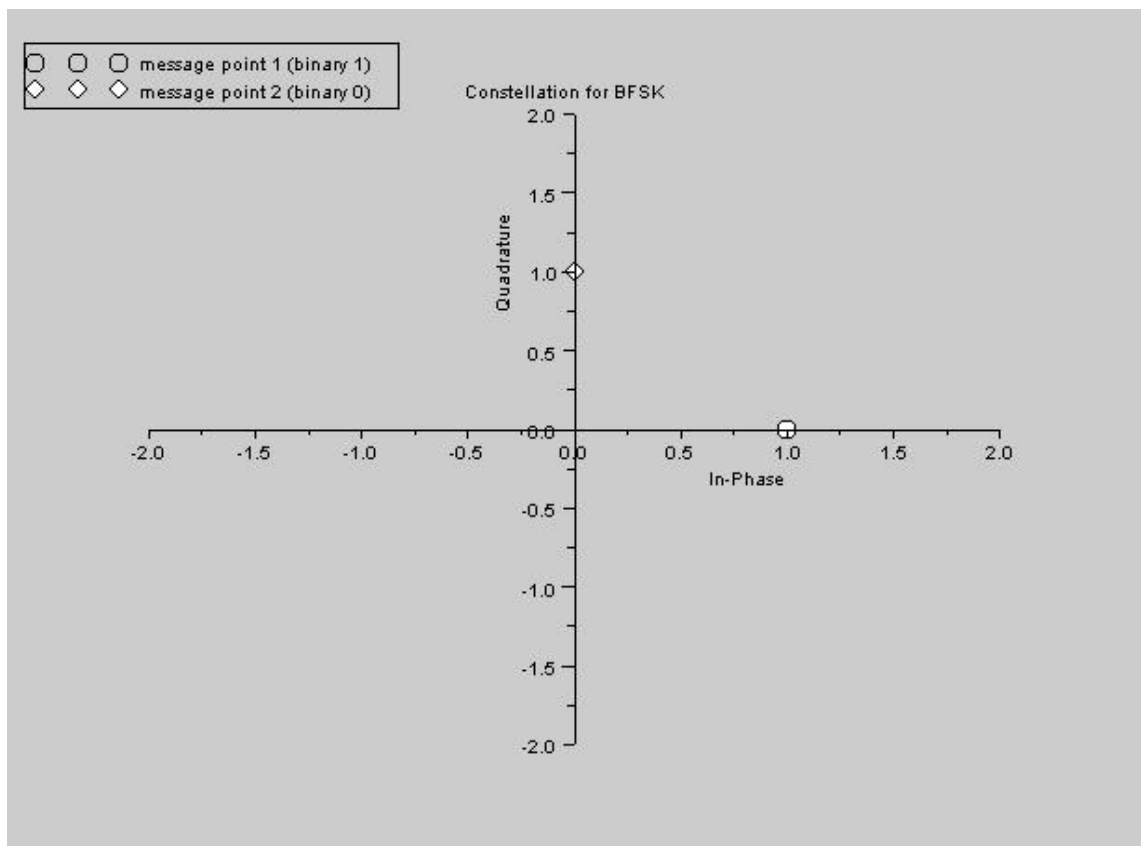


Figure 7.7: Figure 7.4

**Scilab code CF 7.6** Signal space diagram for coherent QPSK waveform

```

1 //Caption:Signal space diagram for coherent QPSK
  waveform
2 //Figure7.6 Signal Space Diagram for coherent QPSK
  system
3 clear
4 clc;
5 close;
6 M =4;
7 i = 1:M;
8 y = cos((2*i-1)*%pi/4)-sin((2*i-1)*%pi/4)*%i;
9 annot = dec2bin([0:M-1],log2(M));
10 disp(y,'coordinates of message points')
11 disp(annot,'dibits value')
12 figure;
13 a =gca();
14 a.data_bounds = [-1,-1;1,1];
15 a.x_location = "origin";
16 a.y_location = "origin";
17 plot2d(real(y(1)),imag(y(1)),-2)
18 plot2d(real(y(2)),imag(y(2)),-4)
19 plot2d(real(y(3)),imag(y(3)),-5)
20 plot2d(real(y(4)),imag(y(4)),-9)
21 xlabel('
                                     In-
                                     Phase');
22 ylabel('
                                     Quadrature');
23 title('Constellation for QPSK')
24 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

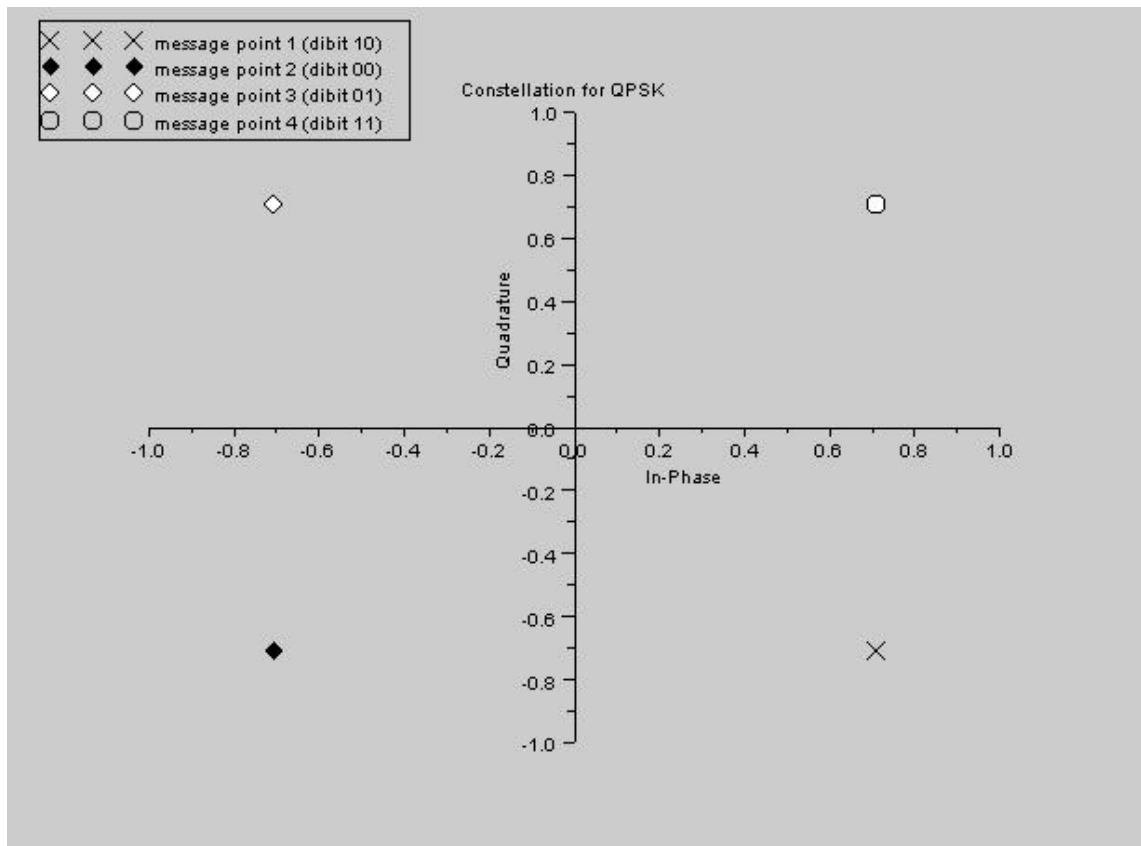


Figure 7.8: Figure7.6

```

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

```

---

**Scilab code Tab 7.7** Bandwidth efficiency of M ary FSK signals

```

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

```

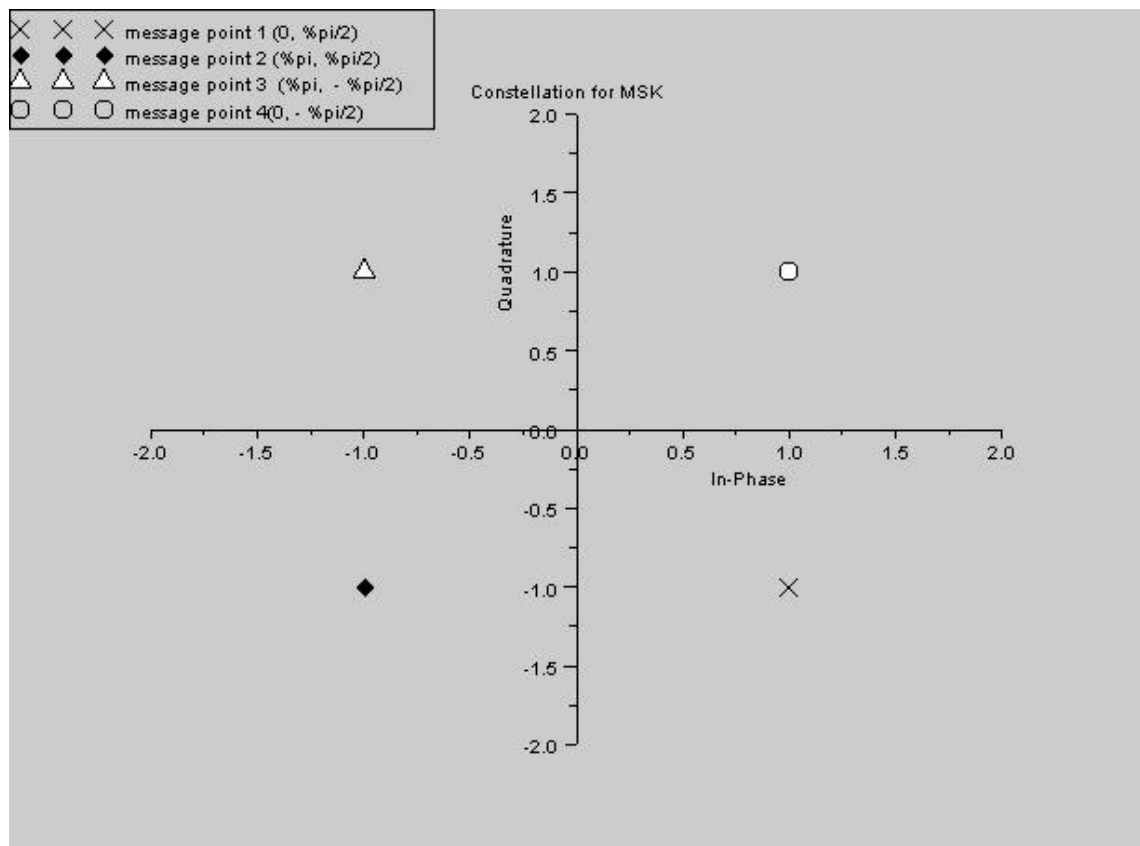


Figure 7.9: Figure7.12

```

-----
')
12 disp(M, 'M')
13 disp('
-----
')
14 disp(Ruo, 'r in bits/s/Hz')
15 disp('
-----
')
```

---

Scilab code CF 7.29 Power Spectra of BPSK and BFSK signals

```

1 //Caption:Power Spectra of BPSK and BFSK signals
2 //Figure7.29:Comparison of Power Spectral Densities
  of BPSK
3 //and BFSK
4 clc;
5 rb = input('Enter the bit rate=');
6 Eb = input('Enter the energy of the bit=');
7 f = 0:1/100:8/rb;
8 Tb = 1/rb; //Bit duration
9 for i= 1:length(f)
10     if(f(i)==(1/(2*Tb)))
11         SB_FSK(i)=Eb/(2*Tb);
12     else
13         SB_FSK(i) = (8*Eb*(cos(%pi*f(i)*Tb)^2))/((%pi
            ^2)*(((4*(Tb^2)*(f(i)^2))-1)^2));
14     end
15     SB_PSK(i)=2*Eb*(sinc_new(f(i)*Tb)^2);
16 end
17 a=gca();
18 plot(f*Tb,SB_FSK/(2*Eb))
19 plot(f*Tb,SB_PSK/(2*Eb))
20 poly1= a.children(1).children(1);
21 poly1.foreground = 6;
22 xlabel('Normalized Frequency ——>')
23 ylabel('Normalized Power Spectral Density——>')
24 title('PSK Vs FSK Power Spectra Comparison')
25 legend(['Frequency Shift Keying','Phase Shift Keying'
    ''])
26 xgrid(1)
27 //Result
28 //Enter the bit rate in bits per second:2
29 //Enter the Energy of bit:1

```

---

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

```

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

```

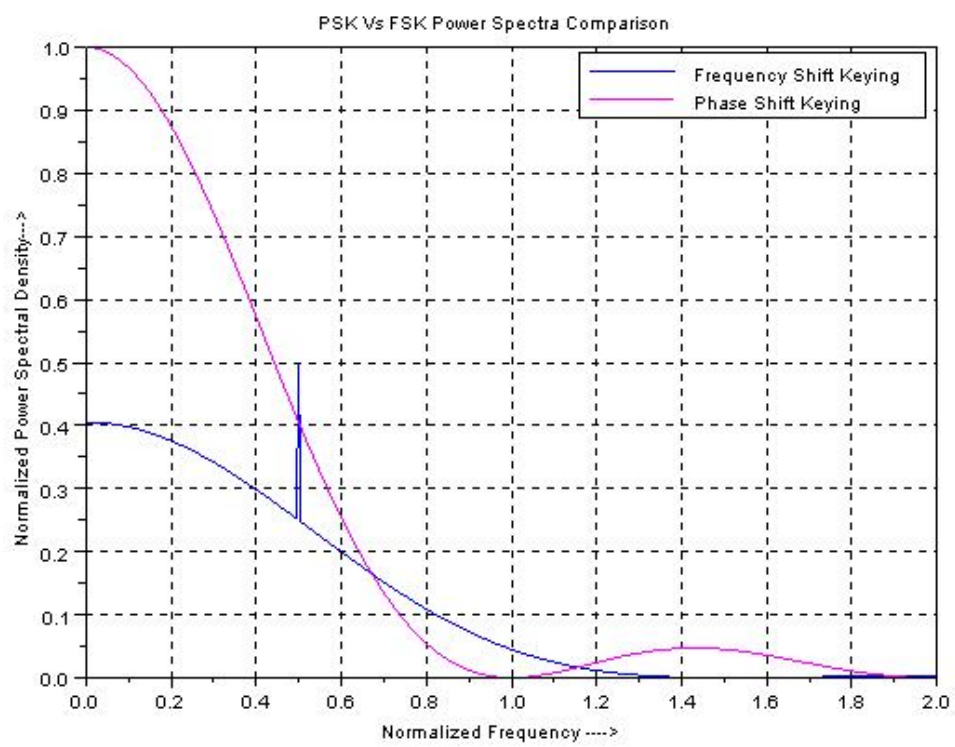


Figure 7.10: Figure7.29

```

        Spectrums
3  //clear;
4  //close;
5  //clc;
6  rb = input('Enter the bit rate in bits per second:')
    ;
7  Eb = input('Enter the Energy of bit:');
8  f = 0:1/(100*rb):(4/rb);
9  Tb = 1/rb; //bit duration in seconds
10 for i = 1:length(f)
11     if(f(i)==0.5)
12         SB_MSK(i) = 4*Eb*f(i);
13     else
14         SB_MSK(i) = (32*Eb/(%pi^2))*(cos(2*%pi*Tb*f(i))
            /((4*Tb*f(i))^2-1))^2;
15     end
16     SB_QPSK(i)= 4*Eb*sinc_new((2*Tb*f(i)))^2;
17 end
18 a = gca();
19 plot(f*Tb,SB_MSK/(4*Eb));
20 plot(f*Tb,SB_QPSK/(4*Eb));
21 poly1= a.children(1).children(1);
22 poly1.foreground = 3;
23 xlabel('Normalized Frequency ——>')
24 ylabel('Normalized Power Spectral Density——>')
25 title('QPSK Vs MSK Power Spectra Comparison')
26 legend(['Minimum Shift Keying','QPSK'])
27 xgrid(1)
28 //Result
29 //Enter the bit rate in bits per second:2
30 //Enter the Energy of bit:1

```

---

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

```

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

```



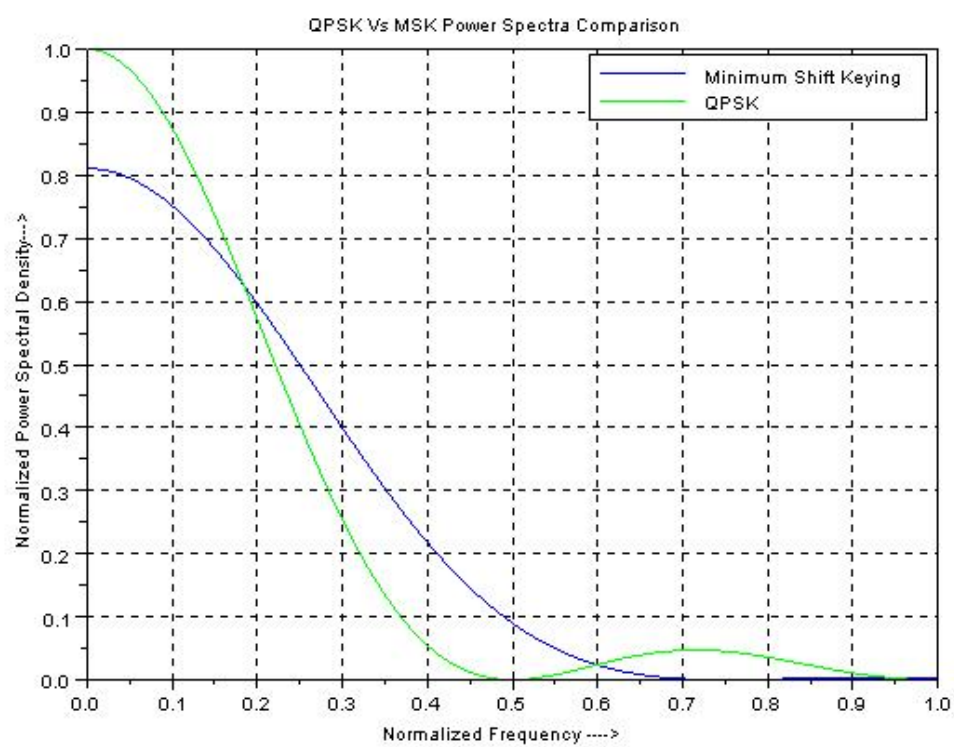


Figure 7.11: Figure7.30

```

3  rb = input('Enter the bit rate=');
4  Eb = input('Enter the energy of the bit=');
5  f = 0:1/100:rb;
6  Tb = 1/rb; //Bit duration
7  M = [2,4,8];
8  for j = 1:length(M)
9      for i= 1:length(f)
10         SB_PSK(j,i)=2*Eb*(sinc_new(f(i)*Tb*log2(M(j)))
            ^2)*log2(M(j));
11     end
12 end
13 a=gca();
14 plot2d(f*Tb,SB_PSK(1,:)/(2*Eb))
15 plot2d(f*Tb,SB_PSK(2,:)/(2*Eb),2)
16 plot2d(f*Tb,SB_PSK(3,:)/(2*Eb),5)
17 xlabel('Normalized Frequency ---->')
18 ylabel('Normalized Power Spectral Density---->')
19 title('Power Spectra of M-ary signals for M =2,4,8')
20 legend(['M=2', 'M=4', 'M=8'])
21 xgrid(1)
22 //Result
23 //Enter the bit rate in bits per second:2
24 //Enter the Energy of bit:1

```

---

**Scilab code CF 7.41** Matched Filter output of rectangular pulse

```

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

```

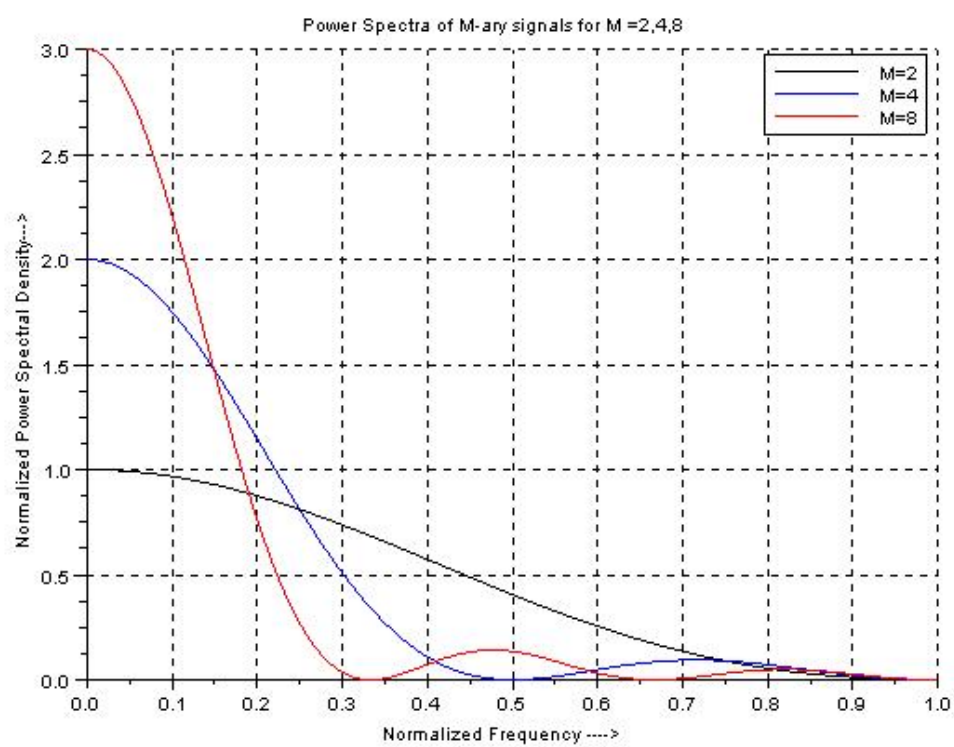


Figure 7.12: Figure7.31

```

12     if(h(i)<0.01)
13         h(i) =0;
14     end
15 end
16 h = h-T;
17 t1 = 0:length(h)-1;
18 figure
19 a =gca();
20 a.data_bounds = [0,0;6,4];
21 plot2d(t,g,5)
22 xlabel('t——>')
23 ylabel('g(t)——>')
24 title('Rectangular pulse duration T = 4, a =2')
25 figure
26 plot2d(t1,h,6)
27 xlabel('t——>')
28 ylabel('Matched Filter output')
29 title('Output of filter matched to rectangular pulse
        g(t)')

```

---

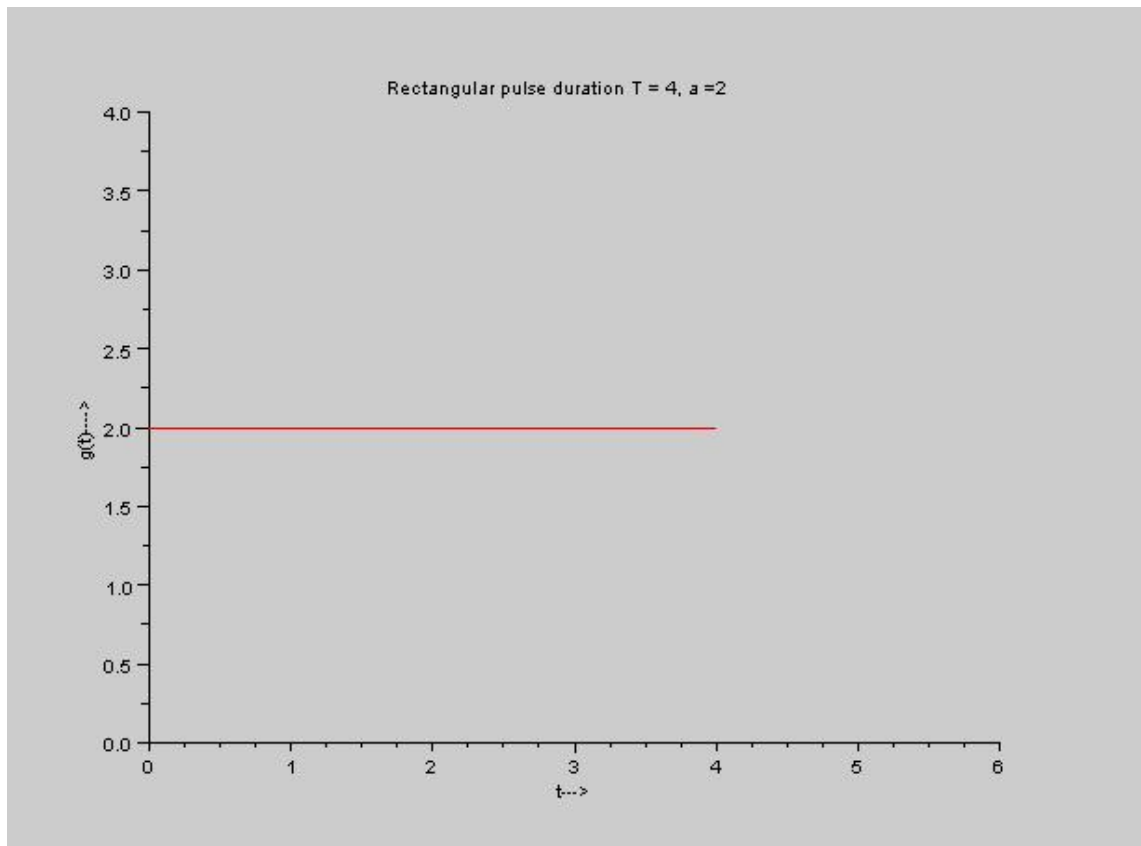


Figure 7.13: Figure7.41a

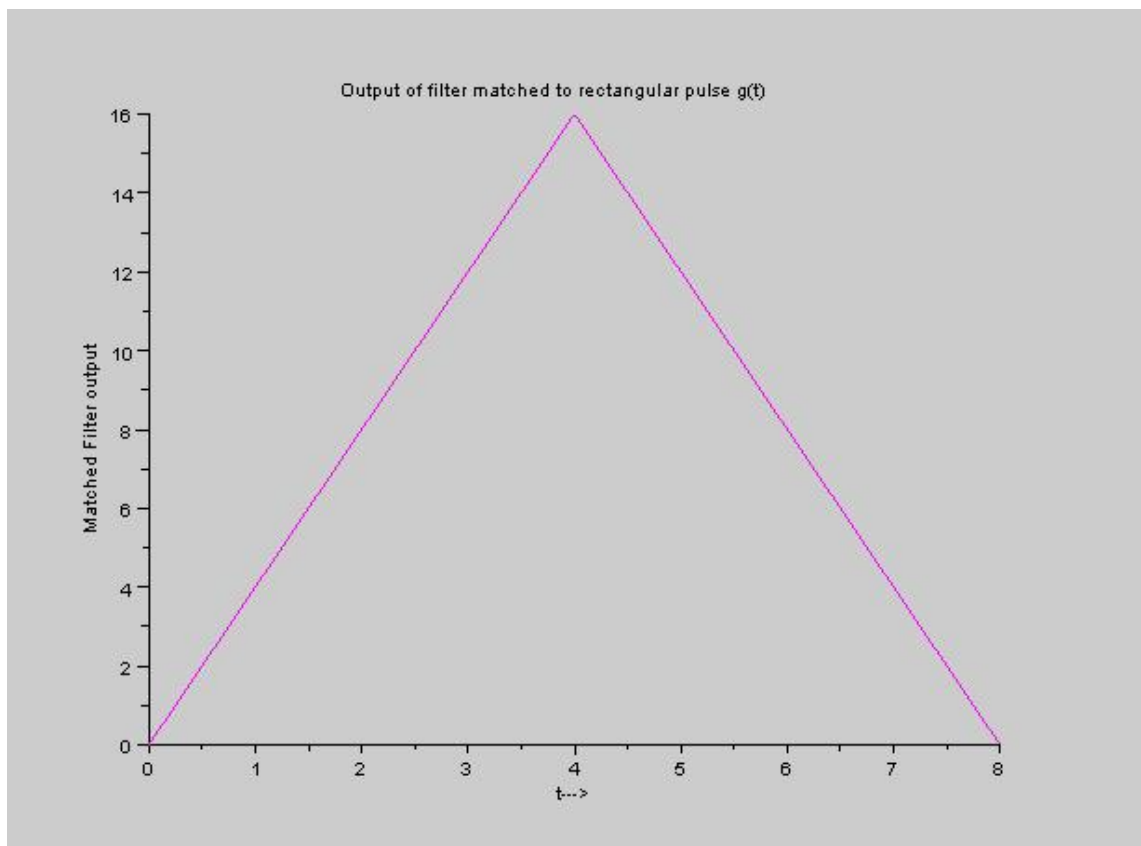


Figure 7.14: Figure7.41b

# Chapter 8

## Error-Control Coding

Scilab code Exa 8.1 Repetition Codes

```
1 //Caption:Repetition Codes
2 //Example8.1:Repetition Codes
3 clear;
4 clc;
5 n =5; //block of identical 'n' bits
6 k =1; //one bit
7 m = 1; // bit value = 1
8 I = eye(n-k,n-k); //Identity matrix
9 P = ones(1,n-k); //coefficient matrix
10 H = [I P']; //parity-check matrix
11 G = [P 1]; //generator matrix
12 x = m.*G; //code word
13 disp(G,'generator matrix');
14 disp(H,'parity-check matrix');
15 disp(x,'code word for binary one input');
```

---

Scilab code Exa 8.2 Hamming Codes

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

```

6 n = 7; //block length
7 m = n-k; //Number of parity bits
8 I = eye(k,k); //identity matrix
9 disp(I, 'identity matrix Ik')
10 P = [1,1,0;0,1,1;1,1,1;1,0,1]; //coefficient matrix
11 disp(P, 'coefficient matrix P')
12 G = [P I]; //generator matrix
13 disp(G, 'generator matrix G')
14 H = [eye(k-1,k-1) P']; //parity check matrix
15 disp(H, 'parity check matrix H')
16 //message bits
17 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;
18 //
19 C = m*G;
20 C = modulo(C,2);
21 disp(C, 'Code words of (7,4) Hamming code')

```

---

### Scilab code Exa 8.3 Hamming Codes Revisited

```

1 //Caption:Hamming Codes Revisited
2 //Example8.3:(7,4) Hamming Code Revisited
3 //message sequence = [1,0,0,1]
4 //D = poly(0,D);
5 clc;
6 D = poly(0, 'D');
7 g = 1+D+0+D^3; //generator polynomial
8 m = (D^3)*(1+0+0+D^3); //message sequence
9 [r,q] = pdiv(m,g);
10 p = coeff(r);
11 disp(r, 'remainder in polynomial form')
12 disp(p, 'Parity bits are:')
13 G = [g;g*D;g*D^2;g*D^3];
14 G = coeff(G);
15 disp(G, 'G')
16 G(3,:) = G(3,:)+G(1,:);
17 G(3,:) = modulo(G(3,:),2);

```



```

18 G(4,:) = G(1,:)+G(2,:)+G(4,:);
19 G(4,:) = modulo(G(4,:),2);
20 disp(G, 'Generator Matrix G =')
21 h = 1+D^-1+D^-2+D^-4;
22 H_D = [D^4*h;D^5*h;D^6*h];
23 H_num =numer(H_D);
24 H = coeff(H_num);
25 H(1,:) =H(1,:)+H(3,:);
26 H(1,:) = modulo(H(1,:),2);
27 disp(H, 'Partiy Check matrix H =')

```

---

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

```

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

```

```

18 disp('3          0          1 1 1')
19 disp('4          1          0 1 1')
20 disp('
-----
')
```

---

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

```

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

---

**Scilab code Exa 8.6** Reed-Solomon Codes

```

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

```

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

```

---

#### Scilab code Exa 8.7 Convolutional Encoding - Time domain approach

```

1 //Caption:Convolutional Encoding – Time domain
  approach
2 //Example8.7:Convolutional Code Generation
3 //Time Domain Approach
4 close;
5 clc;
6 g1 = input('Enter the input Top Adder Sequence:=')
7 g2 = input('Enter the input Bottom Adder Sequence:=')
  )
8 m = input('Enter the message sequence:=')
9 x1 = round(convol(g1,m));
10 x2 = round(convol(g2,m));
11 x1 = modulo(x1,2);
12 x2 = modulo(x2,2);
13 N = length(x1);
14 for i =1:length(x1)
15     x(i,:) =[x1(N-i+1),x2(N-i+1)];
16 end
17 x = string(x)
18 disp(x)
19 //Result
20 //Enter the input Top Adder Sequence:=[1,1,1]
21 //Enter the input Bottom Adder Sequence:=[1,0,1]
22 //Enter the message sequence:=[1,1,0,0,1]
23 //x =
24 //!1 1 !

```

```

25 //!      !
26 //!1    0  !
27 //!      !
28 //!1    1  !
29 //!      !
30 //!1    1  !
31 //!      !
32 //!0    1  !
33 //!      !
34 //!0    1  !
35 //!      !
36 //!1    1  !

```

---

**Scilab code Exa 8.8** Convolutional Encoding Transform domain approach

```

1 //Caption:Convolutional Encoding Transform domain
  approach
2 //Example8.8:Convolutional code – Transform domain
  approach
3 clc;
4 D = poly(0,'D');
5 g1D = 1+D+D^2; //generator polynomial 1
6 g2D = 1+D^2; //generator polynomial 2
7 mD = 1+0+0+D^3+D^4; //message sequence polynomial
  representation
8 x1D = g1D*mD; //top output polynomial
9 x2D = g2D*mD; //bottom output polynomial
10 x1 = coeff(x1D);
11 x2 = coeff(x2D);
12 disp(modulo(x1,2),'top output sequence')
13 disp(modulo(x2,2),'bottom output sequence')
14 //Result
15 //top output sequence
16 //      1.      1.      1.      1.      0.      0.      1.
17 //
18 // bottom output sequence
19 //      1.      0.      1.      1.      1.      1.      1.

```

---

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

```
1 //Caption:Fano metric for binary symmetric channel
   using convolutional code
2 //Example8.11: Convolutional code for binary
   symmetric channel
3 clc;
4 r = 1/2; //code rate
5 n =2; //number of bits
6 pe = 0.04; //transition probability
7 p = 1-pe; // probability of correct reception
8 gama_1 = 2*log2(p)+2*(1-r); //branch metric for
   correct reception
9 gamma_2 = log2(pe*p)+1; //branch metric for any one
   correct reception
10 gamma_3 = 2*log2(pe)+1; //branch metric for no
   correct reception
11 disp(gama_1,'branch metric for correct reception')
12 disp(gamma_2,'branch metric for any one correct
   reception')
13 disp(gamma_3,'branch metric for no correct reception
   ')
14 //branch metric for correct reception
15 //      0.8822126
16 // branch metric for any one correct reception
17 //      - 3.7027499
18 // branch metric for no correct reception
19 //      - 8.2877124
```

---

## Chapter 9

# Spread-Spectrum Modulation

Scilab code Exa 9.1 PN sequence generation

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

```

22 N = 2^m-1;
23 disp('Table 9.1 Range of PN Sequence lengths')
24 disp('
    -----
    ')
25 disp('Length of shift register (m)')
26 disp(m)
27 disp('PN sequence Length (N)')
28 disp(N)
29 disp('
    -----
    ')
30 //RESULTEnter the period of the signal 7
31 // The PN sequence at step 1.
32 // x= 1. 0. 0.
33 // The PN sequence at step 2.
34 // x= 1. 1. 0.
35 // The PN sequence at step 3.
36 // x= 1. 1. 1.
37 // The PN sequence at step 4.
38 // x= 0. 1. 1.
39 // The PN sequence at step 5.
40 // x= 1. 0. 1.
41 // The PN sequence at step 6.
42 // x= 0. 1. 0.
43 // The PN sequence at step 7.
44 // x= 0. 0. 1.

```

---

**Scilab code Exa 9.2** Maximum length sequence property

```

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

```

```

9  x2 =0;
10 x3 =0;
11 N = input('Enter the period of the signal')
12 one_count = 0;
13 zero_count = 0;
14 for i =1:N
15     x3 =x2;
16     x2 =x1;
17     x1 = x0;
18     x0 =xor(x1,x3);
19     disp(i,'The PN sequence at step')
20     x = [x1 x2 x3];
21     disp(x,'x=')
22     C(i) = x3;
23     if(C(i)==1)
24         C_level(i)=1;
25         one_count = one_count+1;
26     elseif(C(i)==0)
27         C_level(i)=-1;
28         zero_count = zero_count+1;
29     end
30 end
31 disp(C,'Output Sequence')//refer equation 9.4
32 disp(C_level,'Output Sequence levels')//refer
    equation 9.5
33 if(zero_count < one_count)
34     disp(one_count,'Number of 1s in the given PN
        sequence')
35     disp(zero_count,'Number of 0s in the given PN
        sequence')
36     disp('Property 1 (Balance property) is satisfied'
        )
37 end
38 Rc_tuo = corr(C_level,N);
39 t = 1:2*length(C_level);
40 //
41 figure
42 a =gca();

```



```

43 a.x_location = "origin";
44 plot2d(t,[C_level; C_level])
45 xlabel('                                t')
46 title('Waveform of maximum-length sequence [0 0 1 1
      1 0 1 0 0 1 1 1 0 1]')
47 //
48 figure
49 a =gca();
50 a.x_location ="origin";
51 a.y_location ="origin";
52 plot2d([-length(Rc_tuo)+1:-1,0:length(Rc_tuo)-1],[
      Rc_tuo($:-1:2),Rc_tuo],5)
53 xlabel('
      tuo')
54 ylabel('
      Rc(tuo)')
55 title('Autocorrelation of maximum-length sequence')

```

---

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

```

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

```

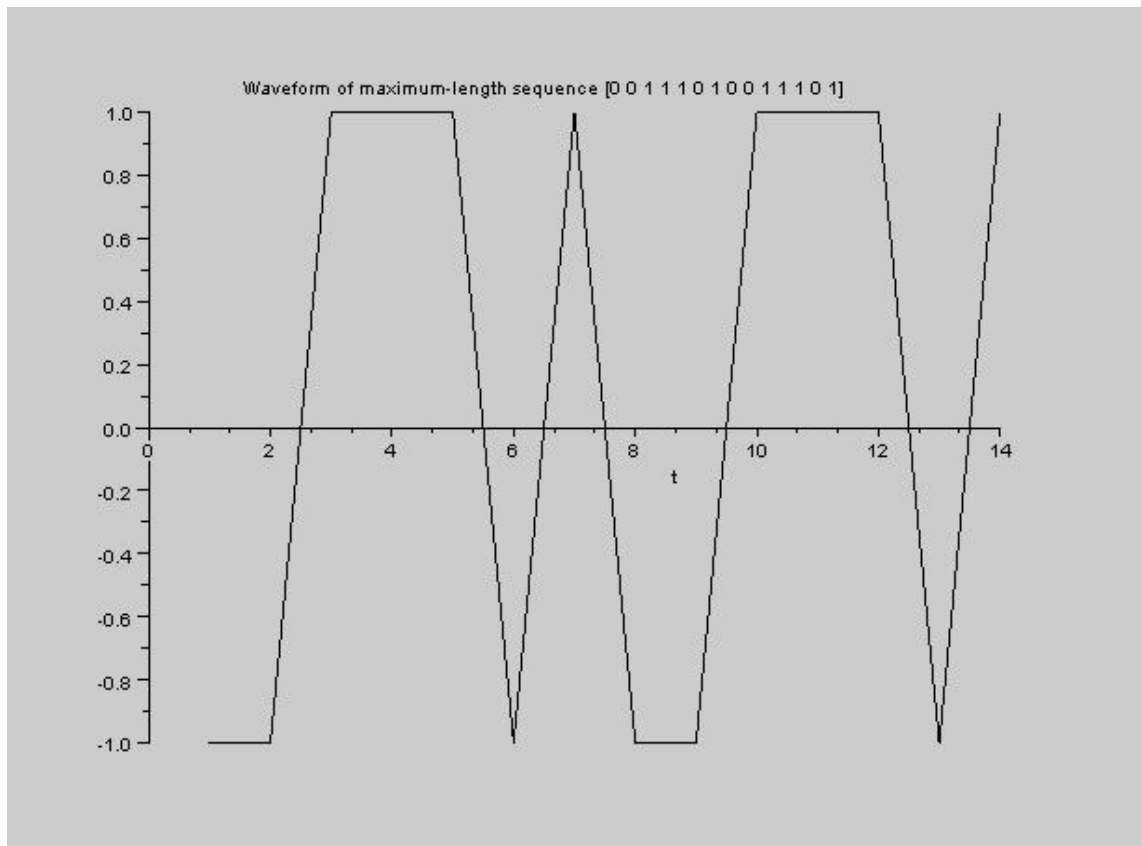


Figure 9.1: Example9.2a

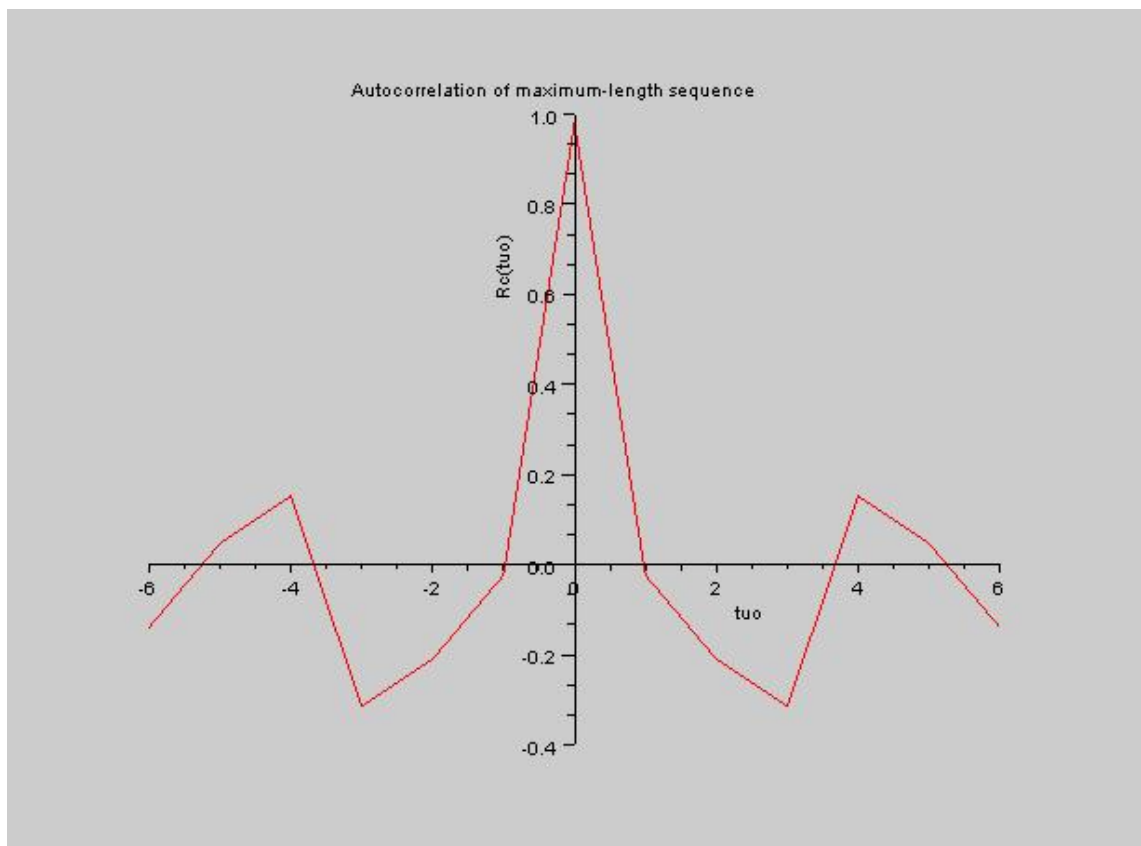


Figure 9.2: Example9.2b

```

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

```

---

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

```

1 //Caption:Slow and Fast Frequency hopping: FH/MFSK
2 //Example9.4 and Example9.5: Parameters of FH/MFSK
  signal
3 //Slow and Fast Frequency Hopping
4 clear;
5 close;
6 clc;
7 K =2; //number of bits per symbol
8 M = 2^K; //Number of MFSK tones
9 N = 2^M-1; //Period of the PN sequence
10 k = 3; //length of PN sequence per hop
11 disp(K, 'number of bits per symbol K =')
12 disp(M, 'Number of MFSK tones M=')
13 disp(N, 'Period of the PN sequence N =')
14 disp(k, 'length of PN sequence per hop k =')
15 disp(2^k, 'Total number of frequency hops =')
16 //Result
17 //number of bits per symbol K = 2.
18 //Number of MFSK tones M = 4.
19 //Period of the PN sequence N = 15.
20 //length of PN sequence per hop k = 3.
21 //Total number of frequency hops = 8.

```

---

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

```

1 //Caption:Direct Sequence Spread Coherent BPSK

```

```

2 //Figure 9.4: Generation of waveforms in DS/BPSK
  spread spectrum transmitter
3 clear;
4 close;
5 clc;
6 t = 0:13;
7 N = 7;
8 wt = 0:0.01:1;
9 bt = [1*ones(1,N) -1*ones(1,N)];
10 ct = [0,0,1,1,1,0,1,0,0,1,1,1,0,1];
11 ct_polar = [-1,-1,1,1,1,-1,1,-1,-1,1,1,1,-1,1];
12 mt = bt.*ct_polar;
13 Carrier = 2*sin(wt*2*pi);
14 st = [];
15 for i = 1:length(mt)
16     st = [st mt(i)*Carrier];
17 end
18 //
19 figure
20 subplot(3,1,1)
21 a = gca();
22 a.x_location = "origin";
23 a.y_location = "origin";
24 a.data_bounds = [0,-2;20,2];
25 plot2d2(t,bt,5)
26 xlabel('
    t')
27 title('Data b(t)')
28 subplot(3,1,2)
29 a = gca();
30 a.x_location = "origin";
31 a.y_location = "origin";
32 a.data_bounds = [0,-2;20,2];
33 plot2d2(t,ct_polar,5)
34 xlabel('
    t')

```

```

35 title('Spreading code c(t)')
36 subplot(3,1,3)
37 a = gca();
38 a.x_location = "origin";
39 a.y_location = "origin";
40 a.data_bounds = [0,-2;20,2];
41 plot2d2(t,mt,5)
42 xlabel('

        t')
43 title('Product Signal m(t)')
44 //
45 figure
46 subplot(3,1,1)
47 a = gca();
48 a.x_location = "origin";
49 a.y_location = "origin";
50 a.data_bounds = [0,-2;20,2];
51 plot2d2(t,mt,5)
52 xlabel('

        t')
53 title('Product Signal m(t)')
54 subplot(3,1,2)
55 a = gca();
56 a.x_location = "origin";
57 a.y_location = "origin";
58 a.data_bounds = [0,-2;20,2];
59 plot(Carrier)
60 xlabel('

        t')
61 title('Carrier Signal')
62 subplot(3,1,3)
63 a = gca();
64 a.x_location = "origin";
65 a.y_location = "origin";
66 a.data_bounds = [0,-2;20,2];

```

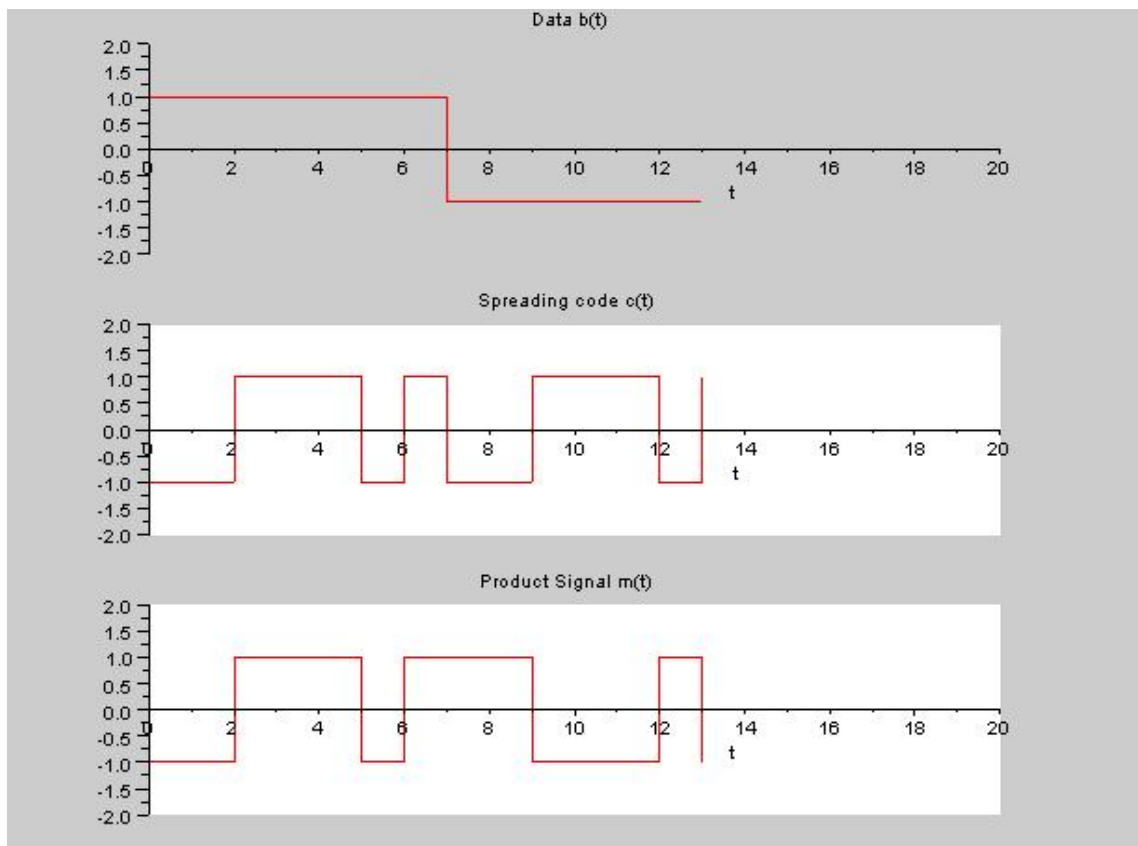


Figure 9.3: Figure9.6a

```

67 plot(st)
68 xlabel('
    t')
69 title('DS/BPSK signal s(t)')
70 //

```

---

#### Scilab code ARC 1 Alaw

```

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

```

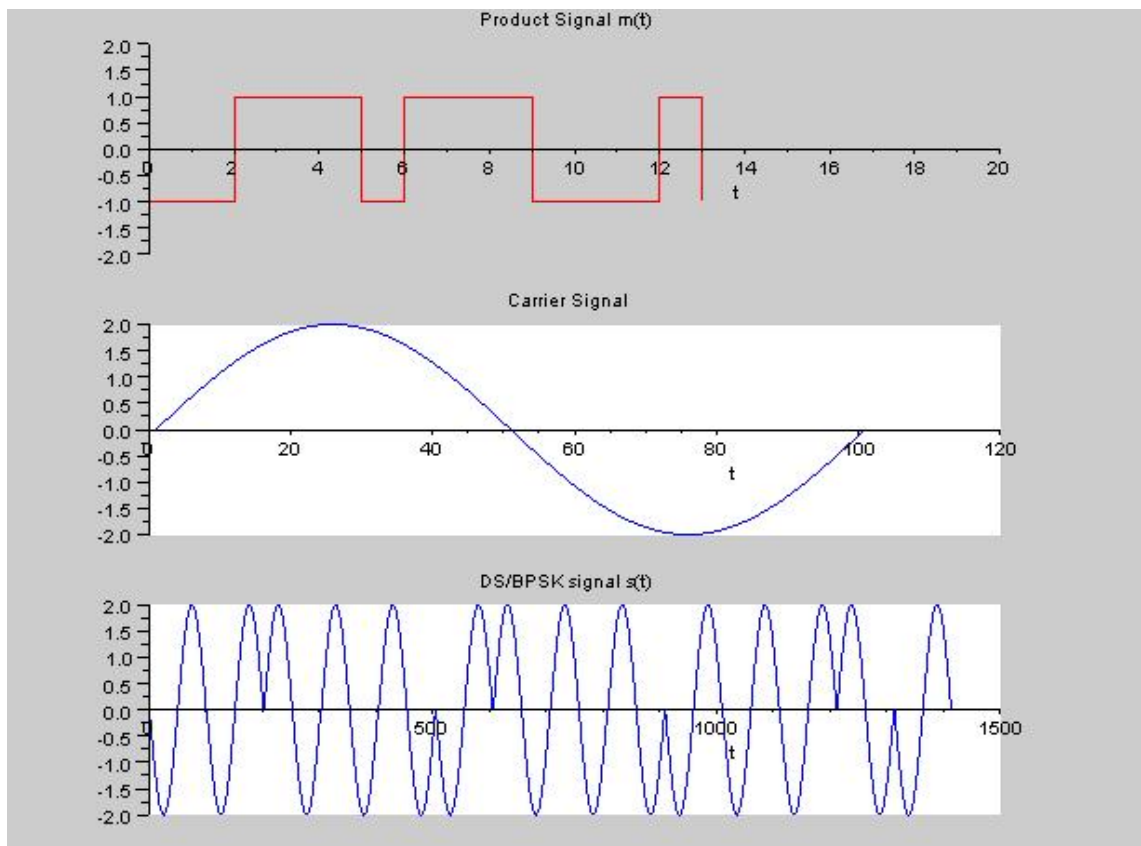


Figure 9.4: Figure9.6b

Figure9.6b



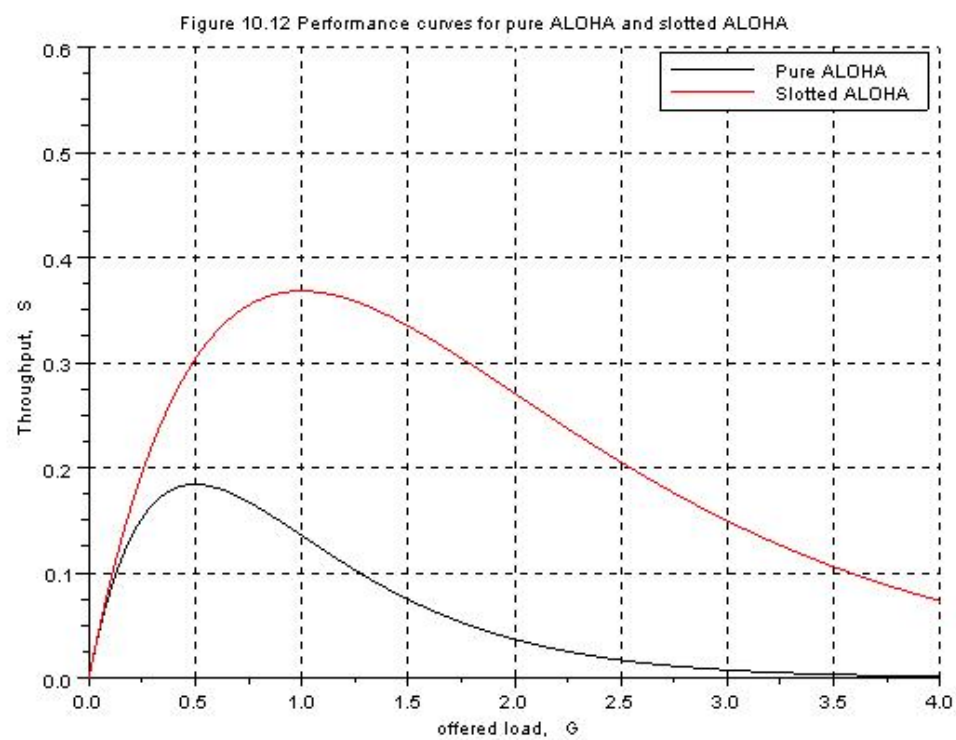


Figure 9.5: Figure10.12

```

4 //x = input vector
5 //Cx = A-law compressor output
6 //Xmax = maximum of input vector x
7 Xmax = max(abs(x));
8 for i = 1:length(x)
9     if(x(i)/Xmax <= 1/A)
10         Cx(i) = A*abs(x(i)/Xmax)./(1+log(A));
11     elseif(x(i)/Xmax > 1/A)
12         Cx(i) = (1+log(A*abs(x(i)/Xmax)))./(1+log(A));
13     end
14 end
15 Cx = Cx/Xmax; //normalization of output vector
16 Cx = Cx';
17 endfunction

```

---

**Scilab code ARC 2** auto correlation

```

1 function [Rxx] = auto_correlation(x)
2 //Autocorrelation of a given Input Sequence
3 //Finding out the period of the signal using
   autocorrelation technique
4
5 L = length(x);
6 h = zeros(1,L);
7 for i = 1:L
8     h(L-i+1) = x(i);
9 end
10 N = 2*L-1;
11 Rxx = zeros(1,N);
12 for i = L+1:N
13     h(i) = 0;
14 end
15 for i = L+1:N
16     x(i) = 0;
17 end
18 for n = 1:N
19     for k = 1:N
20         if(n >= k)

```

```

21         Rxx(n) = Rxx(n)+x(n-k+1)*h(k);
22     end
23 end
24 end
25 disp('Auto Correlation Result is ')
26 Rxx
27 disp('Center Value is the Maximum of autocorrelation
    result ')
28 [m,n] = max(Rxx)
29 disp('Period of the given signal using Auto
    Correlation Sequence ')
30 n
31 endfunction

```

---

### Scilab code ARC 3 Convolutional Coding

```

1 //Caption:Convolutional Code Generation
2 //Time Domain Approach
3 close;
4 clc;
5 g1 = input('Enter the input Top Adder Sequence:=')
6 g2 = input('Enter the input Bottom Adder Sequence:=')
7
8 m = input('Enter the message sequence:=')
9 x1 = round(convol(g1,m));
10 x2 = round(convol(g2,m));
11 x1 = modulo(x1,2);
12 x2 = modulo(x2,2);
13 N = length(x1);
14 for i =1:length(x1)
15     x(i,:) =[x1(N-i+1),x2(N-i+1)];
16 end
17 x = string(x)
18 //Result
19 //Enter the input Top Adder Sequence:=[1,1,1]
20 //Enter the input Bottom Adder Sequence:=[1,0,1]
21 //Enter the message sequence:=[1,1,0,0,1]
22 //x =

```

```

22 //!1 1 !
23 //!
24 //!1 0 !
25 //!
26 //!1 1 !
27 //!
28 //!1 1 !
29 //!
30 //!0 1 !
31 //!
32 //!0 1 !
33 //!
34 //!1 1 !

```

---

#### Scilab code ARC 4 Hamming Distance

```

1 //Caption: Hamming Weight and Hamming Distance
2 //H(7,4)
3 //Code Word Length = 7, Message Word length = 4,
  Parity bits =3
4 //clear;
5 close;
6 clc;
7 //Getting Code Words
8 code1 = input('Enter the first code word');
9 code2 = input('Enter the second code word');
10 Hamming_Distance = 0;
11 for i = 1:length(code1)
12     Hamming_Distance = Hamming_Distance + xor(code1(i),
        code2(i));
13 end
14 disp(Hamming_Distance, 'Hamming Distance')
15 //Result
16 //Enter the first code word [0,1,1,1,0,0,1]
17 //Enter the second code word [1,1,0,0,1,0,1]
18 //Hamming Distance 4.

```

---

#### Scilab code ARC 5 Hamming Encode

```

1 //Caption:Hamming Encoding
2 //H(7,4)
3 //Code Word Length = 7, Message Word length = 4,
   Parity bits =3
4 //clear;
5 close;
6 clc;
7 //Getting Message Word
8 m3 = input('Enter the 1 bit(MSb) of message word');
9 m2 = input('Enter the 2 bit of message word');
10 m1 = input('Enter the 3 bit of message word');
11 m0 = input('Enter the 4 bit(LSb) of message word');
12 //Generating Parity bits
13 for i = 1:(2^4)
14     b2(i) = xor(m0(i),xor(m3(i),m1(i)));
15     b1(i) = xor(m1(i),xor(m2(i),m3(i)));
16     b0(i) = xor(m0(i),xor(m1(i),m2(i)));
17     m(i,:) = [m3(i) m2(i) m1(i) m0(i)];
18     b(i,:) = [b2(i) b1(i) b0(i)];
19 end
20 C = [b m];
21 disp('
   -----
   ')
22 for i = 1:2^4
23     disp(i)
24     disp(m(i,:), 'Message Word')
25     disp(b(i,:), 'Parity Bits ')
26     disp(C(i,:), 'CodeWord')
27     disp("    ");
28     disp("    ");
29 end
30 disp('
   -----
   ')
31 //disp(m b C)
32 //Result
33 //Enter the 1 bit(MSb) of message word

```

```

    [0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1];
34 //Enter the 2 bit of message word
    [0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1];
35 //Enter the 3 bit of message word
    [0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1];
36 //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

```

1 function x = invmulaw(y,mu)
2 //Non-linear Quantization
3 //invmulaw: inverse mulaw nonlinear quantization
4 //x = output vector
5 //y = input vector (using mulaw nonlinear
  compression)
6 x = (((1+mu).^(abs(y))-1)./mu)
7 endfunction

```

---

#### Scilab code ARC 6 PCM Encoding

```

1 function [c] = PCM_Encoding(x,L,en_code)
2 //Encoding: Converting Quantized decimal sample
  values in to binary
3 //x = input sequence
4 //L = number of quantization levels
5 //en_code = normalized input sequence
6 n = log2(L);
7 c = zeros(length(x),n);
8 for i = 1:length(x)
9   for j = n:-1:0
10     if(fix(en_code(i)/(2^j))==1)
11       c(i,(n-j)) =1;
12       en_code(i) = en_code(i)-2^j;
13     end
14   end
15 end
16 disp(c)

```

---

### Scilab code ARC 7 PCM Transmission

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

---

### Scilab code ARC 8 sinc new

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

---

### Scilab code ARC 9 uniform pcm

```
1 function [SQNR,xq,en_code] = uniform_pcm(x,L)
2     //x = input sequence
3     //L = number of quantization levels
4     xmax = max(abs(x));
5     xq = x/xmax;
6     en_code = xq;
7     d = 2/L;
8     q = d*[0:L-1];
9     q = q-((L-1)/2)*d;
10    for i = 1:L
11        xq(find(((q(i)-d/2)<= xq)&(xq<=(q(i)+d/2))))=...
12        q(i).*ones(1,length(find(((q(i)-d/2)<=xq)&(xq<=(
13            q(i)+d/2))))));
14        en_code(find(xq == q(i)))= (i-1).*ones(1,length(
15            find(xq == q(i))));
16    end
17    xq = xq*xmax;
18    SQNR = 20*log10(norm(x)/norm(x-xq));
19 endfunction
```

---

### Scilab code ARC 10 xor

```
1 function [value] = xor(A,B)
2     if(A==B)
3         value = 0;
4     else
5         value = 1;
6     end
7 endfunction
```

---