

Scilab Textbook Companion for
Digital Communication
by S. Haykin¹

Created by
Prof. R. Senthilkumar
B.Tech. + M.Tech
Electronics Engineering
Institute of Road and Transport Technology
College Teacher
Na

Cross-Checked by
Prof. Saravanan Vijayakumaran

October 15, 2013

¹Funded by a grant from the National Mission on Education through ICT, <http://spoken-tutorial.org/NMEICT-Intro>. This Textbook Companion and Scilab codes written in it can be downloaded from the "Textbook Companion Project" section at the website <http://scilab.in>

Book Description

Title: Digital Communication

Author: S. Haykin

Publisher: Wiley India, New Delhi

Edition: 1

Year: 2010

ISBN: 9788126508242

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

Exa Example (Solved example)

Eqn Equation (Particular equation of the above book)

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

For example, 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	6
2 Fundamental Limit on Performance	8
3 Detection and Estimation	15
4 Sampling Process	23
5 Waveform Coding Techniques	25
6 Baseband Shaping for Data Transmission	31
7 Digital Modulation Techniques	44
8 Error Control Coding	63
9 Spread Spectrum Modulation	71

List of Scilab Codes

Exa 1.2	clear	6
Exa 2.1	clear	8
Exa 2.2	clear	9
Exa 2.3	clear	10
Exa 2.4	clear	11
Exa 2.5	clear	12
Exa 2.6	clear	12
Exa 2.7	clear	13
Exa 3.1	clear	15
Exa 3.2	clear	17
Exa 3.3	clear	18
Exa 3.4	clear	19
Exa 3.6	clear	20
Exa 3.29	clear	21
Exa 4.1	clear	23
Exa 4.3	clear	24
Exa 5.1	clear	25
Exa 5.2	clear	26
Exa 5.3	clear	27
Exa 5.5	clear	28
Exa 05.13	clear	29
Exa 5.13	clear	30
Exa 06.01	clear	31
Exa 06.1	clear	32
Exa 6.01	clear	33
Exa 6.1	clear	34
Exa 6.2	clear	35
Exa 6.3	clear	36

Exa 6.4	clear	37
Exa 6.6	clear	39
Exa 6.7	clear	40
Exa 6.9	clear	41
Exa 6.15	clear	42
Exa 7.01	clear	44
Exa 7.1	clear	46
Exa 7.02	clear	48
Exa 7.2	clear	49
Exa 7.3	clear	50
Exa 7.4	clear	52
Exa 7.4.7.20	clear	53
Exa 7.06	clear	54
Exa 7.6	clear	55
Exa 7.7	clear	56
Exa 7.12.7.2	clear	57
Exa 7.29	clear	58
Exa 7.30	clear	59
Exa 7.31	clear	60
Exa 7.41	clear	61
Exa 8.1	clear	63
Exa 8.2	clear	63
Exa 8.3	clear	64
Exa 8.4	clear	65
Exa 8.5	clear	66
Exa 8.6	clear	67
Exa 8.7	clear	67
Exa 8.8	clear	69
Exa 8.11	clear	69
Exa 9.1	clear	71
Exa 9.2	clear	72
Exa 9.3	clear	74
Exa 9.4.9.5	clear	75
Exa 9.4.9.6	clear	76

Chapter 1

Introduction

Scilab code Exa 1.2 clear

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

```
    Time')
22 ylabel('

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

Chapter 2

Fundamental Limit on Performance

Scilab code Exa 2.1 clear

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

Scilab code Exa 2.2 clear

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

```

24   H_Ruo_Square = H_Ruo_Square+P_sigma(i)*log2(1/
      P_sigma(i));
25   end
26   disp('bits ', H_Ruo_Square, 'H(Ruo_Square)=')
27   disp('H(Ruo_Square) = 2*H(Ruo)')

```

Scilab code Exa 2.3 clear

```

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

```

```

23 sigma_1 = P0*(L0-L)^2+P1*(L1-L)^2+P2*(L2-L)^2+P3*(L3
    -L)^2+P4*(L4-L)^2;
24 disp(sigma_1,'Varinace of Huffman code')

```

Scilab code Exa 2.4 clear

```

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

```

Scilab code Exa 2.5 clear

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

```
1 //clear//
2 //Caption:Channel Capacity of a Binary Symmetric
   Channel
3 //Example2.6:Channel Capacity of Binary Symmetri
   Channel
4 clear;
5 close;
6 clc;
7 p = 0:0.01:0.5;
8 for i =1:length(p)
9     if(i~=1)
10        C(i) = 1+p(i)*log2(p(i))+(1-p(i))*log2((1-p(i)))
           ;
```

```

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 clear

```

1 //clear//
2 //Caption:Significance of the Channel Coding theorem
3 //Example2.7: Significance of the channel coding
  theorem
4 //Average Probability of Error of Repetition Code
5 clear;
6 clc;
7 close;
8 p =10^-2;
9 pe_1 =p; //Average Probability of error for code rate
  r = 1
10 pe_3 = 3*p^2*(1-p)+p^3;//probability of error for code
  rate r =1/3
11 pe_5 = 10*p^3*(1-p)^2+5*p^4*(1-p)+p^5;//error for
  code rate r =1/5
12 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
13 r = [1,1/3,1/5,1/7];
14 pe = [pe_1,pe_3,pe_5,pe_7];
15 a=gca();

```

```

16 a.data_bounds=[0,0;1,0.01];
17 plot2d(r,pe,5)
18 xlabel('Code rate , r')
19 ylabel('Average Probability of error , Pe')
20 title('Figure 2.12 Illustrating significance of the
      channel coding theorem')
21 legend('Repetition codes')
22 xgrid(1)
23 disp('Table 2.3 Average Probability of Error for
      Repetition Code')
24 disp('
      -----
      ')
25 disp(r,'Code Rate, r =1/n',pe,'Average Probability of
      Error , Pe')
26 disp('
      -----
      ')

```

Chapter 3

Detection and Estimation

Scilab code Exa 3.1 clear

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



```

21 t7 = 2*T/3:0.01:T;
22 phi3t = sqrt(3/T)*[0, ones(1, length(t7)-2), 0];
23 //
24 figure
25 title('Figure3.4(a) Set of signals to be
        orthonormalized')
26 subplot(4,1,1)
27 a = gca();
28 a.data_bounds = [0,0;2,2];
29 plot2d2(t1, s1t, 5)
30 xlabel('t')
31 ylabel('s1(t)')
32 subplot(4,1,2)
33 a = gca();
34 a.data_bounds = [0,0;2,2];
35 plot2d2(t2, s2t, 5)
36 xlabel('t')
37 ylabel('s2(t)')
38 subplot(4,1,3)
39 a = gca();
40 a.data_bounds = [0,0;2,2];
41 plot2d2(t3, s3t, 5)
42 xlabel('t')
43 ylabel('s3(t)')
44 subplot(4,1,4)
45 a = gca();
46 a.data_bounds = [0,0;2,2];
47 plot2d2(t4, s4t, 5)
48 xlabel('t')
49 ylabel('s4(t)')
50 //
51 figure
52 title('Figure3.4(b) The resulting set of orthonormal
        functions')
53 subplot(3,1,1)
54 a = gca();
55 a.data_bounds = [0,0;2,4];
56 plot2d2(t5, phi1t, 5)

```

```

57 xlabel('t')
58 ylabel('phi1(t)')
59 subplot(3,1,2)
60 a =gca();
61 a.data_bounds = [0,0;2,4];
62 plot2d2(t6,phi2t,5)
63 xlabel('t')
64 ylabel('phi2(t)')
65 subplot(3,1,3)
66 a =gca();
67 a.data_bounds = [0,0;2,4];
68 plot2d2(t7,phi3t,5)
69 xlabel('t')
70 ylabel('phi3(t)')

```

Scilab code Exa 3.2 clear

```

1 //clear//
2 //Caption:M-ARY Signaling
3 //Example3.2:M-ARY SIGNALING
4 //Signal constellation and Representation of dibits
5 clear;
6 close;
7 clc;
8 a =1; //amplitude =1
9 T =1; //Symbol duration in seconds
10 //Four message points
11 Si1 = [(-3/2)*a*sqrt(T),(-1/2)*a*sqrt(T),(3/2)*a*
        sqrt(T),(1/2)*a*sqrt(T)];
12 a =gca();
13 a.data_bounds = [-2,-0.5;2,0.5]
14 plot2d(Si1,[0,0,0,0],-10)
15 xlabel('phi1(t)')
16 title('Figure 3.8 (a) Signal constellation')
17 xgrid(1)

```

```

18 disp('Figure 3.8 (b). Representation of transmitted
      dibits ')
19 disp('Loc. of meg. point |  $(-3/2)\sqrt{T}$  |  $(-1/2)\sqrt{T}$  |
       $(3/2)\sqrt{T}$  |  $(1/2)\sqrt{T}$  ')
20 disp('
      -----
21 disp('Transmitted dibit |          00          |          01
      |          11          |          10 ')
22 disp('')
23 disp('')
24 disp('Figure 3.8 (c). Decision intervals for
      received dibits ')
25 disp('Received dibit      |          00          |          01
      |          11          |          10 ')
26 disp('
      -----
27 disp('Interval on  $\phi_1(t)$  |  $x_1 < -a\sqrt{T}$  |  $-a\sqrt{T}$ 
       $T < x_1 < 0$  |  $0 < x_1 < a\sqrt{T}$  |  $a\sqrt{T} < x_1$  ')

```

Scilab code Exa 3.3 clear

```

1 //clear//
2 //Caption: Matched Filter output for RF pulse
3 //Example 3.3: MATCHED FILTER FOR RF PULSE
4 clear;
5 close;
6 clc;
7 fc =4; //carrier frequency in Hz
8 T =1;
9 t1 = 0:0.01:T;
10 phit = sqrt(2/T)*cos(2*pi*fc*t1);
11 hopt = phit;
12 phiot = convol(phit,hopt);

```

```

13 phiot = phiot/max(phiot);
14 t2 = 0:0.01:2*T;
15 subplot(2,1,1)
16 a =gca();
17 a.x_location = "origin";
18 a.y_location = "origin";
19 a.data_bounds = [0,-1;1,1];
20 plot2d(t1,phit);
21 xlabel('
    t')
22 ylabel('
    phi(t)')
23 title('Figure 3.13 (a) RF pulse input')
24 subplot(2,1,2)
25 a =gca();
26 a.x_location = "origin";
27 a.y_location = "origin";
28 a.data_bounds = [0,-1;1,1];
29 plot2d(t2,phiot);
30 xlabel('
    t')
31 ylabel('
    phi0(t)')
32 title('Figure 3.13 (b) Matched Filter output')

```

Scilab code Exa 3.4 clear

```

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

```

```

    input
4  clear;
5  close;
6  clc;
7  phit =0.1*rand(1,10,'uniform');
8  hopt = phit;
9  phi0t = convol(phit,hopt);
10 phi0t = phi0t/max(phi0t);
11 subplot(2,1,1)
12 a =gca();
13 a.x_location = "origin";
14 a.y_location = "origin";
15 a.data_bounds = [0,-1;1,1];
16 plot2d([1:length(phit)],phit);
17 xlabel('
    t')
18 ylabel('
    phi(t)')
19 title('Figure 3.16 (a) Noise Like input signal')
20 subplot(2,1,2)
21 a =gca();
22 a.x_location = "origin";
23 a.y_location = "origin";
24 a.data_bounds = [0,-1;1,1];
25 plot2d([1:length(phi0t)],phi0t);
26 xlabel('
    t')
27 ylabel('
    phi0(t)')
28 title('Figure 3.16 (b) Matched Filter output')

```

Scilab code Exa 3.6 clear

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

Scilab code Exa 3.29 clear

```
1 //clear//
2 //Implementation of LMS ADAPTIVE FILTER
3 //For noise cancellation application
4 clear;
5 clc;
6 close;
7 order = 18;
8 t =0:0.01:1;
9 x = sin(2*%pi*5*t);
10 noise =rand(1,length(x));
11 x_n = x+noise;
12 ref_noise = noise*rand(10);
```

```

13 w = zeros(order,1);
14 mu = 0.01*(sum(x.^2)/length(x));
15 N = length(x);
16 for k =1:1010
17     for i = 1:N-order-1
18         buffer = ref_noise(i:i+order-1);
19         desired(i) = x_n(i)-buffer*w;
20         w = w+(buffer*mu*desired(i))';
21     end
22 end
23 subplot(4,1,1)
24 plot2d(t,x)
25 title('Original Input Signal')
26 subplot(4,1,2)
27 plot2d(t,noise,2)
28 title('random noise')
29 subplot(4,1,3)
30 plot2d(t,x_n,5)
31 title('Signal+noise')
32 subplot(4,1,4)
33 plot(desired)
34 title('noise removed signal')

```

Chapter 4

Sampling Process

Scilab code Exa 4.1 clear

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



```

19 ylabel('                                g(t)')
20 title('Figure 4.8 (a) Sinc pulse g(t)')
21 subplot(2,1,2)
22 a=gca();
23 a.data_bounds =[-2,0;2,2];
24 a.x_location = "origin"
25 a.y_location = "origin"
26 plot2d(f1,G)
27 xlabel('                                f')
28 ylabel('                                G(f)')
29 title('Figure 4.8 (b) Amplitude spectrum |G(f)|')

```

Scilab code Exa 4.3 clear

```

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

```

Chapter 5

Waveform Coding Techniques

Scilab code Exa 5.1 clear

```
1 //clear//
2 //Caption: Average Transmitted Power for PCM
3 //Example 5.1: Average Transmitted Power of PCM
4 //Page 187
5 clear;
6 clc;
7 sigma_N = input('Enter the noise variance');
8 k = input('Enter the separation constant for on-off
    signaling');
9 M = input('Enter the number of discrete amplitude
    levels for NRZ polar');
10 disp('The average transmitted power is:')
11 P = (k^2)*(sigma_N)*((M^2)-1)/12;
12 disp(P)
13 //Result
14 //Enter the noise variance 10^-6
15 //Enter the separation constant for on-off signaling
    7
16 //Enter the number of discrete amplitude levels for
    NRZ polar 2
17 // The average transmitted power is: 0.0000122
```

Scilab code Exa 5.2 clear

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

Scilab code Exa 5.3 clear

```
1 //clear//
2 //Caption:Signal-to-Quantization Noise Ratio of PCM
3 //Example5.3:Signal-to-Quantization noise ratio
4 //Channel Bandwidth B
5 clear;
6 clc;
7 n = input('Enter no. of bits used to encode:');
8 W = input('Enter the message signal bandwidth in Hz:');
9 B = n*W;
10 disp(B,'Channel width in Hz:');
11 SNRo = 6*n - 7.2;
12 disp(SNRo,'Output Signal to noise ratio in dB:');
13 //Result 1 if n = 8 bits
14 //Enter no. of bits used to encode: 8
15 //Enter the message signal bandwidth in Hz: 4000
16 //Channel width in Hz: 32000.
17 //Output Signal to noise ratio in dB: 40.8
18 ///////////////////////////////////////////////////
19 //Result 2 if n = 9 bits
20 //Enter no. of bits used to encode:9
21 //Enter the message signal bandwidth in Hz:4000
22 //Channel width in Hz: 36000.
23 //Output Signal to noise ratio in dB: 46.8
24 ///////////////////////////////////////////////////
25 //Conclusion: comparing result 1 with result 2 if
    number of bits increased by 1
26 //corresponding output signal to noise in PCM
    increased by 6 dB.
```

Scilab code Exa 5.5 clear

```
1 //clear//
2 //Example 5:Delta Modulation – to avoid slope
  overload distortion
3 //maximum output signal-to-noise ratio for
  sinusoidal modulation
4 //page 207
5 clear;
6 clc;
7 a0 = input('Enter the amplitude of sinusoidal signal
  :');
8 f0 = input('Enter the frequency of sinusoidal signal
  in Hz:');
9 fs = input('Enter the sampling frequency in samples
  per seconds:');
10 Ts = 1/fs;//Sampling interval
11 delta = 2*pi*f0*a0*Ts;//Step size to avoid slope
  overload
12 Pmax = (a0^2)/2;//maximum permissible output power
13 sigma_Q = (delta^2)/3;//Quantization error or noise
  variance
14 W = f0;//Maximum message bandwidth
15 N = W*Ts*sigma_Q; //Average output noise power
16 SNRo = Pmax/N; // Maximum output signal-to-noise
  ratio
17 SNRo_dB = 10*log10(SNRo);
18 disp(SNRo_dB,'Maximum output signal-to-noise in dB
  for Delta Modulation:')
19 //Result 1 for fs = 8000 Hertz
20 //Enter the amplitude of sinusoidal signal:1
21 //Enter the frequency of sinusoidal signal in Hz
  :4000
22 //Enter the sampling frequency in samples per
  seconds:8000
23 //Maximum output signal-to-noise in dB for Delta
  Modulation:-5.1717849
24 //
```

```

////////////////////////////////////
25 //Result 2 for fs = 16000 Hertz
26 //Enter the amplitude of sinusoidal signal:1
27 //Enter the frequency of sinusoidal signal in Hz
   :4000
28 //Enter the sampling frequency in samples per
   seconds:16000
29 //Maximum output signal-to-noise in dB for Delta
   Modulation:3.859115
30 //
   //////////////////////////////////////
31 //Conclusion: comparing result 1 with result 2, if
   the sampling frequency
32 //is doubled the signal to noise increased by 9 dB

```

Scilab code Exa 05.13 clear

```

1 //clear//
2 //Caption:A-law companding
3 //Figure5.13(b)A-law companding, Nonlinear
   Quantization
4 //Plotting A-law characteristics for different
5 //Values of A
6 clc;
7 x = 0:0.01:1; //Normalized input
8 A = [1,2,87.56]; //different values of A
9 for i = 1:length(A)
10   [Cx(i,:),Xmax(i)] = Alaw(x,A(i));
11 end
12 plot2d(x/Xmax(1),Cx(1,:),2)
13 plot2d(x/Xmax(2),Cx(2,:),4)
14 plot2d(x/Xmax(3),Cx(3,:),6)
15 xtitle('Compression Law: A-Law companding',')

```

```

        Normalized Input |x| ', 'Normalized Output |c(x)| ')
    ;
16 legend(['A =1'], ['A=2'], ['A=87.56'])

```

Scilab code Exa 5.13 clear

```

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

```

Chapter 6

Baseband Shaping for Data Transmission

Scilab code Exa 06.01 clear

```
1 //clear//
2 //Caption:Nonreturn-to-zero bipolar format
3 //Figure 6.1(c):Discrete PAM Signals Generation
4 // [3]. BiPolar NRZ
5 //page 235
6 clear;
7 close;
8 clc;
9 x = [0 1 1 0 0 1 0 0 1 1];
10 binary_negative = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1];
11 binary_zero = [0 0 0 0 0 0 0 0 0 0];
12 binary_positive = [1 1 1 1 1 1 1 1 1 1];
13 L = length(x);
14 L1 = length(binary_negative);
15 total_duration = L*L1;
16 //plotting
17 a =gca();
18 a.data_bounds =[0 -2;L*L1 2];
19 for i =1:L
```



```

20  if(x(i)==0)
21      plot([i*L-L+1:i*L],binary_zero);
22      poly1= a.children(1).children(1);
23      poly1.thickness =3;
24  elseif((x(i)==1)&(x(i-1)~=1))
25      plot([i*L-L+1:i*L],binary_positive);
26      poly1= a.children(1).children(1);
27      poly1.thickness =3;
28  else
29      plot([i*L-L+1:i*L],binary_negative);
30      poly1= a.children(1).children(1);
31      poly1.thickness =3;
32  end
33 end
34 xgrid(1)
35 title('BiPolar NRZ')

```

Scilab code Exa 06.1 clear

```

1  //clear//
2  //Caption:Nonreturn-to-zero polar format
3  //Figure 6.1(b): Discrete PAM Signals Generation
4  // [2].Polar NRZ
5  //page 235
6  clear;
7  close;
8  clc;
9  x = [0 1 0 0 0 1 0 0 1 1];
10 binary_negative = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1];
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_negative);
21         poly1= a.children(1).children(1);
22         poly1.thickness =3;
23     else
24         plot([i*L-L+1:i*L],binary_positive);
25         poly1= a.children(1).children(1);
26         poly1.thickness =3;
27     end
28 end
29 xgrid(1)
30 title('Polar NRZ')

```

Scilab code Exa 6.01 clear

```

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

```

Scilab code Exa 6.1 clear

```

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

```

```

:0.647*10-6
16 //Transmission bandwidth for raised cosine spectrum
    in Hz:1545595.1

```

Scilab code Exa 6.2 clear

```

1 //clear//
2 //Caption:Duobinary Encoding
3 //Example6.2: Precoded Duobinary coder and decoder
4 //Page 256
5 clc;
6 b = [0,0,1,0,1,1,0]; //input binary sequence:precoder
    input
7 a(1) = xor(1,b(1));
8 if(a(1)==1)
9     a_volts(1) = 1;
10 end
11 for k =2:length(b)
12     a(k) = xor(a(k-1),b(k));
13     if(a(k)==1)
14         a_volts(k)=1;
15     else
16         a_volts(k)=-1;
17     end
18 end
19 a = a';
20 a_volts = a_volts';
21 disp(a,'Precoder output in binary form:')
22 disp(a_volts,'Precoder output in volts:')
23 //Duobinary coder output in volts
24 c(1) = 1+ a_volts(1);
25 for k =2:length(a)
26     c(k) = a_volts(k-1)+a_volts(k);
27 end
28 c = c';

```

```

29 disp(c, 'Duobinary coder output in volts:')
30 //Duobinary decoder output by applying decision
    rule
31 for k =1:length(c)
32     if(abs(c(k))>1)
33         b_r(k) = 0;
34     else
35         b_r(k) = 1;
36     end
37 end
38 b_r = b_r';
39 disp(b_r, 'Recovered original sequence at detector
    oupupt:')
40 //Result
41 //Precoder output in binary form:
42 //
43 //  1.    1.    0.    0.    1.    0.    0.
44 //
45 // Precoder output in volts:
46 //
47 //  1.    1.  - 1.  - 1.    1.  - 1.  - 1.
48 //
49 // Duobinary coder output in volts:
50 //
51 //  2.    2.    0.  - 2.    0.    0.  - 2.
52 //
53 // Recovered original sequence at detector oupupt:
54 //
55 //  0.    0.    1.    0.    1.    1.    0.

```

Scilab code Exa 6.3 clear

```

1 //clear//
2 //Caption:Generation of bipolar output for duobinary
    coder

```

```

3 //Example6.3: Operation of Circuit in figure 6.13
4 //for generating bipolar format
5 //page 256 and page 257
6 //Refer Table 6.4
7 clc;
8 x = [0,1,1,0,1,0,0,0,1,1]; //input binary sequence:
   precoder input
9 y(1) = 1;
10 for k =2:length(x)+1
11   y(k) = xor(x(k-1),y(k-1));
12 end
13 y_delay = y(1:$-1);
14 y = y';
15 y_delay = y_delay';
16 disp(y, 'Modulo-2 adder output:')
17 disp(y_delay, 'Delay element output:')
18 for k = 1:length(y_delay)
19   z(k) = y(k+1)-y_delay(k);
20 end
21 z = z';
22 disp(z, 'differential encoder bipolar output in volts
   :')
23 //Result
24 //Modulo-2 adder output:
25 //   1.   1.   0.   1.   1.   0.   0.   0.
   0.   1.   0.
26 // Delay element output:
27 //   1.   1.   0.   1.   1.   0.   0.   0.
   0.   1.
28 // differential encoder bipolar output in volts:
29 //   0. - 1.   1.   0. - 1.   0.   0.   0.
   1. - 1.

```

Scilab code Exa 6.4 clear

```

1 //clear//
2 //Caption:Power Spectra of different binary data
   formats
3 //Figure 6.4: Power Spectal Densities of
4 //Different Line Coding Techniques
5 //[1].NRZ Polar Format [2].NRZ Bipolar format
6 //[3].NRZ Unipolar format [4]. Manchester format
7 //Page 241
8 close;
9 clc;
10 //[1]. NRZ Polar format
11 a = input('Enter the Amplitude value:');
12 fb = input('Enter the bit rate:');
13 Tb = 1/fb; //bit duration
14 f = 0:1/(100*Tb):2/Tb;
15 for i = 1:length(f)
16     Sxxf_NRZ_P(i) = (a^2)*Tb*(sinc_new(f(i)*Tb)^2);
17     Sxxf_NRZ_BP(i) = (a^2)*Tb*((sinc_new(f(i)*Tb))^2)
        *((sin(%pi*f(i)*Tb))^2);
18     if (i==1)
19         Sxxf_NRZ_UP(i) = (a^2)*(Tb/4)*((sinc_new(f(i)*Tb
        ))^2)+(a^2)/4;
20     else
21         Sxxf_NRZ_UP(i) = (a^2)*(Tb/4)*((sinc_new(f(i)*Tb
        ))^2);
22     end
23     Sxxf_Manch(i) = (a^2)*Tb*(sinc_new(f(i)*Tb/2)^2)*
        (sin(%pi*f(i)*Tb/2)^2);
24 end
25 //Plotting
26 a = gca();
27 plot2d(f, Sxxf_NRZ_P)
28 poly1= a.children(1).children(1);
29 poly1.thickness = 2; // the tickness of a curve.
30 plot2d(f, Sxxf_NRZ_BP, 2)
31 poly1= a.children(1).children(1);
32 poly1.thickness = 2; // the tickness of a curve.
33 plot2d(f, Sxxf_NRZ_UP, 5)

```

```

34 poly1= a.children(1).children(1);
35 poly1.thickness = 2; // the tickness of a curve.
36 plot2d(f,Sxxf_Manch,9)
37 poly1= a.children(1).children(1);
38 poly1.thickness = 2; // the tickness of a curve.
39 xlabel('f*Tb————>')
40 ylabel('Sxx(f)————>')
41 title('Power Spectral Densities of Different Line
      Codinig Techniques')
42 xgrid(1)
43 legend(['NRZ Polar Format', 'NRZ Bipolar format', 'NRZ
      Unipolar format', 'Manchester format']);
44 //Result
45 //Enter the Amplitude value:1
46 //Enter the bit rate:1

```

Scilab code Exa 6.6 clear

```

1 //clear//
2 //Caption:Ideal solution for zero ISI
3 //Figure 6.6(b): Ideal Solution for Intersymbol
  Interference
4 //SINC pulse
5 //page 249
6 rb = input('Enter the bit rate:');
7 Bo = rb/2;
8 t = -3:1/100:3;
9 x = sinc_new(2*Bo*t);
10 plot(t,x)
11 xlabel('t————>');
12 ylabel('p(t)————>');
13 title('SINC Pulse for zero ISI')
14 xgrid(1)
15 //Result
16 //Enter the bit rate:1

```

Scilab code Exa 6.7 clear

```
1 //clear//
2 //Caption:Practical solution: Raised Cosine
3 //Figure6.7(b):Practical Solution for Intersymbol
  Interference
4 //Raised Cosine Spectrum
5 //page 250
6 close;
7 clc;
8 rb = input('Enter the bit rate:');
9 Tb =1/rb;
10 t =-3:1/100:3;
11 Bo = rb/2;
12 Alpha =0;      //Intialized to zero
13 x =t/Tb;
14 for j =1:3
15     for i =1:length(t)
16         if((j==3)&((t(i)==0.5)|(t(i)==-0.5)))
17             p(j,i) = sinc_new(2*Bo*t(i));
18         else
19             num = sinc_new(2*Bo*t(i))*cos(2*%pi*Alpha*
                Bo*t(i));
20             den = 1-16*(Alpha^2)*(Bo^2)*(t(i)^2)+0.01;
21             p(j,i)= num/den;
22         end
23     end
24     Alpha = Alpha+0.5;
25 end
26 a =gca();
27 plot2d(t,p(1,:))
28 plot2d(t,p(2,:))
29 poly1= a.children(1).children(1);
30 poly1.foreground=2;
```

```

31 plot2d(t,p(3,:))
32 poly2= a.children(1).children(1);
33 poly2.foreground=4;
34 poly2.line_style = 3;
35 xlabel('t/Tb————>');
36 ylabel('p(t)————>');
37 title('RAISED COSINE SPECTRUM – Practical Solution
      for ISI')
38 legend(['ROlloff Factor =0', 'ROlloff Factor =0.5', '
      ROlloff Factor =1'])
39 xgrid(1)
40 //Result
41 //Enter the bit rate:1

```

Scilab code Exa 6.9 clear

```

1 //clear//
2 //Caption:Frequency response of duobinary conversion
  filter
3 //Figure6.9:Frequency Response of Duobinary
  Conversion filter
4 //(a)Amplitude Response
5 //(b)Phase Response
6 //Page 254
7 clear;
8 close;
9 clc;
10 rb = input('Enter the bit rate=');
11 Tb =1/rb; //Bit duration
12 f = -rb/2:1/100:rb/2;
13 Amplitude_Response = abs(2*cos(%pi*f.*Tb));
14 Phase_Response = -(%pi*f.*Tb);
15 subplot(2,1,1)
16 a=gca();
17 a.x_location ="origin";

```

```

18 a.y_location = "origin";
19 plot2d(f, Amplitude_Response, 2)
20 poly1= a.children(1).children(1);
21 poly1.thickness = 2; // the tickness of a curve.
22 xlabel('Frequency f——>')
23 ylabel('|H(f)| ——>')
24 title('Amplitude Reponse of Duobinary Singaling')
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 Reponse of Duobinary Singaling')
35 //Result
36 //Enter the bit rate=8

```

Scilab code Exa 6.15 clear

```

1 //clear//
2 //Caption:Frequency response of modified duobinary
    conversion filter
3 //Figure 6.15: Frequency Response of Modified
    duobinary conversion filter
4 //(a)Amplitude Response
5 //(b)Phase Response
6 //page 259
7 clear;
8 close;
9 clc;

```

```

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

```

Chapter 7

Digital Modulation Techniques

Scilab code Exa 7.01 clear

```
1 //clear//
2 //Caption: Waveforms of Different Digital Modulation
   techniques
3 //Figure7.1
4 //Digital Modulation Techniques
5 //To Plot the ASK, FSK and PSk Waveforms
6 clear;
7 clc;
8 close;
9 f = input('Enter the Analog Carrier Frequency in Hz'
   );
10 t = 0:1/512:1;
11 x = sin(2*pi*f*t);
12 I = input('Enter the digital binary data');
13 //Generation of ASK Waveform
14 Xask = [];
15 for n = 1:length(I)
16     if((I(n)==1)&(n==1))
17         Xask = [x,Xask];
18     elseif((I(n)==0)&(n==1))
19         Xask = [zeros(1,length(x)),Xask];
```

```

20     elseif((I(n)==1)&(n~=1))
21         Xask = [Xask,x];
22     elseif((I(n)==0)&(n~=1))
23         Xask = [Xask,zeros(1,length(x))];
24     end
25 end
26 // Generation of FSK Waveform
27 Xfsk = [];
28 x1 = sin(2*%pi*f*t);
29 x2 = sin(2*%pi*(2*f)*t);
30 for n = 1:length(I)
31     if (I(n)==1)
32         Xfsk = [Xfsk,x2];
33     elseif (I(n)~=1)
34         Xfsk = [Xfsk,x1];
35     end
36 end
37 // Generation of PSK Waveform
38 Xpsk = [];
39 x1 = sin(2*%pi*f*t);
40 x2 = -sin(2*%pi*f*t);
41 for n = 1:length(I)
42     if (I(n)==1)
43         Xpsk = [Xpsk,x1];
44     elseif (I(n)~=1)
45         Xpsk = [Xpsk,x2];
46     end
47 end
48 figure
49 plot(t,x)
50 xtitle('Analog Carrier Signal for Digital Modulation
        ')
51 xgrid
52 figure
53 plot(Xask)
54 xtitle('Amplitude Shift Keying')
55 xgrid
56 figure

```

```

57 plot(Xfsk)
58 xtitle('Frequency Shift Keying')
59 xgrid
60 figure
61 plot(Xpsk)
62 xtitle('Phase Shift Keying')
63 xgrid
64 //Example
65 //Enter the Analog Carrier Frequency 2
66 //Enter the digital binary data[0,1,1,0,1,0,0,1]

```

Scilab code Exa 7.1 clear

```

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

```

```

21     S2 = [S2 s2(m(i),:)]];
22 end
23 S = S1+S2;
24 figure
25 subplot(3,1,1)
26 a =gca();
27 a.x_location = "origin";
28 plot(S1)
29 title('Binary PSK wave of Odd-numbered bits of input
        sequence')
30 subplot(3,1,2)
31 a =gca();
32 a.x_location = "origin";
33 plot(S2)
34 title('Binary PSK wave of Even-numbered bits of
        input sequence')
35 subplot(3,1,3)
36 a =gca();
37 a.x_location = "origin";
38 plot(S)
39 title('QPSK waveform')
40 // -sin((2*i-1)*%pi/4)*%i;
41 // annot = dec2bin([0:length(y)-1],log2(M));
42 // disp(y,'coordinates of message points')
43 // disp(annot,'dibits value')
44 // figure;
45 // a =gca();
46 // a.data_bounds = [-1,-1;1,1];
47 // a.x_location = "origin";
48 // a.y_location = "origin";
49 // plot2d(real(y(1)),imag(y(1)),-2)
50 // plot2d(real(y(2)),imag(y(2)),-4)
51 // plot2d(real(y(3)),imag(y(3)),-5)
52 // plot2d(real(y(4)),imag(y(4)),-9)
53 // xlabel('
        Phase');
54 // ylabel('

```

In-


```

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

```

Scilab code Exa 7.02 clear

```

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

```

```

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

```

Scilab code Exa 7.2 clear

```

1 //clear//
2 //Caption:Signal Space diagram for coherent BPSK
3 //Example7.2: Sequence and Waveforms for MSK signal
4 //Table 7.2 signal space characterization of MSK
5 clear
6 clc;
7 close;
8 M =2;
9 Tb =1;
10 t1 = -Tb:0.01:Tb;
11 t2 = 0:0.01:2*Tb;
12 phi1 = cos(2*%pi*t1).* cos((%pi/(2*Tb))*t1);
13 phi2 = sin(2*%pi*t2).* sin((%pi/(2*Tb))*t2);
14 teta_0 = [0,%pi];
15 teta_tb = [%pi/2,-%pi/2];
16 S1 = [];
17 S2 = [];
18 for i = 1:M
19     s1(i) = cos(teta_0(i));
20     s2(i) = -sin(teta_tb(i));
21     S1 = [S1 s1(i)*phi1];
22     S2 = [S2 s2(i)*phi2];
23 end
24 for i = M:-1:1
25     S1 = [S1 s1(i)*phi1];
26     S2 = [S2 s2(i)*phi2];
27 end
28 Input_Sequence =[1,1,0,1,0,0,0];
29 S = [];

```

```

30 t = 0:0.01:1;
31 S = [S cos(0)*cos(2*%pi*t)-sin(%pi/2)*sin(2*%pi*t)];
32 S = [S cos(0)*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(%pi)*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 S = [S cos(0)*cos(2*%pi*t)-sin(-%pi/2)*sin(2*%pi*t)
];
38 y = [s1(1),s2(1);s1(2),s2(1);s1(2),s2(2);s1(1),s2(2)
];
39 disp(y,'coordinates of message points')
40 figure
41 subplot(3,1,1)
42 a = gca();
43 a.x_location = "origin";
44 plot(S1)
45 title('Scaled time function s1*phi1(t)')
46 subplot(3,1,2)
47 a = gca();
48 a.x_location = "origin";
49 plot(S2)
50 title('Scaled time function s2*phi2(t)')
51 subplot(3,1,3)
52 a = gca();
53 a.x_location = "origin";
54 plot(S)
55 title('Obtained by adding s1*phi1(t)+s2*phi2(t) on a
bit-by-bit basis')

```

Scilab code Exa 7.3 clear

```

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

```

```

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

Scilab code Exa 7.4 clear

```

1 //clear//
2 //Caption:Signal Space diagram for coherent BFSK
3 //Figure7.4 Signal Space Diagram for coherent BFSK
  system
4 clear
5 clc;
6 close;
7 M =2;
8 y = [1,0;0,1];
9 annot = dec2bin([M-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(y(1,1),y(1,2),-9)
18 plot2d(y(2,1),y(2,2),-5)
19 xlabel('
      In-Phase');
20 ylabel('
      Out-Phase');
```

```

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

```

Scilab code Exa 7.4.7.20 clear

```

1 //clear//
2 //Caption:Comparison of error probability of
    different data transmission schemes
3 //Table7.4:Figure 7.20
4 //Comparison of Symbol Error Probability
5 //of Different Digital Transmission System
6 clear;
7 close;
8 clc;
9 //Eb = Energy of the bit No = Noise Spectral
    Density
10 Eb_No = [18,0.3162278];
11 x = Eb_No(2):1/100:Eb_No(1);
12 x_dB = 10*log10(x);
13 for i = 1:length(x)
14 //Error Probability of Coherent BPSK
15 Pe_BPSK(i) = (1/2)*erfc(sqrt(x(i)));
16 //Error Probability of Coherent BFSK
17 Pe_BFSK(i) = (1/2)*erfc(sqrt(x(i)/2));
18 //Error Probability Non-Coherent PSK = DPSK
19 Pe_DPSK(i) = (1/2)*exp(-x(i));
20 //Error Probability Non-Coherent FSK
21 Pe_NFSK(i) = (1/2)*exp(-(x(i)/2));
22 //Error Probability of QPSK & MSK
23 Pe_QPSK_MSK(i) = erfc(sqrt(x(i)))-((1/4)*(erfc(
    sqrt(x(i)))^2));
24 end
25 a = gca();

```

```

26 a.data_bounds=[-5,0;12.5,0.5];
27 plot2d(x_dB,Pe_BPSK)
28 plot2d(x_dB,Pe_BFSK)
29 poly1= a.children(1).children(1);
30 poly1.foreground = 3;
31 plot2d(x_dB,Pe_DPSK)
32 poly1= a.children(1).children(1);
33 poly1.foreground = 4;
34 plot2d(x_dB,Pe_NFSK)
35 poly1= a.children(1).children(1);
36 poly1.foreground = 6;
37 plot2d(x_dB,Pe_QPSK_MSK)
38 poly1= a.children(1).children(1);
39 poly1.foreground = 7;
40 xlabel('Eb/No in dB ---->')
41 ylabel('Probability of Error Pe---->')
42 title('Comparison of Noise Performance of different
      PSK & FSK Scheme')
43 legend(['BPSK', 'BFSK', 'DPSK', 'Non-Coherent FSK', '
      QPSK & MSK'])
44 xgrid(1)

```

Scilab code Exa 7.06 clear

```

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

```

```

    signals ')
10 disp('
    -----
    ')
11 disp(M, 'M')
12 disp('
    -----
    ')
13 disp(Ruo, 'r in bits/s/Hz')
14 disp('
    -----
    ')

```

Scilab code Exa 7.6 clear

```

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

```



```

19 plot2d(real(y(2)), imag(y(2)), -4)
20 plot2d(real(y(3)), imag(y(3)), -5)
21 plot2d(real(y(4)), imag(y(4)), -9)
22 xlabel('
Phase');
23 ylabel('
Quadrature');
24 title('Constellation for QPSK')
25 legend(['message point 1 (dibit 10)'; 'message point
2 (dibit 00)'; 'message point 3 (dibit 01)'; '
message point 4 (dibit 11)'], 5)

```

In-

Scilab code Exa 7.7 clear

```

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

```

```

    ')
15 disp(Ruo, 'r in bits/s/Hz')
16 disp('
-----
    ')

```

Scilab code Exa 7.12.7.2 clear

```

1 //clear//
2 //Caption:Signal space diagram for MSK diagram
3 //Figure7.12 Signal Space Diagram for coherent MSK
  system
4 //Table 7.2 signal space characterization of MSK
5 clear
6 clc;
7 close;
8 M =2;
9 teta_0 = [0,%pi];
10 teta_tb = [%pi/2,-%pi/2];
11 for i = 1:M
12     s1(i) = cos(teta_0(i));
13     s2(i) = -sin(teta_tb(i));
14 end
15 y = [s1(1),s2(1);s1(2),s2(1);s1(2),s2(2);s1(1),s2(2)
      ];
16 disp(y,'coordinates of message points')
17 figure;
18 a =gca();
19 a.data_bounds = [-2,-2;2,2];
20 a.x_location = "origin";
21 a.y_location = "origin";
22 plot2d(y(1,1),y(1,2),-2)
23 plot2d(y(2,1),y(2,2),-4)
24 plot2d(y(3,1),y(3,2),-6)
25 plot2d(y(4,1),y(4,2),-9)

```

```

26 xlabel('
    In-Phase');
27 ylabel('
    Quadrature');
28 title('Constellation for MSK')
29 legend(['message point 1 (0, %pi/2)'; 'message point
    2 (%pi, %pi/2)'; 'message point 3 (%pi, - %pi/2)
    '; 'message point 4(0, - %pi/2)'],5)

```

Scilab code Exa 7.29 clear

```

1 //clear//
2 //Caption:Power Spectra of BPSK and BFSK signals
3 //Figure7.29:Comparison of Power Spectral Densities
  of BPSK
4 //and BFSK
5 clc;
6 rb = input('Enter the bit rate=');
7 Eb = input('Enter the energy of the bit=');
8 f = 0:1/100:8/rb;
9 Tb = 1/rb; //Bit duration
10 for i= 1:length(f)
11     if(f(i)==(1/(2*Tb)))
12         SB_FSK(i)=Eb/(2*Tb);
13     else
14         SB_FSK(i) = (8*Eb*(cos(%pi*f(i)*Tb)^2))/((%pi
            ^2)*(((4*(Tb^2)*(f(i)^2))-1)^2));
15     end
16     SB_PSK(i)=2*Eb*(sinc_new(f(i)*Tb)^2);
17 end
18 a=gca();
19 plot(f*Tb,SB_FSK/(2*Eb))
20 plot(f*Tb,SB_PSK/(2*Eb))

```

```

21 poly1= a.children(1).children(1);
22 poly1.foreground = 6;
23 xlabel('Normalized Frequency ——>')
24 ylabel('Normalized Power Spectral Density——>')
25 title('PSK Vs FSK Power Spectra Comparison')
26 legend(['Frequency Shift Keying', 'Phase Shift Keying
        '])
27 xgrid(1)
28 //Result
29 //Enter the bit rate in bits per second:2
30 //Enter the Energy of bit:1

```

Scilab code Exa 7.30 clear

```

1 //clear//
2 //Caption:Power Spectra of QPSK and MSK signals
3 //Figure7.30:Comparison of QPSK and MSK Power
  Spectrums
4 //clear;
5 //close;
6 //clc;
7 rb = input('Enter the bit rate in bits per second:')
  ;
8 Eb = input('Enter the Energy of bit:');
9 f = 0:1/(100*rb):(4/rb);
10 Tb = 1/rb; //bit duration in seconds
11 for i = 1:length(f)
12     if(f(i)==0.5)
13         SB_MSK(i) = 4*Eb*f(i);
14     else
15         SB_MSK(i) = (32*Eb/(%pi^2))*(cos(2*%pi*Tb*f(i))
            /((4*Tb*f(i))^2-1))^2;
16     end
17     SB_QPSK(i)= 4*Eb*sinc_new((2*Tb*f(i)))^2;
18 end

```

```

19 a = gca();
20 plot(f*Tb, SB_MSK/(4*Eb));
21 plot(f*Tb, SB_QPSK/(4*Eb));
22 poly1= a.children(1).children(1);
23 poly1.foreground = 3;
24 xlabel('Normalized Frequency ——>')
25 ylabel('Normalized Power Spectral Density ——>')
26 title('QPSK Vs MSK Power Spectra Comparison')
27 legend(['Minimum Shift Keying', 'QPSK'])
28 xgrid(1)
29 //Result
30 //Enter the bit rate in bits per second:2
31 //Enter the Energy of bit:1

```

Scilab code Exa 7.31 clear

```

1 //clear//
2 //Caption:Power spectra of M-ary PSK signals
3 //Figure7.31 Comparison of Power Spectral Densities
  of M-ary PSK signals
4 rb = input('Enter the bit rate=');
5 Eb = input('Enter the energy of the bit=');
6 f = 0:1/100:rb;
7 Tb = 1/rb; //Bit duration
8 M = [2,4,8];
9 for j = 1:length(M)
10   for i= 1:length(f)
11     SB_PSK(j,i)=2*Eb*(sinc_new(f(i)*Tb*log2(M(j)))
      ^2)*log2(M(j));
12   end
13 end
14 a=gca();
15 plot2d(f*Tb, SB_PSK(1, :)/(2*Eb))
16 plot2d(f*Tb, SB_PSK(2, :)/(2*Eb), 2)
17 plot2d(f*Tb, SB_PSK(3, :)/(2*Eb), 5)

```

```

18 xlabel('Normalized Frequency ——>')
19 ylabel('Normalized Power Spectral Density——>')
20 title('Power Spectra of M-ary signals for M =2,4,8')
21 legend(['M=2', 'M=4', 'M=8'])
22 xgrid(1)
23 //Result
24 //Enter the bit rate in bits per second:2
25 //Enter the Energy of bit:1

```

Scilab code Exa 7.41 clear

```

1 //clear//
2 //Caption:Matched Filter output of rectangular pulse
3 //Figure7.41
4 //Matched Filter Output
5 clear;
6 clc;
7 T =4;
8 a =2;
9 t = 0:T;
10 g = 2*ones(1,T+1);
11 h =abs(convol(g,g));
12 for i = 1:length(h)
13     if(h(i)<0.01)
14         h(i) =0;
15     end
16 end
17 h = h-T;
18 t1 = 0:length(h)-1;
19 figure
20 a =gca();
21 a.data_bounds = [0,0;6,4];
22 plot2d(t,g,5)
23 xlabel('t——>')
24 ylabel('g(t)——>')

```

```
25 title('Rectangular pulse duration  $T = 4$ ,  $a = 2$ ')
26 figure
27 plot2d(t1,h,6)
28 xlabel('t—>')
29 ylabel('Matched Filter output')
30 title('Output of filter matched to rectangular pulse
      g(t)')
```

Chapter 8

Error Control Coding

Scilab code Exa 8.1 clear

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

Scilab code Exa 8.2 clear


```

1 //clear//
2 //Caption:Hamming Codes
3 //Example8.2:Hamming codes
4 clear;
5 clc;
6 k = 4; //message bits length
7 n = 7; //block length
8 m = n-k; //Number of parity bits
9 I = eye(k,k); //identity matrix
10 disp(I, 'identity matrix Ik')
11 P = [1,1,0;0,1,1;1,1,1;1,0,1]; //coefficient matrix
12 disp(P, 'coefficient matrix P')
13 G = [P I]; //generator matrix
14 disp(G, 'generator matrix G')
15 H = [eye(k-1,k-1) P']; //parity check matrix
16 disp(H, 'parity checkk matrix H')
17 //message bits
18 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];

19 //
20 C = m*G;
21 C = modulo(C,2);
22 disp(C, 'Code words of (7,4) Hamming code')

```

Scilab code Exa 8.3 clear

```

1 //clear//
2 //Caption:Hamming Codes Revisited
3 //Example8.3:(7,4) Hamming Code Revisited
4 //message sequence = [1,0,0,1]
5 //D = poly(0,D);
6 clc;
7 D = poly(0, 'D');
8 g = 1+D+0+D^3; //generator polynomial

```

```

 9 m = (D^3)*(1+0+0+D^3); //message sequence
10 [r,q] = pdiv(m,g);
11 p = coeff(r);
12 disp(r,'remainder in polynomial form')
13 disp(p,'Parity bits are:')
14 G = [g;g*D;g*D^2;g*D^3];
15 G = coeff(G);
16 disp(G,'G')
17 G(3,:) = G(3,:)+G(1,:);
18 G(3,:) = modulo(G(3,:),2);
19 G(4,:) = G(1,:)+G(2,:)+G(4,:);
20 G(4,:) = modulo(G(4,:),2);
21 disp(G,'Generator Matrix G =')
22 h = 1+D^-1+D^-2+D^-4;
23 H_D = [D^4*h;D^5*h;D^6*h];
24 H_num = numer(H_D);
25 H = coeff(H_num);
26 H(1,:) =H(1,:)+H(3,:);
27 H(1,:) = modulo(H(1,:),2);
28 disp(H,'Partiy Check matrix H =')

```

Scilab code Exa 8.4 clear

```

1 //clear//
2 //Caption:Encoder for the (7,4) Cyclic Hamming Code
3 //Example8.4:Encoder for the (7,4) Cyclic hamming
  code
4 //message sequence = [1,0,0,1]
5 //D = poly(0,D);
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 disp('Table 8.3 Contents of the Shift Register in
      the Encoder of fig8.7 for Message Sequence(1001)')
14 disp('
      -----
      ')
15 disp('Shift          Input          Register
      Contents')
16 disp('
      -----
      ')
17 disp('1              1              1 1 0')
18 disp('2              0              0 1 1')
19 disp('3              0              1 1 1')
20 disp('4              1              0 1 1')
21 disp('
      -----
      ')

```

Scilab code Exa 8.5 clear

```

1 //clear//
2 //Caption:Syndrome calculator for the(7,4) Cyclic
  Hamming Code
3 //Example8.5: Syndrome calculator
4 //message sequence = [0,1,1,1,0,0,1]
5 clc;
6 D = poly(0, 'D');
7 g = 1+D+0+D^3; //generator polynomial
8 C1 = 0+D+D^2+D^3+0+0+D^6;//error free codeword
9 C2 = 0+D+D^2+0+0+0+D^6;//middle bit is error
10 [r1,q1] = pdiv(C1,g);
11 S1 = coeff(r1);
12 S1 = modulo(S1,2);

```

```

13 disp(r1,'remainder in polynomial form')
14 disp(S1,'Syndrome bits for error free codeword are:')
    )
15 [r2,q2] = pdiv(C2,g);
16 S2 = coeff(r2);
17 S2 = modulo(S2,2);
18 disp(r2,'remainder in polynomial form for errored
    codeword')
19 disp(S2,'Syndrome bits for errored codeword are:')

```

Scilab code Exa 8.6 clear

```

1 //clear//
2 //Caption:Reed–Solomon Codes
3 //Example8.6: Reed–Solomon Codes
4 //Single–error–correcting RS code with a 2–bit byte
5 clc;
6 m =2; //m–bit symbol
7 k = 1^2; //number of message bits
8 t =1; //single bit error correction
9 n = 2^m-1; //code word length in 2–bit byte
10 p = n-k; //parity bits length in 2–bit byte
11 r = k/n; //code rate
12 disp(n,'n')
13 disp(p,'n–k')
14 disp(r,'Code rate:r = k/n =')
15 disp(2*t,'It can correct any error upto =')

```

Scilab code Exa 8.7 clear

```

1 //clear//
2 //Caption:Convolutional Encoding – Time domain
    approach

```

```

3 //Example8.7: Convolutional Code Generation
4 //Time Domain Approach
5 close;
6 clc;
7 g1 = input('Enter the input Top Adder Sequence:= ');
8 g2 = input('Enter the input Bottom Adder Sequence:= '
9 )
9 m = input('Enter the message sequence:= ');
10 x1 = round(convol(g1,m));
11 x2 = round(convol(g2,m));
12 x1 = modulo(x1,2);
13 x2 = modulo(x2,2);
14 N = length(x1);
15 for i =1:length(x1)
16     x(i,:) =[x1(N-i+1),x2(N-i+1)];
17 end
18 x = string(x)
19 disp(x)
20 //Result
21 //Enter the input Top Adder Sequence:=[1,1,1]
22 //Enter the input Bottom Adder Sequence:=[1,0,1]
23 //Enter the message sequence:=[1,1,0,0,1]
24 //x =
25 //!1  1  !
26 //!   !
27 //!1  0  !
28 //!   !
29 //!1  1  !
30 //!   !
31 //!1  1  !
32 //!   !
33 //!0  1  !
34 //!   !
35 //!0  1  !
36 //!   !
37 //!1  1  !

```

Scilab code Exa 8.8 clear

```
1 //clear//
2 //Caption:Convolutional Encoding Transform domain
  approach
3 //Example8.8:Convolutional code – Transform domain
  approach
4 clc;
5 D = poly(0, 'D');
6 g1D = 1+D+D^2; //generator polynomial 1
7 g2D = 1+D^2; //generator polynomial 2
8 mD = 1+0+0+D^3+D^4; //message sequence polynomial
  representation
9 x1D = g1D*mD; //top output polynomial
10 x2D = g2D*mD; //bottom output polynomial
11 x1 = coeff(x1D);
12 x2 = coeff(x2D);
13 disp(modulo(x1,2), 'top output sequence')
14 disp(modulo(x2,2), 'bottom output sequence')
15 //Result
16 //top output sequence
17 // 1. 1. 1. 1. 0. 0. 1.
18 //
19 // bottom output sequence
20 // 1. 0. 1. 1. 1. 1. 1.
```

Scilab code Exa 8.11 clear

```
1 //clear//
2 //Caption:Fano metric for binary symmetric channel
  using convolutional code
```

```

3 //Example8.11: Convolutional code for binary
  symmetric channel
4 clc;
5 r = 1/2; //code rate
6 n =2; //number of bits
7 pe = 0.04; //transition probability
8 p = 1-pe; // probability of correct reception
9 gama_1 = 2*log2(p)+2*(1-r); //branch metric for
  correct reception
10 gamma_2 = log2(pe*p)+1; //branch metric for any one
  correct reception
11 gamma_3 = 2*log2(pe)+1; //branch metric for no
  correct reception
12 disp(gama_1,'branch metric for correct reception')
13 disp(gamma_2,'branch metric for any one correct
  reception')
14 disp(gamma_3,'branch metric for no correct reception
  ')
15 //branch metric for correct reception
16 //      0.8822126
17 //  branch metric for any one correct reception
18 //      - 3.7027499
19 //  branch metric for no correct reception
20 //      - 8.2877124

```

Chapter 9

Spread Spectrum Modulation

Scilab code Exa 9.1 clear

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



```

20     disp(x, 'x=')
21 end
22 m = [7,8,9,10,11,12,13,17,19];
23 N = 2^m-1;
24 disp('Table 9.1 Range of PN Sequence lengths')
25 disp('
-----
      ')
26 disp('Length of shift register (m)')
27 disp(m)
28 disp('PN sequence Length (N)')
29 disp(N)
30 disp('
-----
      ')
31 //RESULTEnter the period of the signal 7
32 // The PN sequence at step 1.
33 // x=      1.      0.      0.
34 // The PN sequence at step 2.
35 // x=      1.      1.      0.
36 // The PN sequence at step 3.
37 // x=      1.      1.      1.
38 // The PN sequence at step 4.
39 // x=      0.      1.      1.
40 // The PN sequence at step 5.
41 // x=      1.      0.      1.
42 // The PN sequence at step 6.
43 // x=      0.      1.      0.
44 // The PN sequence at step 7.
45 // x=      0.      0.      1.

```

Scilab code Exa 9.2 clear

```

1 //clear//
2 //Caption:Maximum length sequence property

```

```

3 //Example9.2 and Figure 9.2: Maximum-length sequence
4 //Period of PN Sequence N = 7
5 //Properties of maximum-length sequence
6 clc;
7 //Assign Initial value for PN generator
8 x0= 1;
9 x1= 0;
10 x2 =0;
11 x3 =0;
12 N = input('Enter the period of the signal')
13 one_count = 0;
14 zero_count = 0;
15 for i =1:N
16     x3 =x2;
17     x2 =x1;
18     x1 = x0;
19     x0 =xor(x1,x3);
20     disp(i,'The PN sequence at step')
21     x = [x1 x2 x3];
22     disp(x,'x=')
23     C(i) = x3;
24     if(C(i)==1)
25         C_level(i)=1;
26         one_count = one_count+1;
27     elseif(C(i)==0)
28         C_level(i)=-1;
29         zero_count = zero_count+1;
30     end
31 end
32 disp(C,'Output Sequence')//refer equation 9.4
33 disp(C_level,'Output Sequence levels')//refer
    equation 9.5
34 if(zero_count < one_count)
35     disp(one_count,'Number of 1s in the given PN
        sequence')
36     disp(zero_count,'Number of 0s in the given PN
        sequence')
37     disp('Property 1 (Balance property) is satisfied')

```

```

    )
38 end
39 Rc_tuo = corr(C_level,N);
40 t = 1:2*length(C_level);
41 //
42 figure
43 a =gca();
44 a.x_location = "origin";
45 plot2d(t,[C_level; C_level])
46 xlabel(' t')
47 title('Waveform of maximum-length sequence [0 0 1 1
    1 0 1 0 0 1 1 1 0 1]')
48 //
49 figure
50 a =gca();
51 a.x_location ="origin";
52 a.y_location ="origin";
53 plot2d([-length(Rc_tuo)+1:-1,0:length(Rc_tuo)-1],[
    Rc_tuo($:-1:2),Rc_tuo],5)
54 xlabel('
    tuo')
55 ylabel('
    Rc(tuo)')
56 title('Autocorrelation of maximum-length sequence')

```

Scilab code Exa 9.3 clear

```

1 //clear//
2 //Caption: Processing gain, PN sequence length,
    Jamming margin in dB
3 //Example9.3: Processing gain and Jamming Margin
4 clear;
5 clc;

```

```

6  close;
7  Tb = 4.095*10^-3; //Information bit duration
8  Tc = 1*10^-6; //PN chip duration
9  PG = Tb/Tc; //Processing gain
10 disp(PG, 'The processing gain is:')
11 N = PG; //PN sequence length
12 m = log2(N+1); //feedback shift register length
13 disp(N, 'The required PN sequence is:')
14 disp(m, 'The feedback shift register length:')
15 Eb_No = 10; //Energy to noise density ratio
16 J_P = PG/Eb_No; //Jamming Margin
17 disp(10*log10(J_P), 'Jamming Margin in dB:')
18 //Result
19 //The processing gain is: 4095.
20 //The required PN sequence is: 4095.
21 //The feedback shift register length: 12.
22 //Jamming Margin in dB: 26.122539

```

Scilab code Exa 9.4.9.5 clear

```

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

```

```

15 disp(k, 'length of PN sequence per hop k =')
16 disp(2^k, 'Total number of frequency hops =')
17 // Result
18 // number of bits per symbol K = 2.
19 // Number of MFSK tones M = 4.
20 // Period of the PN sequence N = 15.
21 // length of PN sequence per hop k = 3.
22 // Total number of frequency hops = 8.

```

Scilab code Exa 9.4.96 clear

```

1 // clear //
2 //Caption: Direct Sequence Spread Coherent BPSK
3 //Figure 9.4: Generation of waveforms in DS/BPSK
  spread spectrum transmitter
4 clear;
5 close;
6 clc;
7 t = 0:13;
8 N = 7;
9 wt = 0:0.01:1;
10 bt = [1*ones(1,N) -1*ones(1,N)];
11 ct = [0,0,1,1,1,0,1,0,0,1,1,1,0,1];
12 ct_polar = [-1,-1,1,1,1,-1,1,-1,-1,1,1,1,-1,1];
13 mt = bt.*ct_polar;
14 Carrier = 2*sin(wt*2*pi);
15 st = [];
16 for i = 1:length(mt)
17   st = [st mt(i)*Carrier];
18 end
19 //
20 figure
21 subplot(3,1,1)
22 a = gca();
23 a.x_location = "origin";

```

```

24 a.y_location = "origin";
25 a.data_bounds = [0,-2;20,2];
26 plot2d2(t,bt,5)
27 xlabel('

        t')
28 title('Data b(t)')
29 subplot(3,1,2)
30 a =gca();
31 a.x_location = "origin";
32 a.y_location = "origin";
33 a.data_bounds = [0,-2;20,2];
34 plot2d2(t,ct_polar,5)
35 xlabel('

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

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

        t')

```

```
54 title('Product Signal m(t)')
55 subplot(3,1,2)
56 a = gca();
57 a.x_location = "origin";
58 a.y_location = "origin";
59 a.data_bounds = [0,-2;20,2];
60 plot(Carrier)
61 xlabel('
        t')
62 title('Carrier Signal')
63 subplot(3,1,3)
64 a = gca();
65 a.x_location = "origin";
66 a.y_location = "origin";
67 a.data_bounds = [0,-2;20,2];
68 plot(st)
69 xlabel('
        t')
70 title('DS/BPSK signal s(t)')
71 //
```
