Scilab Manual for
Advanced Digital Communication
by Prof S K Satyanarayana
Electronics Engineering
Sreenidhi Institute Of Science And Technology[1]

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

May 18, 2024

[1]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 Experiments

List of Figures

Experiment: 1

Reed Solomon Codes

check Appendix AP 9 for dependency:

`ReedSolomon_Codes.sci`

Scilab code Solution 1.1 1

```
//Reed-Solomon Codes
//Windows 7
//Scilab 6.0.0


//Note: Please run the ReedSolomon_Codes.sci dependency file before executing this program
n=16            //code word
k=4             //information bit
s=8             //no of bit symbols
ReedSolomon_Codes(n,k,s)

//n=16
//k=4
//s=8
//ReedSolomon_Codes(n,k,s)
//parity bits length in  s-bit byte n-k=
```

```
//
//    12.
//
// Code rate:r = k/n =
//
//    0.25
//
// It can detect any error upto =
//
//    12.
//
// It can correct any error upto =
//
//    6.
```

Experiment: 2

Duobinary Encoder & Decoder

check Appendix AP 8 for dependency:

```
xor_new.sci
```

Scilab code Solution 2.2 2

```
//Duobinary Encoding
//Windows 7
//Scilab 6.0.0

clc;
close;
//Note: Please run the xor.sci dependency file
   before executing this program
//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
b=[0 ,1 ,0 ,1 ,1 ,1 ,0]; // input binary sequence :
   precoder input
a(1)=xor_new(1,b(1));
if(a(1)==1)
a_volts(1)=1;
```

```
end
for k=2:length(b)
    a(k)=xor_new(a(k-1),b(k));
    if(a(k)==1)
        a_volts(k)=1;
    else
        a_volts(k)=-1;
    end
end
a=a';
a_volts=a_volts';
disp(a,'Pre coder output in binary form : ')
disp(a_volts,'Pre coder output in volts : ')

//Duobinary coder output in volts
c(1)=1+ a_volts(1);
for k =2:length(a)
    c(k)=a_volts(k -1)+a_volts(k);
end
c=c';
disp(c,'Duobinary coder output in volts : ')

//Duobinary decoder output by applying decision rule
for k =1:length(c)
    if(abs(c(k))>1)
        b_r(k)=0;
    else
        b_r ( k ) = 1;
    end
end
b_r=b_r';
disp(b_r,'Recovered original sequence at detector output :')

//Output
// Pre coder output in binary form :
//
//    1.    0.    0.    1.    0.    1.    1.
```

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

```
//Generation of Differential Phase Shift Keying
   Signal
//Windows 7
//Scilab 6.0.0

clc;
close;
//Note: Please run the xor_new.sci dependency file
   before executing this program
//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
bk = [1,0,1,1,0,1,1,1];//input digital sequence
for i = 1:length(bk)
  if(bk(i)==1)
    bk_not(i) =~1;
```

```
  else
    bk_not(i)= 1;
  end
end
dk_1(1) = 1&bk(1);  //initial value of differential encoded sequence
dk_1_not(1)=0&bk_not(1);
dk(1) = xor_new(dk_1(1),dk_1_not(1))//first bit of dpsk encoder
for i=2:length(bk)
  dk_1(i) = dk(i-1);
  dk_1_not(i) = ~dk(i-1);
  dk(i) = xor_new((dk_1(i)&bk(i)),(dk_1_not(i)&bk_not(i)));
end
for i =1:length(dk)
  if(dk(i)==1)
    dk_radians(i)=0;
  elseif(dk(i)==0)
    dk_radians(i)=%pi;
  end
end
disp(bk,'(bk)')
bk_not = bk_not';
disp(bk_not,'(bk_not)')
dk = dk';
disp(dk,'Differentially encoded sequence (dk)')
dk_radians = dk_radians';
disp(dk_radians,'Transmitted phase in radians')

//Output
//(bk)
//
//    1.    0.    1.    1.    0.    1.    1.    1.
//
// (bk_not)
//
//    0.    1.    0.    0.    1.    0.    0.    0.
```

```
//
// Differentially encoded sequence (dk)
//
//   1.   0.   0.   0.   1.   1.   1.   1.
//
// Transmitted phase in radians
//
//
//         column 1 to 7
//
//   0.   3.1415927   3.1415927   3.1415927   0.
  0.   0.
//
//         column 8
//
//   0.
//
```

Experiment: 4

PseudoNoise Sequence Generator

check Appendix AP 7 for dependency:

xor.sci

Scilab code Solution 4.4 4

```

//Generate Maximum Length Pseudo Noise Sequence
//Windows 7
//Scilab 6.0.0

//Note: Please run the xor.sci dependency file
    before executing this program
//Assign Initial value for PN generator
x0= 1;
x1= 0;
x2 =0;
x3 =0;
x4= 0;
x5= 0;
x6= 0;
```

```
x7= 0;
x8= 0;
N = input('Enter the period of the signal')
for i =1:N
  x1 = x0;
  x8 =x7
  x7 =x6
  x0 =xor(x7,x1)
  x6 =x5
  x5 =x4
  x0 =xor(x1,x5)
  x4 =x3
  x3 =x2;
  x2 =x1;
  x0 =xor(x1,x3);
  disp(i,'The PN sequence at step')
  x = [x1 x2 x3 x4 x5 x6 x7 x8];
  disp(x,'x=')
end
m = [7,8,9,10,11,12,13,17,19];
Nl = 2^m-1;
disp('Table Range of PN Sequence lengths')
disp('Length of shift register (m)')
disp(m)
disp('PN sequence Length (N)')
disp(Nl)

//Execution
//Enter the period of the signal
//5
//
//
// The PN sequence at step
//
//    1.
//
// x=
//
```

```
//    1.    1.    0.    0.    0.    0.    0.    0.
//
// The PN sequence at step
//
//    2.
//
// x=
//
//    1.    1.    1.    0.    0.    0.    0.    0.
//
// The PN sequence at step
//
//    3.
//
// x=
//
//    0.    0.    1.    1.    0.    0.    0.    0.
//
// The PN sequence at step
//
//    4.
//
// x=
//
//    1.    1.    0.    1.    1.    0.    0.    0.
// The PN sequence at step
//
//    5.
//
// x=
//
//    1.    1.    1.    0.    1.    1.    0.    0.
//
// Table Range of PN Sequence lengths
//
// Length of shift register (m)
//
```

```
//    7.    8.    9.    10.    11.    12.    13.    17.
    19.
//
// PN sequence Length (N)
//
//
//           column 1 to 7
//
//    127.    255.    511.    1023.    2047.    4095.
    8191.
//
//           column 8 to 9
//
//    131071.    524287.
```

Experiment: 5

Unipolar NRZ

Scilab code Solution 5.5 5

```
//Unipolar NRZ
//Windows 7
//Scilab 6.0.0

clc;
clear;
close;

x = [0 1 0 0 0 1 0 0 1 1];
binary_zero = [0 0 0 0 0 0 0 0 0 0];
binary_one = [1 1 1 1 1 1 1 1 1 1];
L = length(x);
L1 = length(binary_zero);
total_duration = L*L;

//plotting
a =gca();
a.data_bounds =[0 -2;L*L1 2];
for i =1:L
```

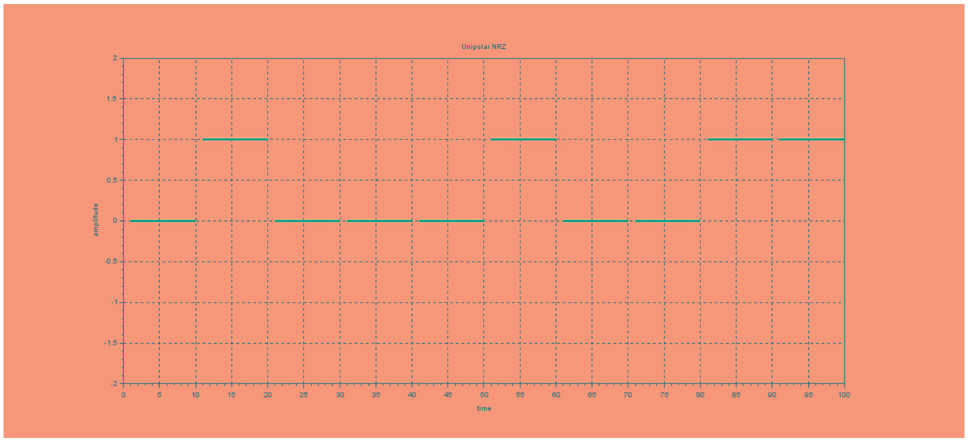

Figure 5.1: 5

```
  if(x(i)==0)
    plot([i*L-L+1:i*L],binary_zero);
    poly1= a.children(1).children(1);
    poly1.thickness =3;
  else
    plot([i*L-L+1:i*L],binary_one);
    poly1= a.children(1).children(1);
    poly1.thickness =3;
  end
end
xgrid(1)
title('Unipolar NRZ')
xlabel('time')
ylabel('amplitude')
```

Experiment: 6

Uniform Quantization PCM

check Appendix AP 6 for dependency:

`uniform_pcm.sci`

Scilab code Solution 6.6 6

```

//Uniform Quantization - PCM
//Windows 7
//Scilab 6.0.0

//Note: Please run the uniform_pcm.sci dependency file before executing this program
x=[1,0,1,0,1,0,1,0]        //input sequence
L=3             //no of quantization levels
[SQNR,xq,en_code] = uniform_pcm(x,L)
disp(SQNR,'SQNR:')
disp(xq,'xq:')
disp(en_code,'en_code:')

//Execution
//[SQNR,xq,en_code] = uniform_pcm(x,L)
// en_code  =
```

```
//
//    2.    1.    2.    1.    2.    1.    2.    1.
//
// xq  =
//
//
//          column 1 to 4
//
//    0.6666667    0.    0.6666667    0.
//
//          column 5 to 8
//
//    0.6666667    0.    0.6666667    0.
//
// SQNR  =
//
//    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

```
//Convolutional Coding Using Transform Domain
    Approach
//Windows 7
//Scilab 6.0.0

clc;
close;
//Note: Please run the ConvolutionCode_TransDomain.
    sci dependency file before executing this program
//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

//Execution
```

```
[x1D,x2D]= ConvolutionCode_TransDomain_new()

//Output
//Enter the generator polynomial 1=1+D^2+D^3
//
//Enter the generator polynomial 2=1+D^1
//
//Enter the message sequence1+D^1+D^2+D^3+D^4
//
//
// top output sequence
//
//    1.    1.    0.    1.    1.    0.    0.    1.
//
// bottom output sequence
//
//    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

```
//Duobinary Signaling Scheme - Magnitude and Phase Response
//Windows 7
//Scilab 6.0.0

clc;
close;
//Note: Please run the Duobinary_Singaling.sci dependecy file before executing this program
//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
```

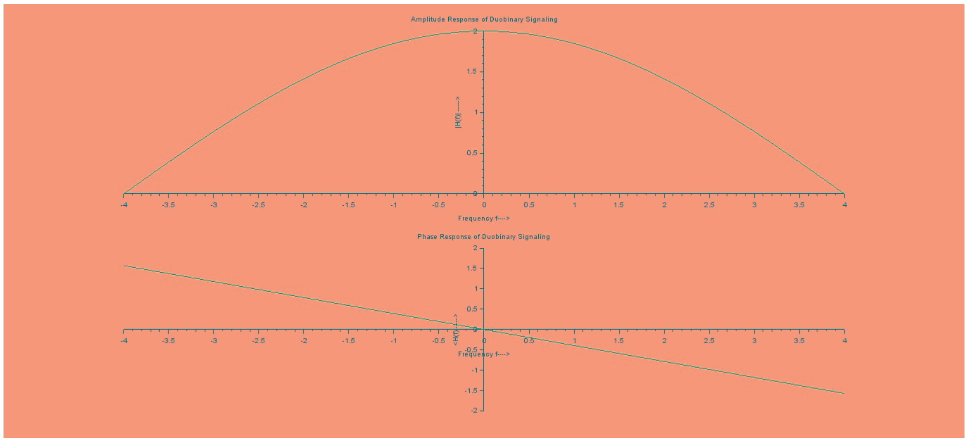

Figure 8.1: 8

```

//Execution
[Amplitude_Response,Phase_Response]=
   Duobinary_Signaling_new()

//Output
//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

```
//Power Spectrum Of Discrete PAM Signals
//Windows 7
//Scilab 6.0.0

clc;
close;
//Note: Please run the sinc_newfunc.sci dependency
   file before executing this program
//Note: Please run the PowerSpectra_PAM.sci
   dependency file before executing this program
```

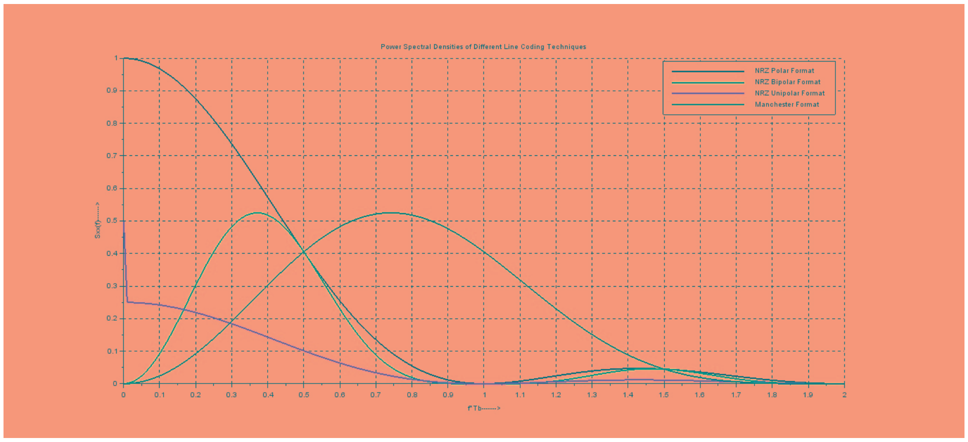

Figure 9.1: 9

```
//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


//Execution
[Sxxf_NRZ_P,Sxxf_NRZ_BP,Sxxf_NRZ_UP,Sxxf_Manch]=
    PowerSpectra_PAM_new()

//Output
//Enter the Amplitude value:1
//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

```
//Power Spectrums of QPSK and MSK
//Windows 7
//Scilab 6.0.0

clc;
close;
//Note: Please run the sinc_newfunc.sci dependency file before executing this program
//Note: Please run the PowerSpectra_MSK_QPSK.sci dependency file before executing this program
```

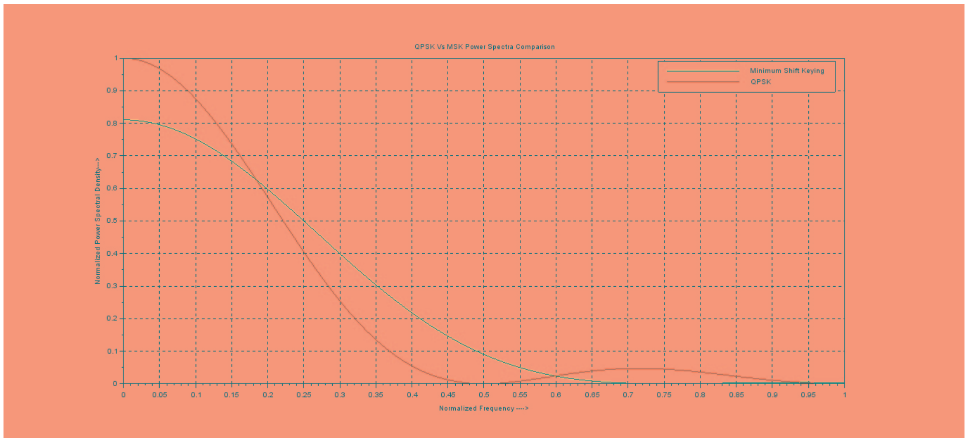

Figure 10.1: 10

```
//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

//Execution
[SB_MSK,SB_QPSK]= PowerSpectra_MSK_QPSK_new()
//Enter the bit rate in bits per second:2
//Enter the Energy of bit:1
```

Appendix

Scilab code AP 1

```
clc;
close;

function [y]=sinc_newfunc_new(x)
i=find(x==0);
x(i)= 1;
y = sin(%pi*x)./(%pi*x);
y(i) = 1;
endfunction
```

sinc new

Scilab code AP 2

```
clc;
close;

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

```
/((4*Tb*f(i))^2-1))^2;
  end
    SB_QPSK(i)= 4*Eb*sinc_newfunc_new((2*Tb*f(i)))^2;
end
a = gca();
plot(f*Tb,SB_MSK/(4*Eb));
plot(f*Tb,SB_QPSK/(4*Eb));
poly1= a.children(1).children(1);
poly1.foreground = 3;
xlabel('Normalized Frequency --->')
ylabel('Normalized Power Spectral Density--->')
title('QPSK Vs MSK Power Spectra Comparison')
legend(['Minimum Shift Keying','QPSK'])
xgrid(1)
endfunction
```

MSK QPSK new

Scilab code AP 13

```
clc;
close;

function [Sxxf_NRZ_P,Sxxf_NRZ_BP,Sxxf_NRZ_UP,Sxxf_Manch]=PowerSpectra_PAM_new()
a = input('Enter the Amplitude value:');
fb = input('Enter the bit rate:');
Tb = 1/fb;  //bit duration
f = 0:1/(100*Tb):2/Tb;
for i = 1:length(f)
  Sxxf_NRZ_P(i) = (a^2)*Tb*(sinc_newfunc_new(f(i)*Tb)^2);
  Sxxf_NRZ_BP(i) = (a^2)*Tb*((sinc_newfunc_new(f(i)*Tb))^2)*((sin(%pi*f(i)*Tb))^2);
  if (i==1)
    Sxxf_NRZ_UP(i) = (a^2)*(Tb/4)*((sinc_newfunc_new(f(i)*Tb))^2)+(a^2)/4;
  else
```

```
      Sxxf_NRZ_UP(i) = (a^2)*(Tb/4)*((sinc_newfunc_new(f(i)*Tb))^2);
    end
    Sxxf_Manch(i) = (a^2)*Tb*(sinc_newfunc_new(f(i)*Tb/2)^2)*(sin(%pi*f(i)*Tb/2)^2);
end

//Plotting
a = gca();
plot2d(f,Sxxf_NRZ_P)
poly1= a.children(1).children(1);
poly1.thickness = 2;  // the thickness of a curve.
plot2d(f,Sxxf_NRZ_BP,2)
poly1= a.children(1).children(1);
poly1.thickness = 2;  // the thickness of a curve.
plot2d(f,Sxxf_NRZ_UP,5)
poly1= a.children(1).children(1);
poly1.thickness = 2;  // the thickness of a curve.
plot2d(f,Sxxf_Manch,9)
poly1= a.children(1).children(1);
poly1.thickness = 2;  // the thickness of a curve.
xlabel('f*Tb------->')
ylabel('Sxx(f)------->')
title('Power Spectral Densities of Different Line Coding Techniques')
xgrid(1)
legend(['NRZ Polar Format','NRZ Bipolar Format','NRZ Unipolar Format','Manchester Format']);
endfunction
```

PS PAM new

Scilab code AP 14

```
clc;
clear;
close;

function [Amplitude_Response,Phase_Response]= Duobinary_Signaling_new()
```

```
rb =  input('Enter the bit rate=');
Tb =1/rb;  //Bit duration
f = -rb/2:1/100:rb/2;
Amplitude_Response = abs(2*cos(%pi*f.*Tb));
Phase_Response = -(%pi*f.*Tb);
subplot(2,1,1)
a=gca();
a.x_location ="origin";
a.y_location ="origin";
plot(f,Amplitude_Response)
xlabel('Frequency f----->')
ylabel('|H(f)| ----->')
title('Amplitude Response of Duobinary Signaling')
subplot(2,1,2)
a=gca();
a.x_location ="origin";
a.y_location ="origin";
plot(f,Phase_Response)
xlabel('                              Frequency f----->')
ylabel('                              <H(f) ----->')
title('Phase Response of Duobinary Signaling')
endfunction
```

Duobinary Signalling new

Scilab code AP 15

```
clc;
clear;
close;

function [x1D,x2D]= ConvolutionCode_TransDomain_new()
//g1D =  generator polynomial 1
//g2D =  generator polynomial 2
//x1D = top output sequence polynomial
//x2D = bottom output sequence polynomial
D = poly(0,'D');
g1D = input('Enter the generator polynomial 1=') //generator polynomial 1
```

```
g2D = input('Enter the generator polynomial 2=') //generator polynomial 2
mD = input('Enter the message sequence')//message sequence polynomial representation
x1D = g1D*mD; //top output polynomial
x2D = g2D*mD; //bottom output polynomial
x1 = coeff(x1D);
x2 = coeff(x2D);
disp(modulo(x1,2),'top output sequence')
disp(modulo(x2,2),'bottom output sequence')
endfunction
```

Conv Code new

Scilab code AP 16

```
clc;
clear;
close;

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

uniformpcm

Scilab code AP 17
```
clc;
clear;
close;

// Function to perform XOR operation on the operands
function [value] = xor(A,B)
  if(A==B)
    value = 0;
  else
    value = 1;
  end
endfunction
```

xor

Scilab code AP 18
```
clc;
clear;
close;

// Function to perform XOR operation on the operands
function [value] = xor_new(A,B)
  if(A==B)
    value = 0;
  else
    value = 1;
  end
endfunction
```

XOR new

Scilab code AP 19
```
clc;
clear;
close;

function[n,p,r] = ReedSolomon_Codes(n,k,s)
```

```
//Single-error-correcting RS code with a s-bit byte
//n=code word
//k=information bit
//s=no of bit symbols
t =(n-k)/2; //single bit error correction
//n = 2^s-1; //code word length in 2-bit byte
p = n-k; //parity bits length in  2-bit byte
r = k/n; //code rate
//disp(n,'code word length in s-bit byte n =')
disp(p,'parity bits length in  s-bit byte n-k=')
disp(r,'Code rate:r = k/n =')
disp(2*t,'It can detect any error upto =')
disp(t,'It can correct any error upto =')
endfunction
```

ReedSolomonCodes