

Scilab Manual for
Mobile Communication
by Prof Hetal Shah
Others
Dharmsinh Desai University¹

Solutions provided by
Prof Hetal Shah
Others
Dharmsinh Desai

July 17, 2025

¹Funded 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 Digital Modulation Functions: ASK, FSK, PSK generation.	4
2 Constellation diagram and Error Rate performance of different modulation techniques with AWGN channel.	7
3 Effect of various channel on transmitted data using different modulation techniques.	13
4 Trunking Theory for Probability of blocking(Erlang B) and probability of delay(Erlang C).	23
5 Walsh Code generation	27
6 PN sequence generation.	31
7 Equalization.	35
8 Channel Coding using Linear Block Code	38
9 Transmit and receive diversity	43
10 Speech coding	47

List of Experiments

Solution 1.1	1	4
Solution 2.1	Sigal space diagram of different modulation	7
Solution 2.2	BER of BPSK and QPSK over AWGN Channel	9
Solution 3.1	BER BPSK Rayleigh fading channel	13
Solution 3.2	BER QPSK Rayleigh channel	16
Solution 3.3	1	20
Solution 4.1	Traffic calculation inErlang B and Erlang C system	23
Solution 5.1	Walsh code generation and spreading and despreading using Walsh code	27
Solution 6.1	3 bit PN sequence generation and spreading and despreading using PN sequence and shifted PN sequence	31
Solution 7.1	Adaptive equalization using LMS filter	35
Solution 8.1	Linear Block Coding over AWGN channel	38
Solution 9.1	Selection Diversity over AWGN channel	43
Solution 9.2	Maximal Ratio Combining over AWGN and Rayleigh fading Channel	44
Solution 10.1	speech coding and Decoding using LPC	47

Experiment: 1

Digital Modulation Functions: ASK, FSK, PSK generation.

Scilab code Solution 1.1 1

```
1 // Amplitude Shift Keying , Frequency Shift Keying And
   Phase Shift keying waveform generation
2 clc;
3 clear;
4 xdel(winsid());
5 sym=10; //no. of symbols
6 g=[1 1 0 1 0 0 1 1 1 0 ] //binary data
7 f1=1;f2=2; //frequencies of carrier
8 t=0:2*pi/99:2*pi; //range of time
9 //ASK
10 cp=[];bit=[];mod_ask=[];mod_fsk=[];mod_psk=[];cp1
    =[];cp2=[];
11 for n=1:length(g); //ASK modulation // Zeros and
    ones are inserted for proper plot of message
    signal
12     if g(n)==0;
13         die=zeros(1,100);
14     else g(n)==1;
15         die=ones(1,100);
```

```

16    end
17    c_ask=sin(f1*t);
18    cp=[cp die];
19    mod_ask=[mod_ask c_ask];
20    end
21 ask=cp.*mod_ask; //ASK modulated signal
22
23 //FSK
24 for n=1:length(g);
25     if g(n)==0;
26         die=ones(1,100);
27         c_fsk=sin(f1*t);
28     else g(n)==1;
29         die=ones(1,100);
30         c_fsk=sin(f2*t);
31     end
32     cp1=[cp1 die];
33     mod_fsk=[mod_fsk c_fsk];
34 end
35 fsk=cp1.*mod_fsk; //FSK molated signal
36
37 //PSK
38 for n=1:length(g);
39     if g(n)==0;
40         die=ones(1,100);
41         c_psk=sin(f1*t);
42     else g(n)==1;
43         die=ones(1,100);
44         c_psk=-sin(f1*t);
45     end
46     cp2=[cp2 die];
47     mod_psk=[mod_psk c_psk];
48 end
49 psk=cp2.*mod_psk; //PSK modulated signal
50 subplot(4,1,1);plot(cp,'LineWidth',1.5); // plot
      binary signal
51 xgrid;
52 title('Binary Signal');//title

```

```

53 mtlb_axis([0 100*length(g) -2.5 2.5]); // axis range
54 subplot(4,1,2);plot(ask,'LineWidth',1.5); // plot of
      ASK modulated signal
55 xgrid;
56 title('ASK modulation'); // title of plot
57 mtlb_axis([0 100*length(g) -2.5 2.5]); // axis range
58 subplot(4,1,3);plot(fsk,'LineWidth',1.5); // plot of
      FSK modulated signal
59 xgrid;
60 title('FSK modulation'); // title of plot
61 mtlb_axis([0 100*length(g) -2.5 2.5]); // axis range
62 subplot(4,1,4);plot(psk,'LineWidth',1.5); // plot of
      PSK modulated signal
63 xgrid;
64 title('PSK modulation'); // title of plot
65 mtlb_axis([0 100*length(g) -2.5 2.5]); // range of
      axis
66 //Result: This experiment results plots of binary
      data , ASK modulation ,FSK modulation and PSK
      modulation

```

Experiment: 2

Constellation diagram and Error Rate performance of different modulation techniques with AWGN channel.

Scilab code Solution 2.1 Sigal space diagram of different modulation

```
1 // Constellation diagram of BPSK and QPSK modulation  
    and BPSK and QPSK modulation over AWGN channel  
2 clc;  
3 clear;  
4 xdel(winsid());  
5 sym=20; //No . of symbols  
6 data1=grand(1,sym,"uin",0,1); //Random symbol  
    generation from 0 to 1 with uniform distribution  
7 snr=10; // Signal to Noise Ratio  
8 qpsk_mod=[];  
9 bpsk_mod=2*data1-1; //BPSK Modulation  
10 for j=1:2:length(data1)// Separation of I & Q  
    component for QPSK modulation  
11     i_phase=2*data1(j)-1; //BPSK modulation of I phase  
        component
```

```

12     q_phase=2*data1(j+1)-1; //BPSK modulation of Q
        phase component
13     temp=i_phase+%i*q_phase; //Combinibg I phase and Q
        phase component for QPSK modulation
14     qpsk_mod=[qpsk_mod temp]; //QPSK modulated signal
15 end
16
17 noise=1/sqrt(2)*(10^(-(snr/20)))*(rand(1,length(
        bpsk_mod), 'normal')+%i*(rand(1,length(bpsk_mod)
        , 'normal'))); //White gaussian noise generation
        for bpsk
18 noise1=1/sqrt(2)*(10^(-(snr/20)))*(rand(1,length(
        qpsk_mod), 'normal')+%i*(rand(1,length(qpsk_mod)
        , 'normal'))); //White gaussian noise generation
        for qpsk
19 bpsk_awgn=bpsk_mod+noise; //BPSK Modulated signal
        passed over AWGN channel
20 qpsk_awgn=qpsk_mod+noise1; //QPSK Modulated signal
        passed over AWGN channel
21
22 figure//constellation diagram of ideal BPSK
        modulated signal and BPSK modulated signal with
        White Gaussian Noise
23 a = gca(); //to handle various object
24 a.data_bounds = [ -1 , -1;1 ,1];
25 a.x_location = "origin";
26 a.y_location = "origin";
27 plot2d ( real(bpsk_mod),imag(bpsk_mod),-2);
28 plot2d ( real(bpsk_awgn),imag(bpsk_awgn),-5);
29 xlabel( 'In phase' ); //X-axis label
30 ylabel( 'Quadrature phase' ); //Y-axis label
31 title( 'Constellation for BPSK with AWGN' ); //title
        of plot
32 legend(['Ideal message point'; 'message point with
        noise']); //legend
33 mtlb_axis([-2 2 -2 2]); //range of axis
34 figure//constellation diagram of ideal QPSK
        modulated signal and QPSK modulated signal with

```

```

        White Gaussian Noise
35 a = gca(); //to handle various object
36 a.data_bounds = [ -1 , -1;1 ,1];
37 a.x_location = "origin";
38 a.y_location = "origin";
39 plot2d ( real(qpsk_mod) ,imag(qpsk_mod) ,-2);
40 plot2d ( real(qpsk_awgn) ,imag(qpsk_awgn) ,-5);
41 xlabel( 'In phase' ); //X-axis label
42 ylabel( 'Quadrature phase' ); //Y-axis label
43 title( 'Constellation for QPSK with AWGN' ); //title
   of plot
44 legend(['Ideal message point'; 'message point with
   noise']); //legend
45 mtlb_axis([-2 2 -2 2]); //range of axis
46 //Result: Generates two plots: BPSK modulated signal
   with and without noise—figure -0
47 //QPSK modulated signal with
   and without noise—figure -1

```

Scilab code Solution 2.2 BER of BPSK and QPSK over AWGN Channel

```

1 // Performance comparison of Simulated BER and
   Theoritical BER of BPSK and QPSK modulation over
   AWGN channel
2 clc;
3 clear;
4 xdel(winsid());
5 sym=10000; //No .of symbols
6 M=4;
7 qpsk_mod=[]; i_phase=[];
8 data1=grand(1,sym,"uin",0,1); //Random Symbol
   generation from 0 to 1 with uniform distribution
9 for j=1:2:length(data1)// Separation of I & Q
   component
10 i_phase=2*data1(j)-1; // BPSK modulation of I

```

```

    phase component
11 q_phase=2*data1(j+1)-1; //BPSK modulation of Q
    phase component
12 temp=i_phase+%i*q_phase; //combining of I phase
        and Q phase component for QPSK modulation
13 qpsk_mod=[qpsk_mod temp]; //QPSK modulated signal
14 end
15 bpsk_mod=2*data1-1; //BPSK Modulated signal
16
17 snr=1:10; //Signal to Noise Ratio
18 for k=1:length(snr)
    H=1/sqrt(2)*(rand(1,length(qpsk_mod), 'normal'
        )+%i*(rand(1,length(qpsk_mod), 'normal')));
    );
20 noise1=1/sqrt(2)*(10^(-(k/20)))*(rand(1,
    length(qpsk_mod), 'normal')+%i*(rand(1,
    length(qpsk_mod), 'normal'))); //White
        Gaussian Noise generation for QPSK
21 noise=1/sqrt(2)*(10^(-(k/20)))*(rand(1,
    length(bpsk_mod), 'normal')+%i*(rand(1,
    length(bpsk_mod), 'normal'))); //White
        Gaussian Noise generation for QPSK
22 rec1_qpsk=qpsk_mod+noise1; //QPSK
        modulated signal over AWGN channel
23 rec1_bpsk= bpsk_mod+noise; //BPSK
        modulated signal over AWGN channel
24
25 rec_data_qpsk=[]; rec_data_bpsk=[];
26 rec1_i=real(rec1_qpsk); //Seperation
        of I phase and Q phase comopnent
        of received QPSK modulated signal
27 rec1_q=imag(rec1_qpsk);
28 //
29 for i=1:length(rec1_i)//QPSK Demodulation:
        BPSK demodulation of I phase and Q phase
        components
30 if rec1_i(i)>=0
            demod_out_i=1;

```

```

32         else rec1_i(i)<0
33             demod_out_i=0;
34         end
35         if rec1_q(i)>=0
36             demod_out_q=1;
37         else rec1_q(i)<0
38             demod_out_q=0;
39         end
40     rec_data_qpsk=[rec_data_qpsk demod_out_i
41                     demod_out_q]; //QPSK Demodulated signal
42     end
43     for i=1:1:length(data1) //BPSK Demodulation
44         if real(rec1_bpsk(i))>=0
45             demod_out_bpsk=1;
46         else real(rec1_bpsk(i))<0
47             demod_out_bpsk=0;
48         end
49     rec_data_bpsk=[rec_data_bpsk
50                     demod_out_bpsk]; //BPSK Demodulated
51                     signal
52     end
53     errA=0;errB=0;
54     for i=1:sym
55         if rec_data_qpsk(i)==data1(i)
56             errA=errA;
57         else
58             errA=errA+1;
59         end
60     BER_qpsk(k)=errA/sym; // BER of QPSK
61     for i=1:sym
62         if rec_data_bpsk(i)==data1(i)
63             errB=errB;
64         else
65             errB=errB+1;
66         end

```

```

67
68     BER_bpsk(k)=errB/sym; //BER of BPSK
69 end
70 theoryBer = 0.5*erfc(sqrt(10.^ (snr/10))); //Theoretical BER of BPSK & QPSK
71 end
72
73 // end
74 snr=1:1:10;
75 plot2d(snr,BER_bpsk,5,logflag="nl"); //plot simulated
    BER of BPSK over AWGN channel
76 plot2d(snr,BER_qpsk,2,logflag="nl"); //plot simulated
    BER of QPSK over AWGN channel
77 plot2d(snr,theoryBer,3,logflag="nl"); //Plot
    theoretical BER of QPSK and BPSK over AWGN
    channel
78 mtlb_axis([0 20 10^-5 0.5]); //axis
79 xgrid(10);
80 xtitle('Bit Error Rate plot for BPSK & QPSK
    Modulation', 'SNR', 'BER'); //title of plot
81
82
83 legend(['BER_sim_BPSK'; 'BER_sim_QPSK'; 'BER_Theory']); //legend
84 //This experiments results plot of bit error rate(
    BER) comparison of simulated BPSK over AWGN
    channel, simulated QPSK over AWGN channel and
    theoretical BER of BPSK and QPSK
85 // It will take few minutes to get plots as 100000
    bits are applied as an input to get better plots

```

Experiment: 3

Effect of various channel on transmitted data using different modulation techniques.

Scilab code Solution 3.1 BER BPSK Rayleigh fading channel

```
1 // Error rate performance of BPSK modulated signal  
    over only AWGN channel and AWGN and Rayleigh  
    channel both  
2 clc;  
3 clear;  
4 xdel(winsid());  
5 sym=10000; //No .of symbols  
6 data1=grand(1,sym,"uin",0,1); //Randomly generated  
    Symbolsfrom 0 to 1with uniform distribution  
7  
8 bpsk_mod=2*data1-1; //BPSK Modulation  
9 snr=1:20; //signal to Noise Ratio  
10 for k=1:1:length(snr)  
11  
12     H1=1/sqrt(2)*(rand(1,length(bpsk_mod),'  
        normal')+%i*(rand(1,length(bpsk_mod),'  
        normal'))); //Rayleigh fading generation
```

```

13
14 noise=1/sqrt(2)*(10^(-(k/20)))*(rand(1,
15     length(bpsk_mod), 'normal')+%i*(rand(1,
16     length(bpsk_mod), 'normal'))); // White
17     Gaussian Noise generation
18
19     rec1_bpsk=bpsk_mod+noise; //BPSK
20         modulated signal over AWGN channel
21     rec1_bpsk_ray1= H1.*bpsk_mod+noise; //
22         BPSK modulated signal over AWGN
23         channel and Rayleigh Fading
24         channel
25
26     rec1_bpsk_ray=conj(H1).*rec1_bpsk_ray1
27         ;//multiplication with conjugate of
28         rayleigh fading to nullify phase
29         because of Rayleigh Fading
30
31     // rec1_bpsk_ray=rec1_bpsk_ray1./(H1.*conj(H1));
32
33     rec_data_bpsk=[];rec_ray_bpsk=[];
34
35     for i=1:1:length(data1)//BPSK Demodulation
36         of received signal over AWGN channel
37         if real(rec1_bpsk(i))>=0
38             demod_out_bpsk=1;
39         else real(rec1_bpsk(i))<0
40             demod_out_bpsk=0;
41         end
42         rec_data_bpsk=[rec_data_bpsk
43             demod_out_bpsk];//Received signal
44
45         if real(rec1_bpsk_ray(i))>=0 //BPSK
46             Demodulation of received signal over
47             AWGN channel and Rayleigh channel
48             demod_ray_bpsk=1;
49         else real(rec1_bpsk_ray(i))<0
50             demod_ray_bpsk=0;
51         end

```

```

36         rec_ray_bpsk=[rec_ray_bpsk
37             demod_ray_bpsk];///Received signal
38
39         errB=0;errC=0;
40         for i=1:sym
41
42             if rec_data_bpsk(i)==data1(i)//Error rate
43                 calculation of received signal by
44                 considering only AWGN Channel
45                 errB=errB;
46             else
47                 errB=errB+1;
48             end
49
50             BER_bpsk(k)=errB/sym;//BER at receiver by
51                 considering only AWGN Channel
52
53             if rec_ray_bpsk(i)==data1(i)//Error rate
54                 calculation of received signal by
55                 considering AWGN Channel and Rayleigh
56                 channel
57                 errC=errC;
58             else
59                 errC=errC+1;
60             end
61
62             BER_bpsk_ray(k)=errC/sym;//BER at receiver
63                 by considering AWGN Channel and rayleigh
64                 channel
65         end    end
66
67         // end
68         snr=1:1:20;
69         plot2d(snr ,BER_bpsk ,5 ,logflag="nl");
70         plot2d(snr ,BER_bpsk_ray ,3 ,logflag="nl");
71         mtlb_axis([0 20 10^-5 0.5]);
72         xgrid(10);

```

```

65 xtitle( 'Bit Error Rate plot for BPSK modulated
    signal over AWGN channel and AWGN and Rayleigh
    channel both' , 'SNR' , 'BER' ) ;
66 legend([ 'BER_BPSK_AWGN' ; 'BER_BPSK_AWGN & Rayleigh' ])
    ;
67 //This experiment results plot of error rate
    performance of BPSK modulated signal over AWGN
    channel and AWGN and Rayleigh channel both.
68 //This experiment will take some time to display
    plot as higher no. of bits entered as an input to
    get better plots.

```

Scilab code Solution 3.2 BER QPSK Rayleigh channel

```

1 // Error rate performance of QPSK modulated signal
    over only AWGN channel and AWGN and Rayleigh
    channel both
2 clc;
3 clear;
4 xdel(winsid());
5 sym=10000; //No . of symbols
6 M=4;
7 qpsk_mod=[]; i_phase=[];
8 data1=grand(1,sym,"uin",0,1); //Random Symbol
    generation from 0 to 1 with uniform distribution
9 for j=1:2:length(data1)// Seperation of I & Q
    component
10    i_phase=2*data1(j)-1; // BPSK modulation of I
        phase component
11    q_phase=2*data1(j+1)-1; //BPSK modulation of Q
        phase component
12    temp=i_phase+%i*q_phase; //combining of I phase
        and Q phase component for QPSK modulation
13    qpsk_mod=[qpsk_mod temp]; //QPSK modulated signal
14 end

```

```

15
16     snr=1:5:41; // Signal to Noise Ratio
17     for k=1:length(snr)
18         H=1/sqrt(2)*(rand(1,length(qpsk_mod), 'normal'
19             ') + %i*(rand(1,length(qpsk_mod), 'normal'))))
20             ); // Rayleigh fading generation
21
22         noise1=1/sqrt(2)*(10^(-(k/20)))*(rand(1,
23             length(qpsk_mod), 'normal') + %i*(rand(1,
24             length(qpsk_mod), 'normal'))); // White
25             Gaussian Noise generation for QPSK
26
27         rec1_qpsk=qpsk_mod+noise1; //QPSK
28             modulated signal over AWGN channel
29         rec1_qpsk_ray1= H.*qpsk_mod+noise1;
30             //BPSK modulated signal over AWGN
31             channel and Rayleigh Fading
32             channel
33
34         rec1_qpsk_ray=conj(H).*rec1_qpsk_ray1
35             ; // multiplication with conjugate
36             of rayleigh fading to nullify
37             phase because of Rayleigh Fading
38
39         rec_data_qpsk=[]; rec_data_qpsk_ray
40             =[];
41
42         rec1_i=real(rec1_qpsk); // Separation
43             of I phase and Q phase component
44             of received QPSK modulated signal
45         rec1_q=imag(rec1_qpsk);
46
47         rec1_i_ray=real(rec1_qpsk_ray); //
48             Separation of I phase and Q phase
49             component of received QPSK
50             modulated signal
51         rec1_q_ray=imag(rec1_qpsk_ray);
52
53         // 
54         for i=1:length(rec1_i) //QPSK Demodulation:

```

```

BPSK demodulation of I phase and Q phase
components
35   if rec1_i(i)>=0
36       demod_out_i=1;
37   else rec1_i(i)<0
38       demod_out_i=0;
39   end
40   if rec1_q(i)>=0
41       demod_out_q=1;
42   else rec1_q(i)<0
43       demod_out_q=0;
44   end
45   if rec1_i_ray(i)>=0
46       demod_out_i_ray=1;
47   else rec1_i(i)<0
48       demod_out_i_ray=0;
49   end
50   if rec1_q_ray(i)>=0
51       demod_out_q_ray=1;
52   else rec1_q_ray(i)<0
53       demod_out_q_ray=0;
54   end
55   rec_data_qpsk=[rec_data_qpsk demod_out_i
56                   demod_out_q]; //QPSK Demodulated signal
56   rec_data_qpsk_ray=[rec_data_qpsk_ray
57                       demod_out_i_ray demod_out_q_ray]; //
58   QPSK Demodulated signal
57   end
58
59   errA=0;errB=0;
60   for i=1:sym
61       if rec_data_qpsk(i)==data1(i)
62           errA=errA;
63       else
64           errA=errA+1;
65       end
66   end
67   BER_qpsk(k)=errA/sym; // BER of QPSK

```

```

68
69      for i=1:sym
70      if rec_data_qpsk_ray(i)==data1(i)
71          errB=errB;
72      else
73          errB=errB+1;
74      end
75
76      BER_qpsk_ray(k)=errB/sym; //BER of BPSK
77  end
78 //theoryBer = 0.5*erfc(sqrt(10^(snr/10))); //
79 //Theoritical BER of BPSK & QPSK
80
81 // end
82 snr=1:5:41;
83 plot2d(snr,BER_qpsk,5,logflag="nl"); //plot simulated
84 BER of BPSK over AWGN channel
85 plot2d(snr,BER_qpsk_ray,2,logflag="nl"); //plot
86 simulated BER of QPSK over AWGN channel
87 //plot2d(snr,theoryBer,3,logflag="nl"); // Plot
88 //theoritical BER of QPSK and BPSK over AWGN
89 //channel
90 mtlb_axis([0 40 10^-5 0.5]); //axis
91 xgrid(10);
92 xtitle('Bit Error Rate plot for QPSK over AWGN
93 channel & AWGN and Rayleigh channel both', 'SNR',
94 'BER'); //title of plot
95
96 legend(['BER_QPSK_AWGN'; 'BER_QPSK_AWGN & Rayleigh']);
97 //legend
98 //This experiments results plot of bit error rate(
99 //BER) comparison of simulated QPSK over AWGN
100 channel, simulated QPSK over AWGN channel and
101 Rayleigh fading channel.
102 // It will take few minutes to get plots as 10000
103 bits are applied as an input to get better plots

```

Scilab code Solution 3.3 1

```
1 // Error rate performance of BPSK modulated signal  
    over only AWGN channel and AWGN and Rayleigh  
    channel both  
2 clc;  
3 clear;  
4 xdel(winsid());  
5 sym=10000; //No .of symbols  
6 data1=grand(1,sym,"uin",0,1); //Randomly generated  
    Symbolsfrom 0 to 1with uniform distribution  
7  
8 bpsk_mod=2*data1-1; //BPSK Modulation  
9 snr=1:20; //signal to Noise Ratio  
10 for k=1:1:length(snr)  
11  
12     H1=1/sqrt(2)*(rand(1,length(bpsk_mod),'  
        normal')+%i*(rand(1,length(bpsk_mod),'  
        normal'))); //Rayleigh fading generation  
13  
14     noise=1/sqrt(2)*(10^(-(k/20)))*(rand(1,  
        length(bpsk_mod), 'normal')+%i*(rand(1,  
        length(bpsk_mod), 'normal'))); // White  
        Gaussian Noise generation  
15  
16     rec1_bpsk=bpsk_mod+noise; //BPSK  
        modulated signal over AWGN channel  
17     rec1_bpsk_ray1= H1.*bpsk_mod+noise; //  
        BPSK modulated signal over AWGN  
        channel and Rayleigh Fading  
        channel  
18     rec1_bpsk_ray=conj(H1).*rec1_bpsk_ray1  
        ;// multiplication with conjugate of  
        rayleigh fading to nullify phase
```

```

        because of Rayleigh Fading
19      // rec1_bpsk_ray=rec1_bpsk_ray1./(H1.*
           conj(H1));
20
21      rec_data_bpsk=[]; rec_ray_bpsk=[];
22
23      for i=1:1:length(data1)//BPSK Demodulation
          of received signal over AWGN channel
24          if real(rec1_bpsk(i))>=0
25              demod_out_bpsk=1;
26          else real(rec1_bpsk(i))<0
27              demod_out_bpsk=0;
28          end
29          rec_data_bpsk=[rec_data_bpsk
                         demod_out_bpsk];// Received signal
30
31          if real(rec1_bpsk_ray(i))>=0 //BPSK
              Demodulation of received signal over
              AWGN channel and Rayleigh channel
32              demod_ray_bpsk=1;
33          else real(rec1_bpsk_ray(i))<0
34              demod_ray_bpsk=0;
35          end
36          rec_ray_bpsk=[rec_ray_bpsk
                         demod_ray_bpsk];// //Received signal
37      end
38
39      errB=0;errC=0;
40      for i=1:sym
41
42          if rec_data_bpsk(i)==data1(i)//Error rate
              calculation of received signal by
              considering only AWGN Channel
43              errB=errB;
44          else
45              errB=errB+1;
46          end
47

```

```

48     BER_bpsk(k)=errB/sym; //BER at receiver by
        considering only AWGN Channel
49
50     if rec_ray_bpsk(i)==data1(i)//Error rate
        calculation of received signal by
        considering AWGN Channel and Rayleigh
        channel
51         errC=errC;
52     else
53         errC=errC+1;
54     end
55
56     BER_bpsk_ray(k)=errC/sym; //BER at receiver
        by considering AWGN Channel and rayleigh
        channel
57     end   end
58
59     // end
60 snr=1:1:20;
61 plot2d(snr,BER_bpsk,5,logflag="nl");
62 plot2d(snr,BER_bpsk_ray,3,logflag="nl");
63 mtlb_axis([0 20 10^-5 0.5]);
64 xgrid(10);
65 xtitle('Bit Error Rate plot for BPSK modulated
        signal over AWGN channel and AWGN and Rayleigh
        channel both', 'SNR', 'BER') ;
66 legend(['BER_BPSK_AWGN'; 'BER_BPSK_AWGN & Rayleigh'])
;
67 //This experiment results plot of error rate
        performance of BPSK modulated signal over AWGN
        channe and AWGN and Rayleigh channel both.
68 //This experiment will take some time to display
        plot as higher no. of bits entered as an input to
        get better plots.

```

Experiment: 4

Trunking Theory for Probability of blocking(Erlang B) and probability of delay(Erlang C).

Scilab code Solution 4.1 Traffic calculation inErlang B and Erlang C system

```
1 //Exp-4 Calculates maximum traffic intensity and
   maximum no. of users accomodated in Erlang B and
   Erlang C system for given no of channels
2 clc;
3 clear;
4 xdel(winsid());
5
6 function [p1]=erlangB(A1,c1)// calculate blocking
   probability for Erlang B system
7     pr2=0;
8     pr1=A1^c1/factorial(c1);
9     for k=1:c1
10       pr2=pr2+(A1^k/factorial(k));
11     end
```

```

12      // A1=A1+1;
13      p1=pr1/pr2;
14  endfunction
15
16  function [p2]=erlangC(A2,c2)// calculate
    probability of blocked call delayed in Erlang C
    system
17      temp_1=0;
18      for k=0:c2-1
19          temp_1=temp_1+A2^k/factorial(k);
20      end
21      denominator=A^c2+(factorial(c2)*(1-(A2/c))*temp_1);
22      p2=A2^c2/denominator;
23  endfunction
24
25 pr_blocking=input('enter probability of blocking');
    //enter probability of blocking for perticular
    system
26 pr_delay=input('enter probability of block call
    delay');//enter probability of blocked call
    delayed for particular system
27 y=input('enter call rate');// Average no . of calls
    per minute
28 H=input('enter the average call duration');// 
    Average call duration in minute
29 c=input("enter no.of channels");//Enter no. of
    channels
30 disp("no.of channel=");
31 disp(c);
32 Au=y*H;// Traffic intensity per user
33
34 p=0;
35     for A=1:1:100
36         while(p<pr_blocking)//Find maximum traffic
            intensity for entered blocking
            probability pr_blocking
37         [p]=erlangB(A,c)//calling function erlangB
38         A=A+1;

```

```

39         end
40     disp(pr_blocking,'for blocking probability of '
41         );//display blocking probability
41     disp(A-1,'Maximum traffic intensity is');///
42         display max. traffic intensity
42     u=(A-1)/Au;//no. of users calculation
43     disp(u,"no .of users are accomodated");///
44         display maximum no.of users accomodated in
44         Erlang B system
44     break;
45 end//
46 p=0;
47 for A=1:1:100
48 while(p<pr_delay)//Find maximum traffic
49     intensity for entered blocking probability
50     pr_blocking
50 [p]=erlangC(A,c)//calling funtion to
50     calculate erlang C probability
50 A=A+1;
51 end
52 disp(pr_delay,'for block call delay
52     probability of');//display blocking
52     probability
53 disp(A-1,'Maximum traffic intensity is');///
53         display max. traffic intensity
54 u=(A-1)/Au;
55 disp(u,"no.of users are accomodated");///
55     display maximum no.of users accomodated in
55     Erlang C system
56 break;
57 end
58 //Enter blocking probability pr_blocking=0.01
59 //Enter probaboly of block call delay pr_delay
59 =0.1
60 //Enter call rate= 3/60
61 //enter call duration= 2(in minute)
62 //Enter no of channels 50
63

```

```
64 //Output:  
65 //no.of channel=      50.  
66  
67 //  for blocking probability of  0.01  
68 // Maximum traffic intensity is 38.  
69 // no .of users are accomodated 380.  
70  
71 //  for block call delay probability of 0.1  
72 // Maximum traffic intensity is 41.  
73 //no.of users are accomodated    410.
```

Experiment: 5

Walsh Code generation

Scilab code Solution 5.1 Walsh code generation and spreading and de-spreading using Walsh code

```
1 // Walsh Code generation
2 //Spreading and despreading of information for three
   users using Walsh code
3 clc;
4 clear;
5 xdel(winsid());
6 a=input('enter the number order of 2:');//input
   required length of Walsh Code which is always
   order of 2
7 c1=[1 -1 -1];//information of user 1
8 c2=[-1 1 -1];//information of user 2
9 c3=[1 -1 1];//information of user 3
10 W=[0 0;0 1];// Basic Walsh code Matrix
11 m=2;
12 %n=2^m;
13 for m =2:1:a
14 for i = 1:1:a//genration of walsh code matrix of
   entered length
15 if i==2^m
16 Winv=bitcmp(W,1);
```

```

17      W=[W W;W Winv];
18      end
19
20  end
21 end
22 temp=0;
23 W1=[];
24 disp(W)
25 for i=1:1:length(W(1,:)) //0 replaced by -1 in walsh
   code matrix
26   for j=1:1:length(W(1,:))
27     if W(i,j)==0 then
28       W(i,j)=W(i,j)-1;
29     else W(i,j)=W(i,j)+0;
30
31   end
32
33 end
34
35 end
36 //disp(W)
37 //spreading using Walsh code
38 tans_c1=[c1(1,1).*W(1,:) c1(1,2).*W(1,:) c1(1,3).*W
   (1,:)]; //spreading of user 1 information using
   first row of Walsh Matrix
39 tans_c2=[c2(1,1).*W(2,:) c2(1,2).*W(2,:) c2(1,3).*W
   (2,:)]; //spreading of user 2 information using
   second row of Walsh Matrix
40 tans_c3=[c3(1,1).*W(3,:) c3(1,2).*W(3,:) c3(1,3).*W
   (3,:)]; //spreading of user 3 information using
   third row of Walsh Matrix
41 aa1=tans_c1(1,1:a)+tans_c2(1,1:a)+tans_c3(1,1:a);
42 aa2=tans_c1(1,(a+1):(2*a))+tans_c2(1,(a+1):(2*a))+
   tans_c3(1,(a+1):(2*a));
43 aa3=tans_c1(1,((2*a))+1:(3*a))+tans_c2(1,((2*a))
   +1:(3*a))+tans_c3(1,((2*a))+1:(3*a));
44 tans_sig=[aa1 aa2 aa3]; //transmission of spreaded
   signal

```

```

45 det_code1=input('enter detection code');//Enter any
    integer no. ranging up to no. of rows of walsh
    matrix
46
47     select det_code1//select case to get information
        of entered user
48     case 1
        det_code=W(1,:);
49     case 2
        det_code=W(2,:);
50     case 3
        det_code=W(3,:);
51     else
        det_code=W(4,:);
52         disp('invalid detection code');//display
            message for input of invalid detection
            code
53
54     end
55
56
57
58
59
60 rec_sig =[det_code(1,:).*aa1 det_code(1,:).*aa2
    det_code(1,:).*aa3];//received signal multiplied
    with detection code
61 det_sig=[rec_sig(1,1)+rec_sig(1,2)+rec_sig(1,3)+
    rec_sig(1,4) rec_sig(1,5)+rec_sig(1,6)+rec_sig
    (1,7)+rec_sig(1,8) rec_sig(1,9)+rec_sig(1,10)+
    rec_sig(1,11)+rec_sig(1,12)];//detection of
    information from received signal
62 final_sig=(1/4)*det_sig;
63 disp('transmited information is');
64 disp(final_sig)//information transmmited using
    selected valid detection code
65 //input a=4
66 //W=[0 0 0 0 ;0 1 0 1;0 0 1 1 ;0 1 1 0]
67 //detection code=2, output=-1 1-1(information of
    user 2 spreaded with second row of Walsh Matrix)
68 //detection code > 3 , results : code not available
    0 0 0

```


Experiment: 6

PN sequence generation.

Scilab code Solution 6.1 3 bit PN sequence generation and spreading and despreading using PN sequence and shifted PN sequence

```
1 // Spreading of sequence using PN sequence and
   despreading of sequence using PN sequence and
   shifted PN sequence
2 clc;
3 clear;
4 xdel(winsid());
5 // Generation of 7 bit PN sequence
6 // Coefficient of polynomial
7 a1=1;
8 a2=1;
9 a3=1;
10 // Initial states of flip flop
11 R(1)=1;
12 R(2)=0;
13 R(3)=0;
14 m=3;
15 disp('output after every clock pulse');
16 for i=1:((2^m)-1)//shift of bit in each register for
   every clock pulse
17     r1=R(1);
```

```

18     r2=R(2);
19     r3=R(3);
20     PN(i)=R(3);
21     //if(a1==0)
22 R1=bitxor(r2,r3); //input of register is modulo2
    addition of R2 and R3
23 R(3)=R(2);
24 R(2)=R(1);
25 R(1)=R1;
26
27 disp(R);
28 end
29 disp('PN sequence is ');
30 disp(PN); //Display 7 bit PN sequence
31 c1=[1 -1 -1]; //information of user 1
32 for j=1:1:length(PN)//0 replaced with -1 in PN
    sequence
33 if PN(j)==0 then
34     PN(j)=PN(j)-1;
35 else PN(j)=PN(j)+0;
36 end
37
38 end
39 disp(PN);
40 spreaded_sig=[c1(1).*PN' c1(2).*PN' c1(3).*PN'] //
    Spreading of data of user 1 using PN sequence
41 detect_code=[spreaded_sig(1:7).*PN' spreaded_sig
    (8:14).*PN' spreaded_sig(15:21).*PN']; //at
    receiver, recieved spreaded signal multiplied
    with PN sequnce
42 corr_code=[sum(detect_code(1:7)) sum(detect_code
    (8:14)) sum(detect_code(15:21))];
43 rec_sig=(1/7).*corr_code;//get information form
    received signal
44 disp('received signal with correct PN sequence is ');
45 disp(rec_sig); //received data of user 1 at receiver
    :1 -1 -1
46 //Despreadng with shifted PN sequence

```

```

47 shift_fact=input('enter the shifting factor');
48 l=1;
49     k=shift_fact-1;
50 for i=1:1:length(PN)      //generation of shifted PN
sequence as per entered shifting factor
51 if i<=shift_fact
52     shift_seq(i)=PN(length(PN)-k);
53 k=k-1;
54 else i>shift_fact
55     shift_seq(i)=PN(l);
56 l=l+1;
57 end
58 end
59 disp('shifted sequence is');
60 disp(shift_seq); //display shifted sequence
61 //despread using shifted PN sequence
62 detect_shift_code=[spreaded_sig(1:7).*shift_seq'
spreaded_sig(8:14).*shift_seq' spreaded_sig
(15:21).*shift_seq'];
63 corr_shift_code=[sum(detect_shift_code(1:7)) sum(
detect_shift_code(8:14)) sum(detect_shift_code
(15:21))];
64 rec_shift_sig=(1/7).*corr_shift_code;
65 disp("recieved signal with shifted PN sequence is
");
66 disp(rec_shift_sig); //Invalid data received
because signal was despreaded with shifted PN
sequence
67 disp('which is not valid transmitted signal');
68 //Result:
69 //output of PN sequence generator after each
clock pulse
70 // PN =0 0 1 0 1 1 1 replace 0 with -1,PN=-1 -1 1
-1 1 1 1
71 //entered shifting factor =3, shifted PN sequence=
1 1 1 -1 -1 1 -1
72 //Invalid signal is received when despreading is
with shifted version of PN

```

```
73 // rec_shift_sig=- 0.1428571      0.1428571  
    0.1428571
```

Experiment: 7

Equalization.

Scilab code Solution 7.1 Adaptive equalization using LMS filter

```
1 // Least Mean Square adaptive equalizer
2 clc;
3 clear all;
4 xdel(winsid());
5 numPoints = 500;
6 numTaps = 1;           //channel order
7 Mu = 0.01;            //iteration step size
8
9 // input is guassian
10 x = rand(numPoints,1,'normal') + %i*rand(numPoints
11 ,1,'normal');
12 //choose channel to be random uniform
13 h = rand(numTaps,1) + %i*rand(numTaps, 1);
14
15 h = h/max(abs(h));   //normalize channel
16 // convolve channel with the input
17 d = filter(h, 1, x);
18
19 //initialize variables
20 w = [];
21 y = [];
```

```

21 in = [] ;
22 e = [] ; // error , final result to be computed
23
24 w = zeros(numTaps+1,1) + %i*zeros(numTaps+1,1);
25 kk=1;
26 aa(kk,:)=w';
27 //LMS Adaptation
28 for n = numTaps+1 : numPoints
29
30     // select part of training input
31     in = x(n : -1 : n-numTaps) ;
32     y(n) =w'* in;
33
34     // compute error
35     e(n) = d(n)-y(n);
36
37     // update taps
38
39 w = w+ Mu*( real(e(n)*conj(in)) - %i*imag(e(n)*conj(
    in)) ) ;
40
41 kk=kk+1;
42 aa(kk,:)=w';
43 end
44
45 // Plot results
46 figure;
47 iter=1:500
48 plot2d(iter,abs(e),5,logflag="nn");
49 title(['LMS Adaptation Learning Curve Using Mu =
    0.01']);
50 xlabel('Iteration Number');
51 ylabel('Output Estimation Error in dB');
52 figure;
53 plot3d(abs(aa(:,1)),abs(aa(:,2)),abs(e));
54 title('LMS adaption curve with weight factors');
55 xlabel('adaptive weight factor1');
56 ylabel('adaptive weight factor2');

```

```
57 zlabel('mean square error');  
58 // Output shows plot of MSE with no. of iterations  
    in figure 1 and 3D plot of MSE with weight  
    factors
```

Experiment: 8

Channel Coding using Linear Block Code

Scilab code Solution 8.1 Linear Block Coding over AWGN channel

```
1
2
3 // this is a linear block coding and decoding over
   awgn channel
4 // 4 bits input signal is coded with linear block
   code (4,7) , 7 bit coded signal is transmitted
   over awgn channel and at receiver side signal is
   decoded. If there is error in one bit , li//near
   block code correct that error and original
   transmitter code is received.
5 // If error is in more than one bit , code is not
   corrected so wrong code is received
6 clc;
7 clear all;
8 xdel(winsid());
9 global P n k;
10
11 n=7; //length of coded input
12 k=4; //length of input
```

```

13 P=[1 1 0; 0 1 1; 1 0 1;1 1 1]; // parity matrix of
   size k*(n-k) to be
14 // selected so that
   the systematic generator
15 // matrix is linearly
   independent or full rank
16 //
17
18 // (n,k) linear block code where k = no. of input
   data bits and n=no. of o/p
19 //data bits. code rate=k/n
20 // x is an input vector containing k bits
21
22 //This is an linear block encoding function
23 function y1=linblkcode(x);
24 global P n k;
25 n=7;
26 k=4;
27 P=[1 1 0; 0 1 1; 1 0 1;1 1 1]; // parity matrix
28 //x=[0 1 1 0];
29
30 //G=[ ]; // % Generator matrix k*n
31 G=[ eye(k,k) P];
32
33 y1=zeros(1,n);
34 for i=1:k//linear block coding
35   var(i,:)=x(1,i) & G(i,:);
36   var(i,:)=bool2s(var(i,:));
37   y1(1,:)=bitxor(var(i,:),y1(1,:)); //coded signal
38 end
39
40 endfunction
41
42
43 // %This is a linear block syndrome decoding function
   file%
44
45 function x1=linblkdecoder(y)

```

```

46 //%% here y is recieved vector 7 bits long
47
48 //%( 7 ,4) linear block code
49 global P n k;
50
51
52 //H=[ ] ; //% PARITY CHECK MATRIX
53
54 H=[P' eye((n-k),(n-k))];
55 Ht=H'; //%transpose of H
56
57 S=zeros(1,n-k); //syndrome of recieved vector x
58 for i=1:n-k// decoding of linear block code
59     S(i)=y(1) & Ht(1,i);
60     S(i)=bool2s(S(i));
61     for j=2:n
62         S(i)=bitxor(S(i), bool2s((y(j) & Ht(j,i))));
63             //decoded signal
64     end
65 end
66
67
68
69 //%%%%SYNDROME LOOK UP TABLE*****
70
71 //%%%%*****
72 //%
73 if S==[0 0 0]
74     e=[0 0 0 0 0 0 0];
75     z=bitxor(y,e);
76 end
77
78 if S==[0 0 1]
79     e=[0 0 0 0 0 0 1];
80     z=bitxor(y,e);
81 end
82 if S==[0 1 0]

```

```

83      e=[0 0 0 0 0 1 0];
84      z=bitxor(y,e);
85  end
86 if S==[1 0 0]
87     e=[0 0 0 0 1 0 0];
88     z=bitxor(y,e);
89 end
90 if S==[1 1 1]
91     e=[0 0 0 1 0 0 0];
92     z=bitxor(y,e);
93 end
94 if S==[1 0 1]
95     e=[0 0 1 0 0 0 0];
96     z=bitxor(y,e);
97 end
98 if S==[0 1 1]
99     e=[0 1 0 0 0 0 0];
100    z=bitxor(y,e);
101 end
102 if S==[1 1 0]
103     e=[1 0 0 0 0 0 0];
104     z=bitxor(y,e);
105 end
106 //disp('error');
107 //disp(e);
108 x1=z(1,1:k);
109 endfunction
110 snr_dB=2;
111
112 x=[1 0 0 1]; // input bits to the
113 // encoder of size 1* k
114 y1=linblkcode(x); // y1 is the output
115 // of linear block encoder
116 n1 = 1/sqrt(2)*[rand(1,length(y1), 'normal') + %i*
117 rand(1,length(y1), 'normal')]; //white gaussian
118 noise generation
119 r=y1+ 10^(-snr_dB/20)*n1; // received signal over awgn
120 channel

```

```

116 // r1=real(r)
117 rec=real(r)>=0.5; // detection of bit 1 and 0 in
    received signal
118 rec_fin=bool2s(rec); // convert boolean matrix to zero
    one matrix
119 // rec_err=rec_fin==y1;
120 // no_err=bool2s(rec_err);
121 disp('The information signal=')//display input
122 disp(x)
123 disp('The transmitted encoded signal=')//display
    coded signal
124 disp(y1)
125 disp('The received signal=')//display received
    signal
126 disp(rec_fin);
127 x1=linblkdecoder(rec_fin); // % x1 is the
    output of the linear block decoder
128 disp('The decoded signal=')//display decoded signal
129 disp(x1);
130 if x1==x then disp('one or less than one error so
    correct code is received');
131 else
132     disp('more than one error so wrong code detected
        ');
133 end
134 //Output: The information signal is : 1001
135 //transmitted code is : 1001001
136 //1. received signal is :1011001(e.g)(error in only
    one bit)
137 //decoded signal: 1001
138 //one or less than one error so correct code is
    received
139 //2. received signal is:1011011(e.g)(error in more
    than one bits)
140 //decoded signal:1010
141 //more than one error so wrong code is received

```

Experiment: 9

Transmit and receive diversity

Scilab code Solution 9.1 Selection Diversity over AWGN channel

```
1 //ber performance with 1, 2 and 3 receiver antennas  
    over awgn channel using selection diversity  
2 clc;  
3 clear;  
4 xdel(winsid());  
5 sym=10000; //no. of symbols  
6 data1=grand(1,sym,"uin",0,1); //randomly generated  
    input  
7 s = 2*data1-1; // BPSK modulation 0 -> -1; 1 -> 1  
8 nRx = [1 2 3]; //no .of receiving antennas  
9 snr_dB = [1:10]; // signal to noise ratio  
10 for j = 1:length(nRx)  
11     for i = 1:length(snr_dB)  
12         n = 1/sqrt(2)*[rand(nRx(j),sym,'normal') +  
            %i*rand(nRx(j),sym,'normal')]; //white  
            gaussian noise  
13  
14         y = ones(nRx(j),1)*s + 10^(-snr_dB(i)/20)*n;  
            //received signal over awgn channel  
15         [yHat1 ind] = mtlb_max(y,[],1); //find  
            strongest received signal from all
```

```

                antennas
16
17     ipHat1 = real(yHat1)>0;
18     ipHat = bool2s(ipHat1); // boolean to zero one
19             matrix conversion
20     // effective SNR
21     nErr(j,i) = size(find([data1- ipHat]),2); // 
22             no. of error calculation
23         end
24 end
25 simBer = nErr/sym; //BER calculation
26 // plot of ber comparison plot for 1,2 and 3
27 receiving antennas
28 snr_dB=1:10
29 plot2d(snr_dB,simBer(1,:),5,logflag="nl");
30 plot2d(snr_dB,simBer(2,:),2,logflag="nl");
31 plot2d(snr_dB,simBer(3,:),12,logflag="nl");
32 xgrid
33 legend( [ '1X1' ; '1X2' ; '1x3' ] );
34 xlabel('Number of receive antenna');
35 ylabel('effective SNR, dB');
36 title('SNR improvement with Selection Combining');
37 //output presents BER performance comparison plots
38 with 1,2 and 3 receiving antennas over awgn
39 channels

```

Scilab code Solution 9.2 Maximal Ratio Combining over AWGN and Rayleigh fading Channel

```

1 // BER Performance coamparison with one receivivng
2 atenna and two receiving antennas with Maximal
3 ratio Combining diversity technique over awgn
4 channe and rayleigh fading channel
5 clc;
6 clear;

```

```

4 xdel(winsid());
5 sym=100000; // no. of symbols
6 M=2;
7 data1=grand(1,sym,"uin",0,1); // input signal is
   randomly generated
8 //N = 10; % number of bits or symbols
9 //ip = rand(1,N)>0.5; % generating 0,1 with equal
   probability
10 s = 2*data1-1; // BPSK modulation 0 -> -1; 1 -> 1
11 nRx = [1 2]; //no of receivers
12 snr_dB = [1:20]; // signal to noise ration in dB
13 for jj = 1:length(nRx)
14     for ii = 1:length(snr_dB)
15         n = 1/sqrt(2)*[rand(nRx(jj),sym,'normal') +
           %i*rand(nRx(jj),sym,'normal')]; //white
           gaussian noise,
16         h = 1/sqrt(2)*[rand(nRx(jj),sym,'normal') +
           %i*rand(nRx(jj),sym,'normal')]; // //
           Rayleigh fading channel
17         // Channel and noise Noise addition
18         sD = kron(ones(nRx(jj),1),s);
19         y = h.*sD + 10^(-snr_dB(ii)/20)*n; //
           received signal over awgn channel and
           ayleigh fading channel
20         // finding the power of the channel on all
           rx chain
21         yHat = sum(conj(h).*y,1)./sum(h.*conj(h)
           ,1); // maximal ratio combining
22         // hPower1 = h.*conj(h);
23
24         ipHat = real(yHat)>0;
25         // effective SNR
26         nErr(jj,ii) = size(find([data1-
           ipHat]),2); //calculate error
27     end
28 end
29 simBer = nErr/sym; //bit error rate calculation
30 // plot

```

```
31 snr_dB=1:20
32 plot2d(snr_dB,simBer(1,:),5,logflag="nl");//snr- ber
      plot with one receiving antenna
33 plot2d(snr_dB,simBer(2,:),2,logflag="nl");//snr- ber
      plot with two receiving antennas
34 //plot(nRx,10*log10(EbN0EffSim),'bp-','LineWidth',2)
      ;
35 //mtlb_axis([1 20 0 6])
36 xgrid
37 legend(['1X1';'1X2']);
38 xlabel('Number of receive antenna');
39 ylabel('effective SNR, dB');
40 title('SNR improvement with Maximal ratio Combining',
      );
```

Experiment: 10

Speech coding

Scilab code Solution 10.1 speech coding and Decoding using LPC

```
1 //Exp-10 Speech coding using Long Term Predictive
   coder
2 //This code read wav file and play original signal
   and compressed signal
3 // It also plots original signal as well as
   compressed signal
4
5 function [aCoeff, tcount_of_aCoeff, e] =
   func_lev_durb(y, M);
6 //M=order and y is array of the data point of the
   current frame
7 sk=0;           //initializing summartion term "sk"
8 a=[zeros(M+1);zeros(M+1)]; //defining a matrix of
   zeros for "a" for init.
9 //MAIN BODY OF THIS PROGRAM STARTS FROM HERE
   >>>>>>>>>>>
10 z=xcorr(y);
11
12 //finding array of R[1]
13 R=z( ( (length(z)+1) ./2 ) : length(z)); //R=array
   of "R[1]", where l=0,1,...(b+N)-1 %R(1)=R[1]
```

```

=0] , R(2)=R[lag=1] , %R(3)=R[lag=2]... etc
14
15 //GETTING OTHER PARAMETERS OF PREDICTOR OF ORDER
   "0":
16 s=1;           //s=step no.
17 J(1)=R(1);      //J=array of "J1" , where l
   =0,1,2...(b+N)-1,    J(1)=J0 , J(2)=J1 , J(3)=J2 etc
18
19 //GETTING OTHER PARAMETERS OF PREDICTOR OF ORDER "(s
   -1)"
20 for s=2:M+1,
21     sk=0;           //clearing "sk" for each
   iteration
22     for i=2:(s-1),
23         sk=sk + a(i,(s-1)).*R(s-i+1);
24     end           //now we know value of "sk",
   the summation term
25           //of formula of calculating
   "k(1)"
26 k(s)=(R(s) + sk)./J(s-1);
27 J(s)=J(s-1).* (1-(k(s)).^2);
28
29 a(s,s)=-k(s);
30 a(1,s)=1;
31 for i=2:(s-1),
32     a(i,s)=a(i,(s-1)) - k(s).*a((s-i+1),(s-1));
33 end
34 end
35 //increment "b" and do same for next frame until end
   of frame when
36 //combining this code with other parts of LPC algo
37
38 //PREDICTION ERROR; FOR TESTING THE ABOVE PREDICTOR
39 aCoeff=a((1:s),s)';           //array of "a(i,s)" , where
   , s=M+1
40 tcount_of_aCoeff = length(aCoeff);
41
42 y_padded_for_delay_r = [y'; zeros(1,1)]; //it is

```

```

        padded with zeros to remove the effect of delay
        in filter
43 est_y_with_dummy_pad = filter([0 -aCoeff(2:9)],1,
        y_padded_for_delay_r); // = s^(n) with a cap
        on page 92 of the book
44 est_y = est_y_with_dummy_pad(2:321);
45 e = y' - est_y; //supposed to be a white noise
46 endfunction
47
48 function [aCoeff, b_LTopt, Topt, e_prime] =
    f_ENCODER_relp(x, fs)
49 M = 8; //prediction order for LP analysis
50 //INITIALIZATION;
51 b=1; //index no. of starting data point of
        current frame
52 fsize = 20e-3; //frame size (in milisec)
53 frame_length = round(fs .* fsize); //=number data
        points in each framesize of "x"
54 N= frame_length - 1; //N+1 = frame length = number
        of data points in each framesize
55
56 y_proc = filter([1 -1], [1 -0.999], x); //pre-
        processing
57 //FRAME SEGMENTATION
58 for b=1 : frame_length : (length(x) - N)
59
60 y_f = y_proc(b:b+N); // "b+N" denotes the end
        point of current frame. "y" denotes an array of
        the data points of the current frame
61 //LP ANALYSIS [lev-durb] & PREDICTION ERROR (short-
        term) FILTER;
62 [a, tcount_of_aCoeff, e_s] = func_lev_durb(y_f,
        M); //e=error signal from lev-durb proc
63 aCoeff(b: (b + tcount_of_aCoeff - 1)) = a; // //aCoeff is array of "a" for whole "x"
64 //LONG-TERM LP ANALYSIS, FILTERING, AND CODING
        analysis:
65 T_min = round (fs .* 5e-3); //=total data

```

```

    points in 5ms of "x"
66     T_max = round (fs .* 15e-3);
67     c1 = 1;
68     for bs = b : 40 : b+length(y_f)-40 //subframing
69         bs = 1281;
70         if bs < T_max
71             break;
72         end
73
74         Jmin(bs) = 10^9;
75
76         for T = T_min : T_max //within 1 (
77             current) frame T = 40;
78             for c = 1:40 //data points of
79                 current subframe c=1; temporary
80                 sm1(c) = ( y_proc(bs+(c-1)) .* 
81                             y_proc(bs-T+(c-1))); //es(n)
82                 sm2(c) = y_proc(bs-T+(c-1)); //=
83                             es(n-T)
84                 sm22(c) = sm2(c).^2;
85             end
86             q1 = sum(sm1);
87             q2 = sum(sm22);
88             b_LT(T) = -(q1 ./ q2);
89             //J loop:
90             for c = 1:40 //data points of
91                 current subframe c=1; temporary
92                 smJ1(c) = y_proc(bs+(c-1));
93                 smJ2(c) = b_LT(T) .* y_proc(bs-T
94                             +(c-1));
95             end
96             smJ = smJ1 + smJ2;
97             qJ = smJ.^2;
98             J(T) = sum(qJ);
99
100            if J(T) < Jmin(bs),
101                Jmin(bs) = J(T);
102                Topt(bs) = T;

```

```

96             if b_LT(T)>=1,
97                 b_LTopt(bs) = 0.9999; // 
98                     truncation
99             else
100                b_LTopt(bs) = b_LT(T);
101            end
102        else
103    end      //T loop ends
104 //predictor:
105 LT_gain = [zeros(1, Topt(bs)-1), b_LTopt(bs)
106     ]; //as it says z^-T in page 121
107 e_s_padded_for_delay_r = [e_s(c1:c1+39);
108     zeros(Topt(bs), 1)]; //it is padded with
109     zeros to remove the effect of delay in
110     filter. %Topt(bs) no. of 'z's and one
111     '1' results in total 'Topt(bs)' amount
112     of delay
113 e_with_dummy_pad = filter([1 LT_gain], 1,
114     e_s_padded_for_delay_r); // = 1 + 0*z
115     ^-1 + 0*z^-2 + ... + b*z^-T
116 e_LT(bs:bs+39,1) = e_with_dummy_pad(Topt(bs
117     )+1 : Topt(bs)+1+39); //LT predicted "
118     e"
119 e(bs:bs+39, 1) = e_s(c1 : c1+39) - e_LT(bs :
120     bs+39);
121
122 //WEIGHTING FILTER:
123 w = [-0.0004;
124     -0.0156; -0.0677; 0.0545; 0.6069; 1.0000; 0.6069; 0.0545; -0.0677;
125     //11 point flattop window is
126     temporarily chosen
127 wndd = conv(w, e(bs:bs+39)); //outputs
128     total 50 samples
129 x_n(bs:bs+39) = wndd(6:45); //middle 40
130     samples are taken
131
132 //POSITION SELECTION & EXCITATION GENERATOR:

```

```

117     for i1 = 0:3
118         for i = i1+bs : 3 : bs+i1+38;
119             x_m(i1+1,i) = x_n(i);
120         end
121
122         E_m(i1+1,1) = sum((x_m(i1+1, bs:3)).^2);
123     end
124     [E_m_max, index_max] = gsort(E_m);
125     e_prime(bs : bs+39) = x_m(index_max(4), bs:
126                                bs+39);
126     c1 = c1 + 40;
127 end
128 end
129 endfunction
130
131 //RELP DECODER portion:
132 function [synth_speech, synth_speech1, LT_gain,
133   e_prime_pad_for_d_r, e_prime_op_dummy_pad,
134   e_prime_op, e_prime_op_pad_delay_r,
135   synth_speech_dummy_pad] = f_DECODER_relp(aCoeff,
136   b_LTopt, Topt, e_prime)
136 //re-calculating frame_length for this decoder
137 frame_length=9; //initial value for calculation
138 for i=10:length(aCoeff)
139     if aCoeff(i) == 0
140         frame_length = frame_length + 1;
141     else break;
142     end
143 e_prime = e_prime'; //making it a column matrix
144 //for convenience
145 for b=1 : frame_length : length(aCoeff) //length(
146 aCoeff) should be very close (i.e less than a
147 frame_length error) to length(x)
148     for bs = b : 40 : b+frame_length-40 ///
149         subframing

```

```

146 //EXCITATION GENERATOR: not done yet.
147 //because e_prime has been sent to this
148 //decoder directly. without quantization.
149 //PITCH SYNTHESIS FILTER: %has to be done
150 //per subframe
151 LT_gain = [zeros(1, Topt(bs)-1), b_LTopt(bs)
152 ]; //as it says z^-T
153 e_prime_pad_for_d_r = [e_prime(bs:bs+39);
154 zeros(Topt(bs), 1)]; //it is padded with
155 //zeros to remove the effect of delay in
156 //filter. %Topt(bs) no. of 'z's and one
157 // '1' results in total 'Topt(bs)' amount
158 //of delay
159 e_prime_op_dummy_pad = filter(1, [1 LT_gain
160 ], e_prime_pad_for_d_r); //= 1 / (1 +
161 // 0*z^-1 + 0*z^-2 + ... + b*z^-T)
162 e_prime_op(bs:bs+39,1) =
163 e_prime_op_dummy_pad(Topt(bs)+1 : Topt(bs
164 )+1+39); //pitch-synthesis filter
165 output
166 end //FORMANT SYNTHESIS FILTER:
167 e_prime_op_pad_delay_r= [e_prime_op(b : b
168 +159); zeros(1,1)]; //it is padded with
169 //zeros to remove the effect of delay in
170 //filter
171 synth_speech_dummy_pad = filter(1, [1 aCoeff
172 (b+1 : b+8)], e_prime_op_pad_delay_r);
173 synth_speech1(b : b+159) =
174 synth_speech_dummy_pad (2:161); //DE-
175 //EMPHASIS (de-propocessing):
176 synth_speech(b : b+159) = filter([1 -0.999],
177 [1 -1], synth_speech1(b : b+159)); //De-
178 //processing
179 end
180 endfunction
181
182 clc;
183 clear all;

```

```

162 xdel(winsid());
163 inpfilenm = "SCI/modules/sound/demos/s1ofwb.wav";
164 [x,fs,bits] = wavread(inpfilenm);
165
166 t=length(x)/fs; // total time t seconds
167 //COMPRESSION STARTS HERE,
168 disp('original signal');
169 sound(x, fs);
170 [aCoeff, b_LTopt, Topt, e_prime] = f_ENCODER_relp(x,
171 fs);
172
173 //halt();
174 //halt('Press a key to play the original sound!')
175
176 [synth_speech] = f_DECODER_relp(aCoeff, b_LTopt,
177 Topt, e_prime);
178 //RESULTS,
179
180
181 disp('compressed signal');
182 sound(synth_speech, fs);
183
184 figure;
185 subplot(211),
186 plot(x); title(['Original signal = "', inpfilenm, '"']);
187 subplot(212), plot(synth_speech); title('RELP
188 compressed output');
189 //Output plays original signal and after
190 //approximately 5 minutes it plays compressed sound
191 //and plot the original signal and compressed
192 //signal.

```
