

Scilab Manual for  
Advanced Digital Communication  
by Prof S K Satyanarayana  
Electronics Engineering  
Sreenidhi Institute Of Science And  
Technology<sup>1</sup>

Solutions provided by  
Prof S K Satyanarayana  
Electronics Engineering  
Sreenidhi Institute Of Science And Technology

April 2, 2026

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



# Contents

|  |    |
|--|----|
| List of Scilab Solutions                               | 3  |
| 1 Reed Solomon Codes                                   | 5  |
| 2 Duobinary Encoder & Decoder                          | 7  |
| 3 Differential Phase Shift Keying                      | 10 |
| 4 PseudoNoise Sequence Generator                       | 13 |
| 5 Unipolar NRZ   | 17 |
| 6 Uniform Quantization PCM                             | 19 |
| 7 Convolutional Coding using Transform Domain Approach | 21 |
| 8 Duobinary Signaling - Amplitude & Phase Response     | 23 |
| 9 Power Spectrum of Discrete PAM Signal                | 25 |
| 10 Power Spectrum of MSK & QPSK                        | 27 |

# List of Experiments

|                |                          |    |
|----------------|--------------------------|----|
| Solution 1.1   | 1                        | 5  |
| Solution 2.2   | 2                        | 7  |
| Solution 3.3   | 3                        | 10 |
| Solution 4.4   | 4                        | 13 |
| Solution 5.5   | 5                        | 17 |
| Solution 6.6   | 6                        | 19 |
| Solution 7.7   | 7                        | 21 |
| Solution 8.8   | 8                        | 23 |
| Solution 9.9   | 9                        | 25 |
| Solution 10.10 | 10                       | 27 |
| AP 1           | sinc new                 | 29 |
| AP 2           | MSK QPSK new             | 30 |
| AP 3           | PS PAM new               | 31 |
| AP 4           | Duobinary Signalling new | 32 |
| AP 5           | Conv Code new            | 33 |
| AP 6           | uniformpcm               | 33 |
| AP 7           | xor                      | 34 |
| AP 8           | XOR new                  | 34 |
| AP 9           | ReedSolomonCodes         | 35 |

# List of Figures

|      |    |       |    |
|------|----|-------|----|
| 5.1  | 5  | ..... | 18 |
| 8.1  | 8  | ..... | 24 |
| 9.1  | 9  | ..... | 26 |
| 10.1 | 10 | ..... | 28 |

# Experiment: 1

## Reed Solomon Codes

check Appendix [AP 9](#) for dependency:

```
ReedSolomon_Codes.sci
```

### Scilab code Solution 1.1 1

```
1 //Reed-Solomon Codes
2 //Windows 7
3 //Scilab 6.0.0
4
5
6 //Note: Please run the ReedSolomon_Codes.sci
   dependency file before executing this program
7 n=16           //code word
8 k=4            //information bit
9 s=8            //no of bit symbols
10 ReedSolomon_Codes(n,k,s)
11
12 //n=16
13 //k=4
14 //s=8
15 //ReedSolomon_Codes(n,k,s)
16 //parity bits length in s-bit byte n-k=
```

```
17 //
18 //    12.
19 //
20 // Code rate:  $r = k/n =$ 
21 //
22 //    0.25
23 //
24 // It can detect any error upto =
25 //
26 //    12.
27 //
28 // It can correct any error upto =
29 //
30 //    6.
```

---

# Experiment: 2

## Duobinary Encoder & Decoder

check Appendix [AP 8](#) for dependency:

```
xor_new.sci
```

### Scilab code Solution 2.2 2

```
1 //Duobinary Encoding
2 //Windows 7
3 //Scilab 6.0.0
4
5 clc;
6 close;
7 //Note: Please run the xor.sci dependency file
   before executing this program
8 //Note: Don't run the clear command after running
   the dependency(xor.sci) as it contains the
   required funtion which will be cleared by the
   clear command
9 b=[0 ,1 ,0 ,1 ,1 ,1 ,0]; // input binary sequence :
   precoder input
10 a(1)=xor_new(1,b(1));
11 if(a(1)==1)
12 a_volts(1)=1;
```

```

13 end
14 for k=2:length(b)
15     a(k)=xor_new(a(k-1),b(k));
16     if(a(k)==1)
17         a_volts(k)=1;
18     else
19         a_volts(k)=-1;
20     end
21 end
22 a=a';
23 a_volts=a_volts';
24 disp(a,'Pre coder output in binary form : ')
25 disp(a_volts,'Pre coder output in volts : ')
26
27 //Duobinary coder output in volts
28 c(1)=1+ a_volts(1);
29 for k =2:length(a)
30     c(k)=a_volts(k -1)+a_volts(k);
31 end
32 c=c';
33 disp(c,'Duobinary coder output in volts : ')
34
35 //Duobinary decoder output by applying decision rule
36 for k =1:length(c)
37     if(abs(c(k))>1)
38         b_r(k)=0;
39     else
40         b_r ( k ) = 1;
41     end
42 end
43 b_r=b_r';
44 disp(b_r,'Recovered original sequence at detector
45     output :')
46 //Output
47 // Pre coder output in binary form :
48 //
49 //  1.   0.   0.   1.   0.   1.   1.

```

```
50 //  
51 // Pre coder output in volts :  
52 //  
53 // 1. -1. -1. 1. -1. 1. 1.  
54 //  
55 // Duobinary coder output in volts :  
56 //  
57 // 2. 0. -2. 0. 0. 0. 2.  
58 //  
59 // Recovered original sequence at detector output :  
60 //  
61 // 0. 1. 0. 1. 1. 1. 0.
```

---

# Experiment: 3

## Differential Phase Shift Keying

check Appendix [AP 8](#) for dependency:

```
xor_new.sci
```

### Scilab code Solution 3.3 3

```
1 //Generation of Differential Phase Shift Keying
  Signal
2 //Windows 7
3 //Scilab 6.0.0
4
5 clc;
6 close;
7 //Note: Please run the xor_new.sci dependency file
  before executing this program
8 //Note: Don't run the clear command after running
  the dependency(xor_new.sci) as it contains the
  required funtion which will be cleared by the
  clear command
9 bk = [1,0,1,1,0,1,1,1]; //input digital sequence
10 for i = 1:length(bk)
11   if(bk(i)==1)
12     bk_not(i) =~1;
```

```

13     else
14         bk_not(i)= 1;
15     end
16 end
17 dk_1(1) = 1&bk(1); //initial value of differential
    encoded sequence
18 dk_1_not(1)=0&bk_not(1);
19 dk(1) = xor_new(dk_1(1),dk_1_not(1))//first bit of
    dpsk encoder
20 for i=2:length(bk)
21     dk_1(i) = dk(i-1);
22     dk_1_not(i) = ~dk(i-1);
23     dk(i) = xor_new((dk_1(i)&bk(i)),(dk_1_not(i)&
        bk_not(i)));
24 end
25 for i =1:length(dk)
26     if(dk(i)==1)
27         dk_radians(i)=0;
28     elseif(dk(i)==0)
29         dk_radians(i)=%pi;
30     end
31 end
32 disp(bk, '(bk)')
33 bk_not = bk_not';
34 disp(bk_not, '(bk_not)')
35 dk = dk';
36 disp(dk, 'Differentially encoded sequence (dk)')
37 dk_radians = dk_radians';
38 disp(dk_radians, 'Transmitted phase in radians')
39
40 //Output
41 //(bk)
42 //
43 //  1.   0.   1.   1.   0.   1.   1.   1.
44 //
45 // (bk_not)
46 //
47 //  0.   1.   0.   0.   1.   0.   0.   0.

```

```
48 //
49 // Differentially encoded sequence (dk)
50 //
51 // 1. 0. 0. 0. 1. 1. 1. 1.
52 //
53 // Transmitted phase in radians
54 //
55 //
56 //      column 1 to 7
57 //
58 // 0. 3.1415927 3.1415927 3.1415927 0.
   // 0. 0.
59 //
60 //      column 8
61 //
62 // 0.
63 //
```

---

# Experiment: 4

## PseudoNoise Sequence Generator

check Appendix [AP 7](#) for dependency:

```
xor.sci
```

### Scilab code Solution 4.4 4

```
1
2 //Generate Maximum Length Pseudo Noise Sequence
3 //Windows 7
4 //Scilab 6.0.0
5
6 //Note: Please run the xor.sci dependency file
   before executing this program
7 //Assign Initial value for PN generator
8 x0= 1;
9 x1= 0;
10 x2 =0;
11 x3 =0;
12 x4= 0;
13 x5= 0;
14 x6= 0;
```

```

15 x7= 0;
16 x8= 0;
17 N = input('Enter the period of the signal')
18 for i =1:N
19     x1 = x0;
20     x8 =x7
21     x7 =x6
22     x0 =xor(x7,x1)
23     x6 =x5
24     x5 =x4
25     x0 =xor(x1,x5)
26     x4 =x3
27     x3 =x2;
28     x2 =x1;
29     x0 =xor(x1,x3);
30     disp(i,'The PN sequence at step')
31     x = [x1 x2 x3 x4 x5 x6 x7 x8];
32     disp(x,'x=')
33 end
34 m = [7,8,9,10,11,12,13,17,19];
35 N1 = 2^m-1;
36 disp('Table Range of PN Sequence lengths')
37 disp('Length of shift register (m)')
38 disp(m)
39 disp('PN sequence Length (N)')
40 disp(N1)
41
42 //Execution
43 //Enter the period of the signal
44 //5
45 //
46 //
47 // The PN sequence at step
48 //
49 // 1.
50 //
51 // x=
52 //

```

```

53 // 1. 1. 0. 0. 0. 0. 0. 0.
54 //
55 // The PN sequence at step
56 //
57 // 2.
58 //
59 // x=
60 //
61 // 1. 1. 1. 0. 0. 0. 0. 0.
62 //
63 // The PN sequence at step
64 //
65 // 3.
66 //
67 // x=
68 //
69 // 0. 0. 1. 1. 0. 0. 0. 0.
70 //
71 // The PN sequence at step
72 //
73 // 4.
74 //
75 // x=
76 //
77 // 1. 1. 0. 1. 1. 0. 0. 0.
78 // The PN sequence at step
79 //
80 // 5.
81 //
82 // x=
83 //
84 // 1. 1. 1. 0. 1. 1. 0. 0.
85 //
86 // Table Range of PN Sequence lengths
87 //
88 // Length of shift register (m)
89 //

```

```
90 // 7. 8. 9. 10. 11. 12. 13. 17.  
    19.  
91 //  
92 // PN sequence Length (N)  
93 //  
94 //  
95 //          column 1 to 7  
96 //  
97 // 127. 255. 511. 1023. 2047. 4095.  
    8191.  
98 //  
99 //          column 8 to 9  
100 //  
101 // 131071. 524287.
```

---

# Experiment: 5

## Unipolar NRZ

Scilab code Solution 5.5 5

```
1 //Unipolar NRZ
2 //Windows 7
3 //Scilab 6.0.0
4
5 clc;
6 clear;
7 close;
8
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
16 //plotting
17 a =gca();
18 a.data_bounds = [0 -2;L*L1 2];
19 for i =1:L
```

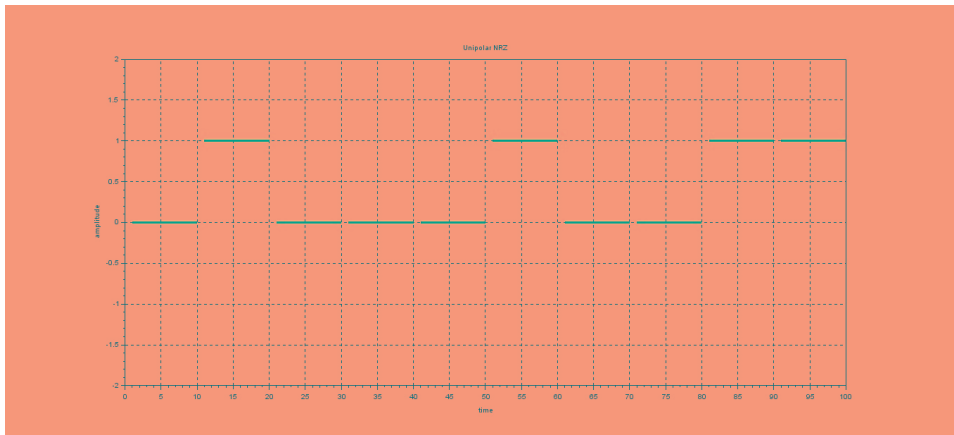


Figure 5.1: 5

```

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  else
25      plot([i*L-L+1:i*L],binary_one);
26      poly1= a.children(1).children(1);
27      poly1.thickness =3;
28  end
29  end
30  xgrid(1)
31  title('Unipolar NRZ')
32  xlabel('time')
33  ylabel('amplitude')

```

---

# Experiment: 6

## Uniform Quantization PCM

check Appendix [AP 6](#) for dependency:

```
uniform_pcm.sci
```

### Scilab code Solution 6.6 6

```
1
2 //Uniform Quantization – PCM
3 //Windows 7
4 //Scilab 6.0.0
5
6 //Note: Please run the uniform_pcm.sci dependency
   file before executing this program
7 x=[1,0,1,0,1,0,1,0] //input sequence
8 L=3 //no of quantization levels
9 [SQNR,xq,en_code] = uniform_pcm(x,L)
10 disp(SQNR, 'SQNR: ')
11 disp(xq, 'xq: ')
12 disp(en_code, 'en_code: ')
13
14 //Execution
15 // [SQNR,xq,en_code] = uniform_pcm(x,L)
16 // en_code =
```

```
17 //
18 // 2. 1. 2. 1. 2. 1. 2. 1.
19 //
20 // xq =
21 //
22 //
23 //      column 1 to 4
24 //
25 // 0.6666667 0. 0.6666667 0.
26 //
27 //      column 5 to 8
28 //
29 // 0.6666667 0. 0.6666667 0.
30 //
31 // SQNR =
32 //
33 // 9.5424251
```

---

# Experiment: 7

## Convolutional Coding using Transform Domain Approach

check Appendix [AP 5](#) for dependency:

```
ConvolutionCode_TransDomain_new.sci
```

### Scilab code Solution 7.7 7

```
1 //Convolutional Coding Using Transform Domain
   Approach
2 //Windows 7
3 //Scilab 6.0.0
4
5 clc;
6 close;
7 //Note: Please run the ConvolutionCode_TransDomain.
   sci dependency file before executing this program
8 //Note: Don't run the clear command after running
   the dependency(ConvolutionCode_TransDomain_new.
   sci) as it contains the required funtion which
   will be cleared by the clear command
9
10 //Execution
```

```
11 [x1D,x2D]= ConvolutionCode_TransDomain_new()
12
13 //Output
14 //Enter the generator polynomial 1=1+D^2+D^3
15 //
16 //Enter the generator polynomial 2=1+D^1
17 //
18 //Enter the message sequence 1+D^1+D^2+D^3+D^4
19 //
20 //
21 // top output sequence
22 //
23 // 1. 1. 0. 1. 1. 0. 0. 1.
24 //
25 // bottom output sequence
26 //
27 // 1. 0. 0. 0. 0. 1.
```

---

# Experiment: 8

## Duobinary Signaling - Amplitude & Phase Response

check Appendix [AP 4](#) for dependency:

```
Duobinary_Signaling_new.sci
```

### Scilab code Solution 8.8 8

```
1 //Duobinary Signaling Scheme – Magnitude and Phase  
  Response  
2 //Windows 7  
3 //Scilab 6.0.0  
4  
5 clc;  
6 close;  
7 //Note: Please run the Duobinary_Singaling.sci  
  dependency file before executing this program  
8 //Note: Don't run the clear command after running  
  the dependency(Duobinary_Signaling.sci) as it  
  contains the required funtion which will be  
  cleared by the clear command
```

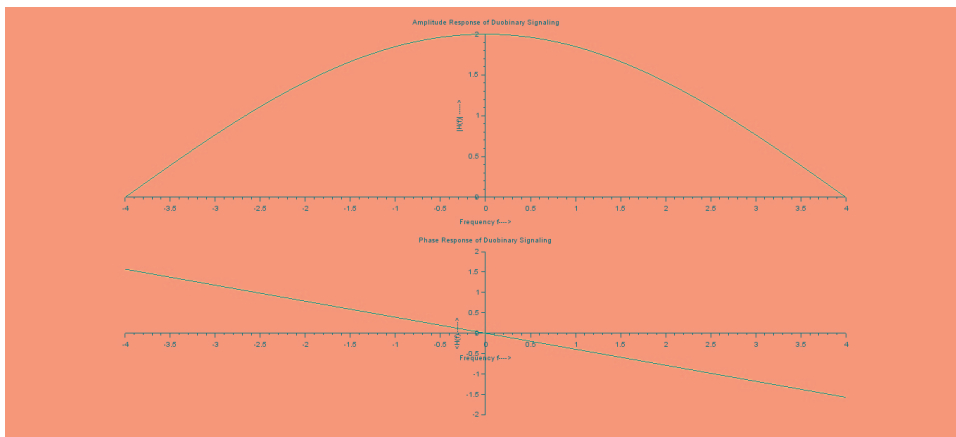


Figure 8.1: 8

```

9
10 //Execution
11 [Amplitude_Response,Phase_Response]=
    Duobinary_Signaling_new()
12
13 //Output
14 //Enter the bit rate= 8

```

---

# Experiment: 9

## Power Spectrum of Discrete PAM Signal

check Appendix [AP 3](#) for dependency:

```
PowerSpectra_PAM_new.sci
```

check Appendix [AP 1](#) for dependency:

```
sinc_newfunc_new.sci
```

### Scilab code Solution 9.9 9

```
1 //Power Spectrum Of Discrete PAM Signals
2 //Windows 7
3 //Scilab 6.0.0
4
5 clc;
6 close;
7 //Note: Please run the sinc_newfunc.sci dependency
   file before executing this program
8 //Note: Please run the PowerSpectra_PAM.sci
   dependency file before executing this program
```

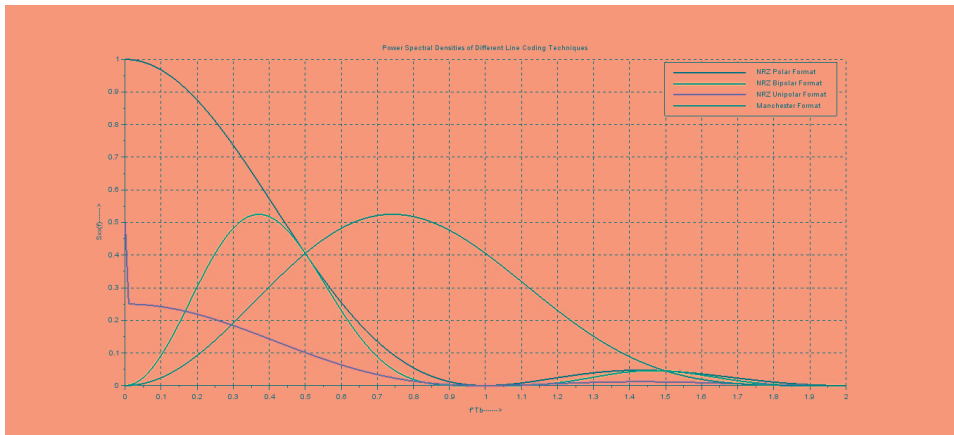


Figure 9.1: 9

```

9 //Note: Don't run the clear command after running
  the dependencies(sinc_newfunc_new.sci ,
  PowerSpectra_PAM_new.sci) as it contains the
  required funtion which will be cleared by the
  clear command
10
11
12 //Execution
13 [Sxxf_NRZ_P ,Sxxf_NRZ_BP ,Sxxf_NRZ_UP ,Sxxf_Manch]=
  PowerSpectra_PAM_new()
14
15 //Output
16 //Enter the Amplitude value:1
17 //Enter the bit rate:1

```

---

# Experiment: 10

## Power Spectrum of MSK & QPSK

check Appendix [AP 2](#) for dependency:

```
PowerSpectra_MSK_QPSK_new.sci
```

check Appendix [AP 1](#) for dependency:

```
sinc_newfunc_new.sci
```

### Scilab code Solution 10.10 10

```
1 //Power Spectrums of QPSK and MSK
2 //Windows 7
3 //Scilab 6.0.0
4
5 clc;
6 close;
7 //Note: Please run the sinc_newfunc.sci dependency
   file before executing this program
8 //Note: Please run the PowerSpectra_MSK_QPSK.sci
   dependency file before executing this program
```

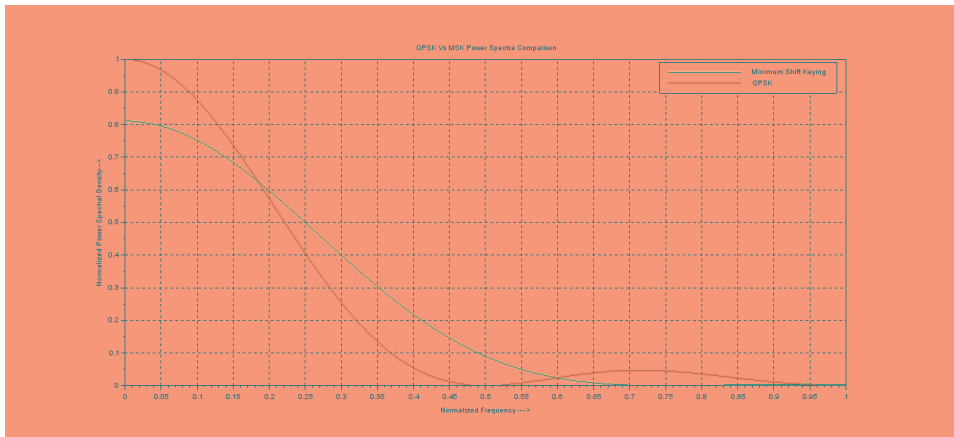


Figure 10.1: 10

```

9 //Note: Don't run the clear command after running
  the dependencies(sinc_newfunc_new.sci ,
  PowerSpectra_MSK_QPSK_new.sci) as it contains the
  required funtion which will be cleared by the
  clear command
10
11 //Execution
12 [SB_MSK,SB_QPSK]= PowerSpectra_MSK_QPSK_new()
13 //Enter the bit rate in bits per second:2
14 //Enter the Energy of bit:1

```

---

# Appendix

Scilab code AP 11 `clc;`

```
2 close;
3
4 function [y]=sinc_newfunc_new(x)
5 i=find(x==0);
6 x(i)= 1;
7 y = sin(%pi*x)./(%pi*x);
8 y(i) = 1;
9 endfunction
```

sinc new

---

Scilab code AP 12 `clc;`

```
2 close;
3
4 function [SB_MSK,SB_QPSK]= PowerSpectra_MSK_QPSK_new
   ()
5 rb = input('Enter the bit rate in bits per second:')
   ;
6 Eb = input('Enter the Energy of bit:');
7 f = 0:1/(100*rb):(4/rb);
8 Tb = 1/rb; //bit duration in seconds
9 for i = 1:length(f)
10   if(f(i)==0.5)
11     SB_MSK(i) = 4*Eb*f(i);
12   else
13     SB_MSK(i) = (32*Eb/(%pi^2))*(cos(2*%pi*Tb*f(i))
```

```

        /((4*Tb*f(i))^2-1))^2;
14  end
15  SB_QPSK(i)= 4*Eb*sinc_newfunc_new((2*Tb*f(i)))
        ^2;
16  end
17  a = gca();
18  plot(f*Tb,SB_MSK/(4*Eb));
19  plot(f*Tb,SB_QPSK/(4*Eb));
20  poly1= a.children(1).children(1);
21  poly1.foreground = 3;
22  xlabel('Normalized Frequency ——>')
23  ylabel('Normalized Power Spectral Density ——>')
24  title('QPSK Vs MSK Power Spectra Comparison')
25  legend(['Minimum Shift Keying', 'QPSK'])
26  xgrid(1)
27  endfunction

```

MSK QPSK new

---

```

Scilab code AP B clc;
2  close;
3
4  function [Sxxf_NRZ_P,Sxxf_NRZ_BP,Sxxf_NRZ_UP,
        Sxxf_Manch]=PowerSpectra_PAM_new()
5  a = input('Enter the Amplitude value:');
6  fb = input('Enter the bit rate:');
7  Tb = 1/fb; //bit duration
8  f = 0:1/(100*Tb):2/Tb;
9  for i = 1:length(f)
10   Sxxf_NRZ_P(i) = (a^2)*Tb*(sinc_newfunc_new(f(i)*Tb
        )^2);
11   Sxxf_NRZ_BP(i) = (a^2)*Tb*((sinc_newfunc_new(f(i)*
        Tb))^2)*((sin(%pi*f(i)*Tb))^2);
12   if (i==1)
13     Sxxf_NRZ_UP(i) = (a^2)*(Tb/4)*((sinc_newfunc_new
        (f(i)*Tb))^2)+(a^2)/4;
14   else

```

```

15     Sxxf_NRZ_UP(i) = (a^2)*(Tb/4)*((sinc_newfunc_new
        (f(i)*Tb))^2);
16     end
17     Sxxf_Manch(i) = (a^2)*Tb*(sinc_newfunc_new(f(i)*Tb
        /2)^2)*(sin(%pi*f(i)*Tb/2)^2);
18 end
19
20 //Plotting
21 a = gca();
22 plot2d(f,Sxxf_NRZ_P)
23 poly1= a.children(1).children(1);
24 poly1.thickness = 2; // the thickness of a curve.
25 plot2d(f,Sxxf_NRZ_BP,2)
26 poly1= a.children(1).children(1);
27 poly1.thickness = 2; // the thickness of a curve.
28 plot2d(f,Sxxf_NRZ_UP,5)
29 poly1= a.children(1).children(1);
30 poly1.thickness = 2; // the thickness of a curve.
31 plot2d(f,Sxxf_Manch,9)
32 poly1= a.children(1).children(1);
33 poly1.thickness = 2; // the thickness of a curve.
34 xlabel('f*Tb————>')
35 ylabel('Sxx(f)————>')
36 title('Power Spectral Densities of Different Line
        Coding Techniques')
37 xgrid(1)
38 legend(['NRZ Polar Format','NRZ Bipolar Format','NRZ
        Unipolar Format','Manchester Format']);
39 endfunction

```

---

PS PAM new

Scilab code AP 4 clc;

```

2 clear;
3 close;
4
5 function [Amplitude_Response,Phase_Response]=
        Duobinary_Signaling_new()

```

```

6 rb = input('Enter the bit rate=');
7 Tb =1/rb; //Bit duration
8 f = -rb/2:1/100:rb/2;
9 Amplitude_Response = abs(2*cos(%pi*f.*Tb));
10 Phase_Response = -(%pi*f.*Tb);
11 subplot(2,1,1)
12 a=gca();
13 a.x_location = "origin";
14 a.y_location = "origin";
15 plot(f,Amplitude_Response)
16 xlabel('Frequency f——>')
17 ylabel('|H(f)| ——>')
18 title('Amplitude Response of Duobinary Signaling')
19 subplot(2,1,2)
20 a=gca();
21 a.x_location = "origin";
22 a.y_location = "origin";
23 plot(f,Phase_Response)
24 xlabel('Frequency f——>')
25 ylabel('<H(f) ——>')
26 title('Phase Response of Duobinary Signaling')
27 endfunction

```

Duobinary Signalling new

---

**Scilab code AP 5 clc;**

```

2 clear;
3 close;
4
5 function [x1D,x2D]= ConvolutionCode_TransDomain_new
   ()
6 //g1D = generator polynomial 1
7 //g2D = generator polynomial 2
8 //x1D = top output sequence polynomial
9 //x2D = bottom output sequence polynomial
10 D = poly(0, 'D');
11 g1D = input('Enter the generator polynomial 1=') //
   generator polynomial 1

```

```

12 g2D = input('Enter the generator polynomial 2=') //
    generator polynomial 2
13 mD = input('Enter the message sequence')//message
    sequence polynomial representation
14 x1D = g1D*mD; //top output polynomial
15 x2D = g2D*mD; //bottom output polynomial
16 x1 = coeff(x1D);
17 x2 = coeff(x2D);
18 disp(modulo(x1,2),'top output sequence')
19 disp(modulo(x2,2),'bottom output sequence')
20 endfunction

```

Conv Code new

---

Scilab code AP 16 `clc;`

```

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

```

uniformpcm

---

**Scilab code AP 17** `clc;`

```
2 clear;
3 close;
4
5 // Function to perform XOR operation on the operands
6 function [value] = xor(A,B)
7     if(A==B)
8         value = 0;
9     else
10        value = 1;
11    end
12 endfunction
```

XOR

---

**Scilab code AP 18** `clc;`

```
2 clear;
3 close;
4
5 // Function to perform XOR operation on the operands
6 function [value] = xor_new(A,B)
7     if(A==B)
8         value = 0;
9     else
10        value = 1;
11    end
12 endfunction
```

XOR\_new

---

**Scilab code AP 19** `clc;`

```
2 clear;
3 close;
4
5 function [n,p,r] = ReedSolomon_Codes(n,k,s)
```

```

6 //Single-error-correcting RS code with a s-bit byte
7 //n=code word
8 //k=information bit
9 //s=no of bit symbols
10 t =(n-k)/2; //single bit error correction
11 //n = 2^s-1; //code word length in 2-bit byte
12 p = n-k; //parity bits length in 2-bit byte
13 r = k/n; //code rate
14 //disp(n,'code word length in s-bit byte n =')
15 disp(p,'parity bits length in s-bit byte n-k=')
16 disp(r,'Code rate:r = k/n =')
17 disp(2*t,'It can detect any error upto =')
18 disp(t,'It can correct any error upto =')
19 endfunction

```

ReedSolomonCodes

---