Scilab Textbook Companion for Digital Control by K. M. Moudgalya[1]

Created by
Inderpreet Arora
Digital Control
Chemical Engineering
IIT Bombay
College Teacher
NA
Cross-Checked by
Chaitanya

May 28, 2016

[1]Funded 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 Control

Author: K. M. Moudgalya

Publisher: Wiley, India

Edition: 1

Year: 2009

ISBN: 978-81-265-2206-4

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

List of Figures

Chapter 2

Modelling of Sampled Data Systems

Scilab code Exa 2.1 Model of inverted pendulum

```
1  // Model of inverted pendulum
2  // 2.1
3
4  Km = 0.00767;
5  Kg = 3.7;
6  Rm = 2.6;
7  r = 0.00635;
8  M = 0.522;
9  m = 0.231;
10 g = 9.81;
11 L = 0.305;
12 J = 0;
13
14 D1 = (J+m*L^2)*(M+m)-m^2*L^2;
15 alpha = m*g*L*(M+m)/D1;
16 beta1 = m*L/D1;
17 gamma1 = m^2*g*L^2/D1;
18 delta = (J+m*L^2)/D1;
19
```

```
20 alpha1 = Km*Kg/Rm/r;
21 alpha2 = Km^2*Kg^2/Rm/r^2;
22
23 A = zeros(4,4);
24 A(1,3) = 1;
25 A(2,4) = 1;
26 A(3,2) = -gamma1;
27 A(3,3) = -alpha2*delta;
28 A(4,2) = alpha;
29 A(4,3) = alpha2*beta1;
30
31 B = zeros(4,1);
32 B(3) = alpha1*delta;
33 B(4) = -alpha1*beta1;
```

Scilab code Exa 2.2 Exponential of the matrix

```
1 // Exponential of the matrix
2 // 2.2
3
4 F = [-1 0;1 0];
5 expm(F)
```

Scilab code Exa 2.3 ZOH equivalent state space system

```
1 // ZOH equivalent state space system
2 // 2.3
3
4 F = [-1 0;1 0]; G = [1; 0];
5 C = [0 1]; D = 0; Ts=1;
6 sys = syslin('c',F,G,C,D);
7 sysd = dscr(sys,Ts)
```

Chapter 3

Linear Systems

Scilab code Exa 3.1 Energy of a signal

```
1 // Energy of a signal
2 // 3.1
3
4 u = [4 5 6];
5 Eu = norm(u)^2;
6 ruu = xcorr(u);
7 Lu = length(ruu);
8 Eu = ruu(ceil(Lu/2));
```

Scilab code Exa 3.2 Convolution of two sequences

```
1 // Convolution of two sequences
2 // 3.2
3
4 h = [1 2 3];
5 u = [4 5 6];
6 y = convol(u,h)
```

Chapter 4

Z Transform

Scilab code Exa 4.1 To produce a sequence

```
1  // To produce a^n 1(n)
2  // 4.1
3
4  exec('stem.sci',-1);
5  exec('label.sci',-1);
6
7  a = 0.9;
8  n = -10:20;
9  y = zeros(1,size(n,'*'));
10 for i = 1:length(n)
11     if n(i)>=0,
12        y(i) = a^n(i);
13     end
14 end
15 stem(n,y)
16 label('u1',4,'Time(n)','0.9^n1(n)',4)
```

Scilab code Exa 4.2 To produce a sequence

```
// Plot of -0.9^n1(-n-1)
// 4.2

exec('stem.sci',-1);
exec('label.sci',-1);

a = 0.9;
n = -10:20;
y = zeros(1,size(n,'*'));
for i = 1: length(n)
    if n(i)<= -1,
        y(i) = -(a^n(i));
        else y(i) = 0;
    end
end
stem(n,y)
label('u2',4,'Time(n)','-0.9^n1(-n-1)',4)
```

Scilab code Exa 4.3 To produce pole zero plots

```
// To produce pole-zero plots
// 4.3

exec('label.sci',-1);

zero = [0 5/12];
num = poly(zero,'z',"roots");
pole = [1/2 1/3];
den = poly(pole,'z',"roots");
h = syslin('d',num./den);
plzr(h);

label('Pole-Zero Plot',4,'Real(z)','Imaginary(z)',4)
   ;
```

Scilab code Exa 4.4 Discrete transfer function of the continuous state space system

```
1 // Discrete transfer function of the continuous
     state space system
2 // 4.4
3
4 F = [0 0; 1 -0.1]; G = [0.1; 0];
5 C = [0 1]; dt = 0.2;
6 sys = syslin('c',F,G,C);
7 sysd = dscr(sys,dt);
8 H = ss2tf(sysd);
```

Scilab code Exa 4.5 Computation of residues

```
1 // Computation of residues
2 // 4.5
3 // Numerator and denominator coefficients
4 // are passed in decreasing powers of z(say)
5
6 function [res,pol,q] = respol(num,den)
7 len = length(num);
8 if num(len) == 0
9     num = num(1:len-1);
10 end
11
12 [resi,q] = pfe(num,den);
13 res = resi(:,2);
14 res = int(res) + (clean(res - int(res),1.d-04));
15 pol = resi(:,1);
16 pol = int(pol) + (clean(pol - int(pol),1.d-04));
17 endfunction;
```

```
18
19 ////////////////////////////////////////////////////
20 // Partial fraction expansion
21
22 function [resid1,q] = pfe(num,den)
23 x = poly(0,'x');
24 s = %s;
25
26 num2 = flip(num);
27 den2 = flip(den);
28 num = poly(num2,'s','coeff');
29 den = poly(den2,'s','coeff');
30 [fac,g] = factors(den);
31 polf = polfact(den);
32 n = 1;
33
34 r = clean(real(roots(den)),1.d-5);
35 //disp(r);
36 l = length(r);
37 r = gsort(r,'g','i');
38 r = [r; 0];
39 j = 1;
40 t1 = 1; q = [];
41 rr = 0;
42 rr1 = [0 0];
43 m = 1;
44
45   for i = j:l
46      if abs(r(i)- r(i+1)) < 0.01 then
47        r(i);
48        r(i+1);
49        n = n+1;
50        m = n;
51        //pause
52        t1 = i;
53        //disp('Repeated roots')
54      else
55        m = n;
```

```
56          //pause
57          n = 1;
58        end
59        i;
60        if n == 1 then
61          rr1 = [rr1; r(i) m];
62          //pause
63        end;
64          j = t1 + 1;
65    end;
66  rr2 = rr1(2:$,:);
67  [r1,c1] = size(rr2);
68  den1 = 1;
69
70  for i = 1:r1
71    den1 = den1 * ((s-rr2(i,1))^(rr2(i,2)));
72  end;
73  [rem,quo] = pdiv(num,den);
74  q = quo;
75  if quo ~= 0
76    num = rem;
77  end
78
79  tf = num/den1;
80  res1 = 0;
81  res3 = [s 0];
82  res5 = [0 0];
83  for i = 1:r1
84    j = rr2(i,2) + 1;
85    tf1 = tf; //strictly proper
86    k = rr2(i,2);
87    tf2 = ((s-rr2(i,1))^k)*tf1;
88    rr2(i,1);
89    res1 = horner(tf2,rr2(i,1));
90    res2 = [(s - rr2(i,1))^(rr2(i,2)) res1];
91    res4 = [rr2(i,1) res1];
92    res3 = [res3; res2];
93    res5 = [res5; res4];
```

```
94    res = res1;
95      for m = 2:j-1
96      k;
97      rr2(i,1);
98        tf1 = derivat(tf2)/factorial(m-1); //ith
             derivative
99        res = horner(tf1,rr2(i,1));
100       res2 = [(s - rr2(i,1))^(j-m) res];
101       res4 = [rr2(i,1) res];
102       res5 = [res5; res4];
103       res3 = [res3; res2];
104       tf2 = tf1;
105     end;
106 end;
107 resid = res3(2:$,:); //with s terms
108 resid1 = res5(2:$,:); //only poles(in decreasing no.
      of repetitions)
109 endfunction;
110 //
    ////////////////////////////////////////////////////////
```

Scilab code Exa 4.6 Partial fraction expansion

```
1  // Partial fraction expansion for Example 4.24
2  // 4.6
3
4  //              2z^2 + 2z
5  // G(z)  =  ---------------
6  //             z^2 + 2z - 3
7
8  exec('respol.sci',-1);
9  exec('flip.sci',-1);
10
11 num = [2 2 0];
```

```
12 den = [1 2 -3];
13 [res,pol] = respol(num,den) //respol is a user
      defined function
```

Scilab code Exa 4.7 Partial fraction expansion

```
1  // Partial fraction expansion for Example 4.26
2  // 4.7
3
4  //              z^2 + z          A          B
               C
5  //  G(z) = ------------  =  ------- + --------- +
   ----------
6  //           (z - 1)^3       (z - 1)   (z - 1)^2   (z
       - 1)^3
7
8  exec('respol.sci',-1);
9  exec('flip.sci',-1);
10
11 num = [1 1 0];
12 den = convol([1 -1],convol([1 -1],[1 -1])); // poly
      multiplication
13 [res,pol] = respol(num,den)
14
15 // Output interpretation
16 // res  =
17 //C = 2
18 //B = 1
19 //A = 0
20
21 // pol  =
22 // 1       (z - 1)^3
23 // 1       (z - 1)^2
24 // 1       (z - 1)
```

Scilab code Exa 4.8 Partial fraction expansion

```
1 // Partial fraction expansion for Example 4.27
2 // 4.8
3
4 //          11z^2 - 15z + 6       A1          A2
          B
5 // G(z) = ------------------ = ------- + --------- +
   --------
6 //         (z - 2) (z - 1)^2    (z - 1)    (z - 1)^2
   (z - 2)
7
8 exec('respol.sci',-1);
9 exec('flip.sci',-1);
10
11 num = [11 -15 6];
12 den = convol([1 -2],convol([1 -1],[1 -1]));
13 [res,pol] = respol(num,den) //User defined function
```

Scilab code Exa 4.9 Partial fraction expansion

```
1 // Partial fraction expansion for Example 4.29
2 // 4.9
3
4 //                  z^2 + 2z
5 // G(z)  =  ---------------------
6 //           (z + 1)^2 (z - 2)
7
8 exec('respol.sci',-1);
9 exec('flip.sci',-1);
10
11 num = [1 2 0];
```

```
12 den = convol(convol([1 1],[1 1]),[1 -2]);
13 [res,pol] = respol(num,den)
```

Scilab code Exa 4.10 Partial fraction expansion

```
1 // Partial fraction expansion for Example 4.30
2 // 4.10
3
4 //                  3 - (5/6)z^-1                    3z^2 - (5/6)z
5 // G(z) = ------------------------------ = ------------------------------
  ----------
6 //          (1-(1/4)z^-1)(1-(1/3)z^-1)       (z - (1/4))(z - (1/3))
7
8 // No equivalent of residuez
9
10 exec('respol.sci',-1);
11 exec('flip.sci',-1);
12
13 num = [3 -5/6 0];
14 den = convol([1 -1/4],[1 -1/3]);
15 [res,pol,q] = respol(num,den)
```

Scilab code Exa 4.11 Long division of problems

```
1 // Long division of problems discussed in Example
     4.32 on page 102
2 // 4.11
3
4 exec('tf.sci',-1);
5 exec('label.sci',-1);
6
```

```
num = [11 -15 6];
den = convol([1 -2], convol([1 -1],[1 -1]));
u = [1 zeros(1,4)];
y = filter(num,den,u);
G = tf(num,den,-1);
u1=zeros(1,90); u1(1)=1;
x1=dsimul(tf2ss(G),u1);
plot2d(x1)
label('Impulse Response',4,'Time(seconds)','
   Amplitude',4)
```

Chapter 5

Frequency Domain Analysis

Scilab code Exa 5.1 Sinusoidal plots for increasing frequency

```
1 // Sinusoidal plots for increasing frequency
2 // 5.1
3
4 exec('stem.sci',-1);
5
6 n=0:16;
7 subplot(2,2,1), stem(n,cos(n*%pi/8))
8 xgrid,xtitle('','n','cos(n*pi/8)')
9 subplot(2,2,2), stem(n,cos(n*%pi/4))
10 xgrid,xtitle('','n','cos(n*pi/4)')
11 subplot(2,2,3), stem(n,cos(n*%pi/2))
12 xgrid,xtitle('','n','cos(n*pi/2)')
13 subplot(2,2,4), stem(n,cos(n*%pi))
14 xgrid,xtitle('','n','cos(n*pi)')
```

Scilab code Exa 5.2 Bode plots

```
1 // Bode plots for Example 5.7 on page 141
```

```
// 5.2

exec('label.sci',-1);

omega = linspace(0,%pi);
g1 = 0.5 ./ (cos(omega)-0.5+%i*sin(omega));
mag1 = abs(g1);
angle1 = phasemag(g1);
g2 = (0.5+0.5*cos(omega)-1.5*%i*sin(omega)) ...
   * 0.25 ./ (1.25-cos(omega));
mag2 = abs(g2);
angle2 = phasemag(g2);
subplot(2,1,1)
plot(omega,mag1,omega,mag2,'--');
label('',4,' ','Magnitude',4);
subplot(2,1,2);
plot(omega,angle1,omega,angle2,'--');
label('',4,'w (rad/s)','Phase',4);
```

Scilab code Exa 5.3 Bode plot of the moving average filter

```
// Bode plot of the moving average filter, discussed
    in Example 5.5 on page 129
// 5.3

exec('label.sci',-1);

w = 0.01:0.01:%pi;
subplot(2,1,1);
mag = abs(1+2*cos(w))/3;
plot2d("ll",w,mag,2);
label('',4,' ','Magnitude',4);
subplot(2,1,2);
plot2d("ln",w,phasemag(1+2*cos(w)),style = 2,rect
   =[0.01 -0.5 10 200]);
```

```
13 label('',4,'w','Phase',4)
```

Scilab code Exa 5.4 Bode plot of the differencing filter

```
1  // Bode plot of the differencing filter, discussed
   in Example 5.6 on page 130
2  // 5.4
3
4  exec('label.sci',-1);
5
6  w = 0.01:0.01:%pi;
7  G = 1-exp(-%i*w);
8  subplot(2,1,1)
9  plot2d1("gll",w,abs(G),style = 2);
10 label('',4,' ','Magnitude',4);
11 subplot(2,1,2)
12 plot2d1("gln",w,phasemag(G),style = 2);
13 label('',4,'w','Phase',4)
```

Scilab code Exa 5.5 Bode plot of minimum and nonminimum phase filters

```
1  // Bode plot of minimum and nonminimum phase filters
   , discussed in Example 5.9 on page 145
2  // 5.5
3
4  exec('label.sci',-1);
5
6  omega = linspace(0,%pi);
7  ejw = exp(-%i*omega);
8  G1 = 1.5*(1-0.4*ejw);
9  mag1 = abs(G1); angle1 = phasemag(G1);
10 G2 = -0.6*(1-2.5*ejw);
11 mag2 = abs(G2); angle2 = phasemag(G2);
```

```
12 subplot(2,1,1);
13 plot(omega,mag1,omega,mag2,'--');
14 label('',4,' ','Magnitude',4);
15 subplot(2,1,2);
16 plot(omega,angle1,omega,angle2,'--');
17 label('',4,'w (rad/s)','Phase',4);
```

Chapter 6

Identification

Scilab code Exa 6.1 Least squares solution

```
1 // Least squares solution of the simple problem
     discussed in Example 6.4 on page 164
2 // 6.1
3
4 Mag = 10; V = 10; No_pts = 100; theta = 2;
5 Phi = Mag * (1-2*rand(No_pts,1));
6 E = V * (1-2*rand(No_pts,1));
7 Z = Phi*theta + E;
8 LS = Phi \ Z
9 Max = max(Z ./ Phi), Min = min(Z ./ Phi)
```

Scilab code Exa 6.2 ACF calculation

```
1 // ACF calculation for the problem discussed in
     Example 6.5 on page 167
2 // 6.2
3
4 u = [1 2];
```

```
r = xcov(u);
rho = xcov(u,"coeff");
```

Scilab code Exa 6.3 To demonstrate the periodicity property of ACF

```
// To demonstrate the periodicity property of ACF as
    discussed in Example 6.7 on page 173
// 6.3

exec('plotacf.sci',-1);
exec('label.sci',-1);

L = 500;
n = 1:L;
w = 0.1;
S = sin(w*n);
m = 1;
xi = m*rand(L,1,'normal');
Spxi = S+xi';
xset('window',0);
plot(Spxi);
label('',4,'n','y',4)
xset('window',1);
plotacf(Spxi,1,L,1);
```

Scilab code Exa 6.4 To demonstrate the maximum property of ACF at zero lag

```
// To demonstrate the maximum property of ACF at
   zero lag, as discussed in Example 6.8 on page
   175.
// 6.4

```

```
exec('label.sci',-1);

S1 = [1 2 3 4];
S2 = [1,-2,3,-4];
S3 = [-1,-2,3,4];
len = length(S1)-1;
xv = -len:len;
m = 1;
xi = rand(4,1,'normal');
Spxi1 = S1 + m*xi';
Spxi2 = S2 + m*xi';
Spxi3 = S3 + m*xi';
n = 1:length(S1);
plot(n,Spxi1,'o-',n,Spxi2,'x--',n,Spxi3,'*:');
label('',4,'n','y',4);
ACF1 = xcov(Spxi1,"coeff");
ACF2 = xcov(Spxi2,"coeff");
ACF3 = xcov(Spxi3,"coeff");
xset('window',1);
a = gca();
a.data_bounds = [-len -1; len 1];
plot(xv,ACF1,'o-',xv,ACF2,'x--',xv,ACF3,'*:');
label('',4,'Lag','ACF',4);
```

Scilab code Exa 6.5 Demonstrate the order of MA

```
// Demonstrate the order of MA(q) as discussed in
   Example 6.11 on page 182.
// 6.5

exec('plotacf.sci',-1);
exec('label.sci',-1);

xi = 0.1*rand(1,10000,'normal');
a = 1; b = [];
```

```
d = [1 1 -0.5];
ar = armac(a,b,d,1,1,1);
v = arsimul(ar,xi);
z = [v xi];

// Plot noise, plant output and ACF
subplot(2,1,1), plot(v(1:500))
label('',4,'','v',4)
subplot(2,1,2), plot(xi(1:500))
label('',4,'n','xi',4)
xset('window',1)
plotacf(v,1,11,1);
```

Scilab code Exa 6.6 Procedure to plot the ACF

```
// Procedure to plot the ACF, as discussed in Sec.
   6.4.3. An example usage is given in 6.5.
// 6.6

// PLOTACF: Plots normalized autocorrelation
   function
//
// USAGE:: [acf]=plotacf(x,errlim,len,print_code)
//
// WHERE:: acf = autocorrelation values
// x = time series data
// errlim > 0; error limit = 2/sqrt(data_len)
// len = length of acf that need to to be plotted
// NOTE: if len=0 then len=data_length/2;
// print_code = 0 ==> does not plot OR ELSE plots
//
// Pranob Banerjee

function [x]=plotacf(y,errlim,len,code)
exec('label.sci',-1)
```

```
x = xcov(y); l = length(y); x = x/x(l);
r=l:2*(l-1); lim=2/sqrt(l); rl=1:length(r) ;
N=length(rl); x=x(r);
if len>0 & len<N, rl=1:len; x=x(rl); N=len; end;
if(code > 0 )
  if(errlim > 0 )
    rl=rl-1;
    plot(rl,x,rl,x,'o' , rl,lim*ones(N,1),'--', ...
         rl,-lim*ones(N,1),'--')
    xgrid
  else
    plot(rl,x)
  end
end;
a = gca();
a.data_bounds = [0 min(min(x),-lim-0.1); len-1 1.1];
label(' ',4,'Lag','ACF',4)
endfunction;
```

Scilab code Exa 6.7 Illustration of nonuniqueness in estimation of MA model parameters using ACF

```
// Illustration of nonuniqueness in estimation of MA
    model parameters using ACF, discussed in Example
    6.14 on page 184
// 6.7

exec('plotacf.sci',-1);
exec('pacf.sci',-1);
exec('label.sci',-1);

xi = 0.1*rand(1,10000);

// Simulation and estimation of first model
m1 = armac(1,0,[1,-3,1.25],1,1,1);
```

```
12 v1 = arsimul(m1,xi);
13 M1 = armax1(0,0,2,v1,zeros(1,10000))
14 disp(M1)
15
16 // Simulation and estimation of second model
17 m2 = armac(1,0,[1,-0.9,0.2],1,1,1);
18 v2 = arsimul(m2,xi);
19 M2 = armax1(0,0,2,v2,zeros(1,10000))
20 disp(M2)
21
22 // ACF and PACF of both models
23 plotacf(v1,1,11,1);
24 xset('window',1), plotacf(v2,1,11,1);
25 xset('window',2), pacf(v1,11);
26 xset('window',3), pacf(v2,11);
```

Scilab code Exa 6.8 Estimation with a larger order model results in large uncertainty

```
1 // Estimation with a larger order model results in
    large uncertainty, as discussed in Example 6.15
    on page 185.
2 // 6.8
3
4 m = armac(1,0,[1 -0.9 0.2],1,1,1);
5 xi = 0.1*rand(1,10000);
6 v = arsimul(m,xi);
7 M1 = armax1(0,0,2,v,zeros(1,10000))
8 disp(M1)
9 M2 = armax1(0,0,3,v,zeros(1,10000))
10 disp(M2)
```

Scilab code Exa 6.9 Determination of order of AR process

```
1  // Determination of order of AR(p) process, as
       discussed in Example 6.18 on page 189.
2  // 6.9
3
4  exec('pacf.sci',-1);
5  exec('label.sci',-1);
6
7  // Define model and generate data
8  m = armac([1,-1,0.5],0,1,1,1,1);
9  xi = 0.1*rand(1,10000,'normal');
10 v = arsimul(m,xi);
11
12 // Plot noise, plant output and PACF
13 subplot(2,1,1), plot(v(1:500));
14 label('',6,'','v',6);
15 subplot(2,1,2), plot(xi(1:500));
16 label('',6,'n','xi',6);
17 xset('window',1)
18 pacf(v,10);
```

Scilab code Exa 6.10 Determination of the PACF of AR process

```
1  // Determination of the PACF of AR(p) process, as
       explained in Sec. 6.4.5.
2  // 6.10
3
4  function [ajj] = pacf(v,M)
5  exec('label.sci',-1);
6  rvvn = xcorr(v,'coeff');
7  len = length(rvvn);
8  zero = (len+1)/2;
9  rvvn0 = rvvn(zero);
10 rvvn_one_side = rvvn(zero+1:len);
11 ajj = [];
12 exec('pacf_mat.sci',-1);
```

```
for j = 1:M,
  ajj = [ajj pacf_mat(rvvn0,rvvn_one_side,j,1)];
end
p = 1:length(ajj);
N = length(p);
lim = 2/sqrt(length(v));

// Plot the figure
plot(p,ajj,p,ajj,'o',p,lim*ones(N,1),'--',...
                 p,-lim*ones(N,1),'--');
label('',4,'Lag','PACF',4);
endfunction;
```

Scilab code Exa 6.11 Construction of square matrix required to compute PACF ajj

```
// Construction of square matrix required to compute
    PACF ajj, useful for the calculations in Sec.
   6.4.5.
// 6.11

function ajj = pacf_mat(rvv0,rvv_rest,p,k)
if argn(2) == 3,
  k = 1;
end
for i = 1:p
  for j = 1:p
    index = (k+i-1)-j;
    if index == 0,
      A(i,j) = rvv0;
    elseif index < 0,
      A(i,j) = rvv_rest(-index);
    else
      A(i,j) = rvv_rest(index);
    end
```

```
18    end
19    b(i) = -rvv_rest(k+i-1);
20  end
21  a = A\b;
22  ajj = a(p);
23  endfunction;
```

Scilab code Exa 6.12 PACF plot of an MA process decays slowly

```
1  // PACF plot of an MA process decays slowly, as
       discussed in Example 6.19 on page 190.
2  // 6.12
3
4  exec('plotacf.sci',-1);
5  exec('pacf.sci',-1);
6  exec('label.sci',-1);
7
8  m = armac(1,0,[1,-0.9,0.2],1,1,1);
9  xi = 0.1*rand(1,10000);
10 v = arsimul(m,xi);
11 plotacf(v,1,11,1);
12 xset('window',1);
13 pacf(v,11);
```

Scilab code Exa 6.13 Implementation of trial and error procedure to determine ARMA process

```
1  // Implementation of trial and error procedure to
       determine ARMA(1,1) process, presented in Example
        6.20 on page 191.
2  // 6.13
3
4  exec('plotacf.sci',-1);
```

```
5  exec('pacf.sci',-1);
6  exec('label.sci',-1);
7
8  // Set up the model for simulation
9  arma_mod = armac([1 -0.8],0,[1 -0.3],1,1,1);
10
11 // Generate the inputs for simulation
12 // Deterministic Input can be anything
13 u = zeros(1,2048);
14 e = rand(1,2048,'normal');
15
16 // Simulate the model
17 v = arsimul(arma_mod,[u e]);
18
19 // Plot ACF and PACF for 10 lags
20 plotacf(v,1e-03,11,1);
21 xset('window',1), pacf(v,10);
22
23 // Estimate AR(1) model and present it
24 // compute the residuals
25 [mod_est1,err_mod1] = armax1(1,0,0,v,zeros(1,length(
     v)));
26 disp(mod_est1);
27
28 // Plot ACF and PACF for 10 lags
29 xset('window',2), plotacf(err_mod1,1e-03,11,1);
30 xset('window',3), pacf(err_mod1,10);
31
32 // Check ACF and PACF of residuals
33 [mod_est2,err_mod2] = armax1(1,0,1,v,zeros(1,length(
     v)));
34 disp(mod_est2);
35
36 // Plot ACF and PACF for 10 lags
37 xset('window',4), plotacf(err_mod2,1e-03,11,1);
38 xset('window',5), pacf(err_mod2,10);
```

Scilab code Exa 6.14 Determination of FIR parameters

```
1  // Determination of FIR parameters as described in
       Example 6.22 on page 200.
2  // 6.14
3
4  exec('cra.sci',-1);
5  exec('filt.sci',-1);
6  exec('covf.sci',-1);
7
8  sig = 0.05;
9  process_mod = armac([1 -0.5],[0 0.6 -0.2],1,1,1,sig)
       ;
10
11 u = prbs_a(2225,40);
12 xi = rand(1,2225,'normal');
13 y = arsimul(process_mod,[u xi]);
14 u = [u zeros(1,length(y)-length(u))];
15 z = [y' u'];
16
17 // Plot y as a function of u and xi
18 exec('label.sci',-1)
19 subplot(3,1,1), plot(y(1:500)),
20 label('',4,'','y',4)
21 subplot(3,1,2), plot(u(1:500))
22 label('',4,'','u',4)
23 subplot(3,1,3), plot(sig*xi(1:500))
24 label('',4,'n','xi',4)
25
26 xset('window',1);
27 [ir,r,cl_s] = cra(detrend(z,'constant'));
28 ir_act = filt([0 0.6 -0.2],[1 -0.5],...
29                [1 zeros(1,9)]);
30 replot([0,min(min(ir),min(ir_act)) - 0.1,9,max(max(
```

```
    ir),max(ir_act)) + 0.1]);
31 t = 0:9;
32 plot(t,ir_act,'ko');
33 plot2d3(t,ir_act);
34 legends(['Estimated'; 'Actual'], [2;-9],'ur');
```

Scilab code Exa 6.15 Determination of ARX parameters

```
1  // Determination of ARX parameters as described in
     Example 6.25 on page 203.
2  // 6.15
3
4  exec('armac1.sci',-1);
5  exec('cra.sci',-1);
6  exec('arx.sci',-1);
7  exec('filt.sci',-1);
8  exec('covf.sci',-1);
9  exec('stem.sci',-1);
10
11 process_arx = armac1([1 -0.5],[0 0 0.6
     -0.2],1,1,1,0.05);
12 u = prbs_a(5000,250);
13 xi = rand(1,5000,'normal');
14 y = arsimul(process_arx,[u xi]);
15 z = [y(1:length(u))' u'];
16 zd = detrend(z,'constant');
17
18 // Compute IR for time-delay estimation
19 [ir,r,cl_s] = cra(detrend(z,'constant'));
20
21 // Time-delay = 2 samples
22 // Estimate ARX model (assume known orders)
23 na = 1; nb = 2; nk = 2;
24 [theta_arx,cov_arx,nvar,resid] = arx(zd,na,nb,nk);
25
```

```
26 // Residual plot
27 [cov1,m1] = xcov(resid,24,"coeff");
28 xset('window',1);
29 subplot(2,1,1)
30 stem(0:24,cov1(25:49)');xgrid();
31 xtitle('Correlation function of residuals from
      output y1','lag');
32 [cov2,m2] = xcov(resid, zd(:,2),24,"coeff");
33 subplot(2,1,2)
34 stem(-24:24,cov2');xgrid();
35 xtitle('Cross corr. function between input u1 and
      residuals from output y1','lag');
```

Scilab code Exa 6.16 Determination of ARMAX parameters

```
1 // Determination of ARMAX parameters as described in
     Example 6.27 on page 206.
2 // 6.16
3
4 exec('cra.sci',-1);
5 exec('stem.sci',-1);
6 exec('filt.sci',-1);
7 exec('covf.sci',-1);
8
9 process_armax = armac([1 -0.5],[0 0 0.6 -0.2],[1
     -0.3],1,1,0.05);
10 u = prbs_a(5000,250);
11 xi = rand(1,5000);
12 y = arsimul(process_armax,[u xi]);
13 z = [y(1:length(u))' u'];
14 zd = detrend(z,'constant');
15
16 //Compute IR for time-delay estimation
17 [ir,r,cl_s] = cra(detrend(z,'constant'));
18
```

```
19 //Estimate ARMAX model (assume known orders)
20 na = 1; nb = 3; nc = 1; nk = 2;
21 [theta_armax,resid] = armax1(na,nb,nc,zd(:,1)',zd
      (:,2)',1);
22 disp(theta_armax)
23
24 // Residual plot
25 [cov1,m1] = xcov(resid,24,"coeff");
26 xset('window',1);
27 subplot(2,1,1)
28 stem(0:24,cov1(25:49));xgrid();
29 xtitle('Correlation function of residuals from
      output y1','lag');
30 [cov2,m2] = xcov(resid, zd(:,2),24,"coeff");
31 subplot(2,1,2)
32 stem(-24:24,cov2);xgrid();
33 xtitle('Cross corr. function between input u1 and
      residuals from output y1','lag');
```

Scilab code Exa 6.17 Determination of OE parameters

```
1 // Determination of OE parameters as described in
     Example 6.28 on page 209.
2 // 6.17
3
4 exec('armac1.sci',-1);
5 exec('oe.sci',-1);
6 exec('cra.sci',-1);
7 exec('stem.sci',-1);
8 exec('filt.sci',-1);
9 exec('covf.sci',-1);
10 exec('deconvol.sci',-1);
11
12 b = [0 0 0.6 -0.2];
13 f = [1 -0.5];
```

```
c = 1; d = 1;
process_oe = armac1(1,b,c,d,f,0.05);
u = prbs_a(2555,250);
xi = rand(1,2555,'normal');
y = arsimul(process_oe,[u xi]);
z = [y(1:length(u))' u'];
zd = detrend(z,'constant');

// Compute IR for time-delay estimation
[ir,r,cl_s] = cra(zd);

// Time-delay = 2 samples
// Estimate ARX model (assume known orders)
nb = 2; nf = 1; nk = 2;
[thetaN,covfN,nvar,resid] = oe(zd,nb,nf,nk);

// Residual plot
[cov1,m1] = xcov(resid,24,"coeff");
xset('window',1);
subplot(2,1,1)
stem(0:24,cov1(25:49)');xgrid();
xtitle('Correlation function of residuals from
   output y1','lag');
[cov2,m2] = xcov(resid, zd(:,2),24,"coeff");
subplot(2,1,2)
stem(-24:24,cov2');xgrid();
xtitle('Cross corr. function between input u1 and
   residuals from output y1','lag');
```

Chapter 7

Structures and Specifications

Scilab code Exa 7.1 Procedure to draw root locus for the problem

```
1 // Procedure to draw root locus for the problem
     discussed in Example 7.1 on page 247.
2 // 7.1
3
4 exec('tf.sci',-1);
5
6 H = tf(1,[1 -1 0],-1);
7 evans(H)
```

Scilab code Exa 7.2 Procedure to draw the Nyquist plot

```
1 // Procedure to draw the Nyquist plot, as discussed
     in Example 7.2 on page 250.
2 // 7.2
3
4 exec('tf.sci',-1);
5
6 H = tf(1,[1 -1 0],-1);
7 nyquist(H,0.1,0.5)
```

Scilab code Exa 7.3 Procedure to draw Bode plots

```
1 // Procedure to draw Bode plots in Fig. 7.11 on page
      255.
2 // 7.3
3
4 exec('tf.sci',-1);
5
6 pol1 = [1 -0.9]; pol2 = [1 -0.8];
7 G1 = tf(pol1,[1 0],-1);
8 G2 = tf(pol2,[1 0],-1);
9 w = linspace(0.001,0.5,1000);
10 xset('window',1);
11 bode([G1;G2],w);
12 G = tf(pol1,pol2,-1);
13 xset('window',2);
14 bode(G,w);
```

Scilab code Exa 7.4 A procedure to design lead controllers

```
1 // A procedure to design lead controllers, as
     explained in Fig. 7.12 on page 257.
2 // 7.4
3
4 exec('tf.sci',-1)
5
6 w = linspace(0.001,%pi,1000);
7 a = linspace(0.001,0.999,100);
8 lena = length(a);
9 omega = []; lead = [];
10 for i = 1:lena,
```

```
11      zero = a(i);
12      pole = 0.9*zero;
13      sys = tf([1 -zero],[1 -pole],-1);
14      frq = w/(2*%pi);
15      [frq,repf]=repfreq(sys, frq);
16      [db,phase] =dbphi(repf);
17      [y,j] = max(phase);
18      omega = [omega w(j)];
19      lead = [lead y];
20      comega = (pole+zero)/(pole*zero+1);
21      clead = zero-pole;
22      clead1 = sqrt((1-zero^2)*(1-pole^2));
23      clead = clead/clead1;
24  //     [w(j) acos(comega) y atan(clead)*180/pi]
25  end
26  subplot(2,1,1), plot(lead,omega)
27  xtitle('','','Frequency, in radians'), xgrid;
28  halt;
29  subplot(2,1,2), plot(lead,a)
30  xtitle('','Lead generated, in degrees','Zero
       location'), xgrid;
```

Scilab code Exa 7.5 Bode plot of a lead controller

```
1  // Bode plot of a lead controller, as shown in Fig.
       7.13 on page 257.
2  // 7.5
3
4  exec('tf.sci',-1);
5
6  w = linspace(0.001,0.5,1000);
7  G = tf([1 -0.8],[1 -0.24],-1);
8  bode(G,w)
```

Scilab code Exa 7.6 Verification of performance of lead controller on antenna system

```
1 // Verification of performance of lead controller on
      antenna system, as discussed in Example 7.3.
2 // 7.6
3
4 // Continuous time antenna model
5 a = 0.1;
6 F = [0 1;0 -a]; g = [0; a]; c = [1 0]; d = 0;
7 Ga = syslin('c',F,g,c,d); [ds,num,den] = ss2tf(Ga);
8 Num = clean(num); Den = clean(den);
9 Ts = 0.2;
10 G = dscr(Ga,Ts);
11
12 // lead controller
13 beta1 = 0.8;
14 N = [1 -0.9802]*(1-beta1)/(1-0.9802); Rc = [1 -beta1
      ];
15
16 // simulation parameters using g_s_cl2.cos
17 gamm = 1; Sc = 1; Tc = 1; C = 0; D = 1;
18 st = 1; st1 = 0;
19 t_init = 0; t_final = 20;
20
21 // u1: -4 to 11
22 // y1: 0 to 1.4
23 exec('cosfil_ip.sci',-1);
24 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
25 [Np,Rcp] = cosfil_ip(N,Rc); // N/Rc
26 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
27 [Cp,Dp] = cosfil_ip(C,D); // C/D
```

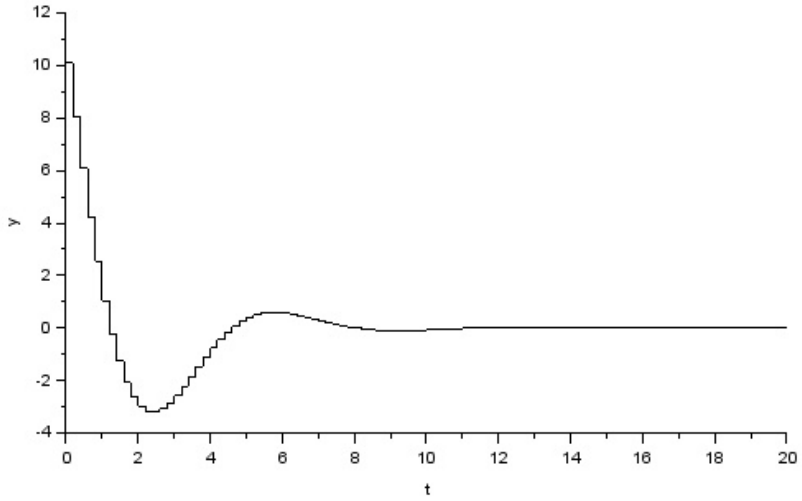

Figure 7.1: Verification of performance of lead controller on antenna system

This code can be downloaded from the website wwww.scilab.in

This code can be downloaded from the website wwww.scilab.in

Scilab code Exa 7.7 Illustration of system type

```
1 // Illustration of system type, as explained in
    Example 7.10 on page 275.
2 // 7.7
3
4 exec('rowjoin.sci',-1);
```

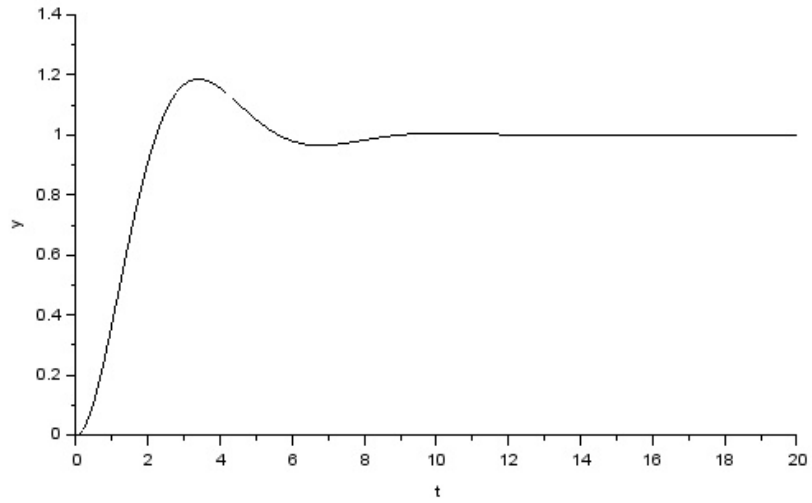

Figure 7.2: Verification of performance of lead controller on antenna system

```
5  exec('zpowk.sci',-1);
6  exec('polmul.sci',-1);
7  exec('polsize.sci',-1);
8  exec('indep.sci',-1);
9  exec('t1calc.sci',-1);
10 exec('makezero.sci',-1);
11 exec('move_sci.sci',-1);
12 exec('clcoef.sci',-1);
13 exec('colsplit.sci',-1);
14 exec('seshft.sci',-1);
15 exec('left_prm.sci',-1);
16 exec('cindep.sci',-1);
17 exec('xdync.sci',-1);
18 exec('pp_pid.sci',-1);
19 exec('cosfil_ip.sci');
20
21 // Plant
22 B = 1; A = [1 -1]; zk = [0 1]; Ts = 1; k = 1;
23 // Value of k absent in original code
24 // Specify closed loop characteristic polynomial
```

```
25 phi = [1 -0.5];
26
27 // Design the controller
28 reject_ramps = 1;
29   if reject_ramps == 1,
30     Delta = [1 -1]; // to reject ramps another Delta
31   else
32     Delta = 1; // steps can be rejected by plant
         itself
33   end
34 [Rc,Sc] = pp_pid(B,A,k,phi,Delta);
35
36 // parameters for simulation using stb_disc.mdl
37 Tc = Sc; gamm = 1; N = 1;
38 C = 0; D = 1; N_var = 0;
39 st = 1; t_init = 0; t_final = 20;
40
41 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
42 [Rcp1,Rcp2] = cosfil_ip(1,Rc); // 1/Rc
43 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
44 [Bp,Ap] = cosfil_ip(B,A); // B/A
45 [zkp1,zkp2] = cosfil_ip(zk,1); // zk/1
46 [Cp,Dp] = cosfil_ip(C,D); // C/D
47
48 // Give appropriate path
49 //xcos('stb_disc.xcos');
```

Scilab code Exa 7.8 Solution to Aryabhatta identity

```
1 // Solution to Aryabhatta's identity, presented in
    Example 7.12 on page 293.
```

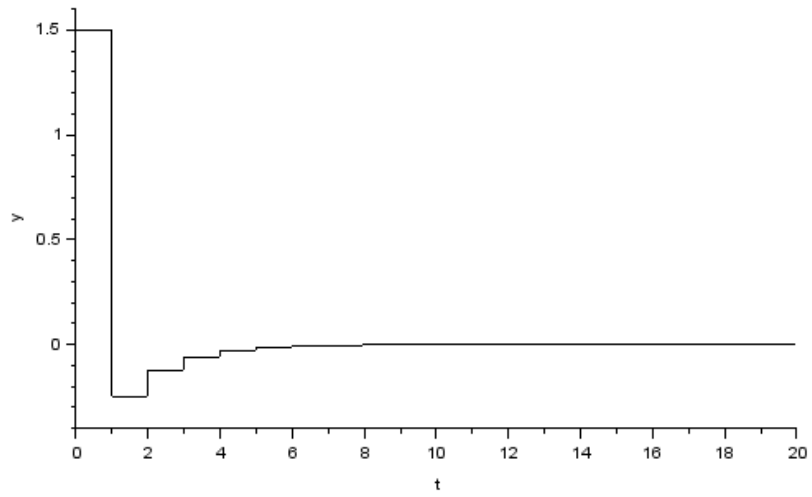

Figure 7.3: Illustration of system type

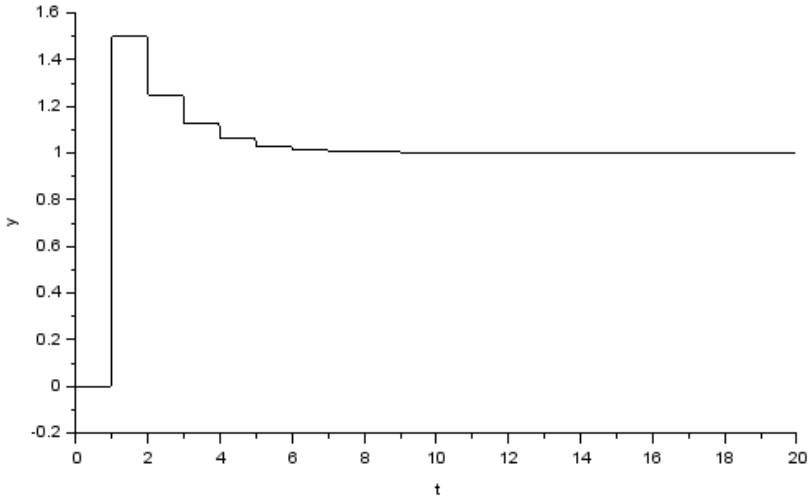

Figure 7.4: Illustration of system type

```
2 // 7.8
3
4 exec('indep.sci',-1);
5 exec('rowjoin.sci',-1);
6 exec('polsize.sci',-1);
7 exec('makezero.sci',-1);
8 exec('clcoef.sci',-1);
9 exec('cindep.sci',-1);
10 exec('seshft.sci',-1);
11 exec('move_sci.sci',-1);
12 exec('colsplit.sci',-1);
13 exec('left_prm.sci',-1);
14 exec('t1calc.sci',-1);
15 exec('xdync.sci',-1);
16
17 N = convol([0 1],[1 1]);
18 D = convol([1 -4],[1 -1]);
19 dN = 2; dD = 2;
20 C = [1 -1 0.5];
21 dC = 2;
22 [Y,dY,X,dX,B,dB,A,dA] = xdync(N,dN,D,dD,C,dC)
```

Scilab code Exa 7.9 Left coprime factorization

```
1 // Left coprime factorization as discussed in
    Example 7.13 on page 295.
2 // 7.9
3
4 exec('rowjoin.sci',-1);
5 exec('makezero.sci',-1);
6 exec('colsplit.sci',-1);
7 exec('clcoef.sci',-1);
8 exec('polsize.sci',-1);
9 exec('seshft.sci',-1);
10 exec('indep.sci',-1);
```

```
11 exec('move_sci.sci',-1);
12 exec('t1calc.sci',-1);
13 exec('left_prm.sci',-1);
14
15 D = [1 0 0 0 0 0
16 0 1 0 1 0 0
17 0 0 1 1 1 0];
18 N = [
19 1 0 0
20 0 1 0
21 0 0 1];
22 dD = 1;
23 dN = 0;
24 [B,dB,A,dA] = left_prm(N,dN,D,dD)
```

Scilab code Exa 7.10 Solution to polynomial equation

```
1 // Solution to polynomial equation, as discussed in
     Example 7.14 on page 295.
2 // 7.10
3
4 exec('move_sci.sci',-1);
5 exec('makezero.sci',-1);
6 exec('seshft.sci',-1);
7 exec('colsplit.sci',-1);
8 exec('clcoef.sci',-1);
9 exec('cindep.sci',-1);
10 exec('indep.sci',-1);
11 exec('t1calc.sci',-1);
12 exec('left_prm.sci',-1);
13 exec('polsize.sci',-1);
14 exec('rowjoin.sci',-1);
15 exec('xdync.sci',-1);
16
17 N = [0  4 0 1
```

```
18          -1  8  0  3];
19  dN = 1;
20  D = [0 0   1 4 0 1
21          0 0 -1 0 0 0];
22  dD = 2;
23  C = [1 0 1 1
24       0 2 0 1];
25  dC = 1;
26  [Y,dY,X,dX,B,dB,A,dA] = xdync(N,dN,D,dD,C,dC)
```

Chapter 8

Proportional Integral Derivative Controllers

Scilab code Exa 8.1 Continuous to discrete time transfer function

```
1 // Continuous to discrete time transfer function
2 // 8.1
3
4 exec('tf.sci');
5
6 sys = tf(10,[5 1]);
7 sysd = ss2tf(dscr(sys,0.5));
```

Chapter 9

Pole Placement Controllers

Scilab code Exa 9.1 Pole placement controller for magnetically suspended ball problem

```
1 // Pole placement controller for magnetically
     suspended ball problem, discussed in Example 9.3
     on page 331.
2 // 9.1
3
4 exec('myc2d.sci',-1);
5 exec('desired.sci',-1);
6 exec('zpowk.sci',-1);
7 exec('polsplit2.sci',-1);
8 exec('polsize.sci',-1);
9 exec('t1calc.sci',-1);
10 exec('indep.sci',-1);
11 exec('move_sci.sci',-1);
12 exec('colsplit.sci',-1);
13 exec('clcoef.sci',-1);
14 exec('cindep.sci',-1);
15 exec('polmul.sci',-1);
16 exec('seshft.sci',-1);
17 exec('makezero.sci',-1);
18 exec('xdync.sci',-1);
```

```
19 exec('left_prm.sci',-1);
20 exec('rowjoin.sci',-1);
21 exec('pp_basic.sci',-1);
22 exec('polyno.sci',-1);
23 exec('cosfil_ip.sci',-1);
24
25 // Magnetically suspended ball problem
26 // Operating conditions
27 M = 0.05; L = 0.01; R = 1; K = 0.0001; g = 9.81;
28
29 //Equilibrium conditions
30 hs = 0.01; is = sqrt(M*g*hs/K);
31
32 // State space matrices
33 a21 = K*is^2/M/hs^2; a23 = - 2*K*is/M/hs; a33 = - R/
     L;
34 b3 = 1/L;
35 a1 = [0 1 0; a21 0 a23; 0 0 a33];
36 b1 = [0; 0; b3]; c1 = [1 0 0]; d1 = 0;
37
38 // Transfer functions
39 G = syslin('c',a1,b1,c1,d1); Ts = 0.01;
40 [B,A,k] = myc2d(G,Ts);
41
42 //polynomials are returned
43 [Ds,num,den] = ss2tf(G);
44 num = clean(num); den = clean(den);
45
46 // Transient specifications
47 rise = 0.15; epsilon = 0.05;
48 phi = desired(Ts,rise,epsilon);
49
50 // Controller design
51 [Rc,Sc,Tc,gamm] = pp_basic(B,A,k,phi);
52
53 // Setting up simulation parameters for basic.xcos
54 st = 0.0001; // desired change in h, in m.
55 t_init = 0; // simulation start time
```

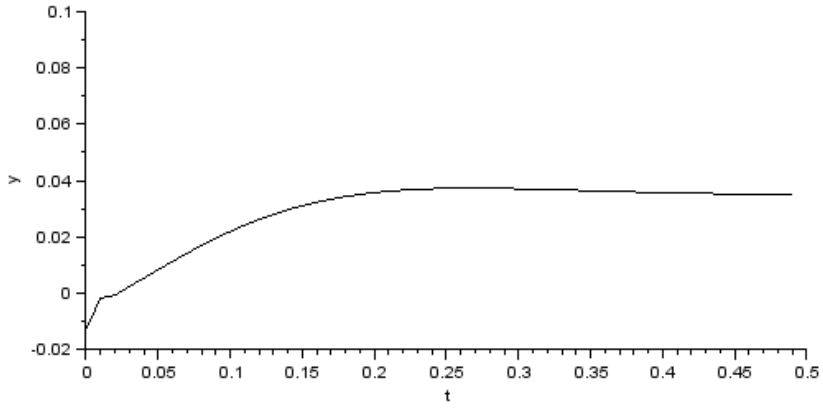

Figure 9.1: Pole placement controller for magnetically suspended ball problem

```
56 t_final = 0.5; // simulation end time
57
58 // Setting up simulation parameters for c_ss_cl.xcos
59 N_var = 0; xInitial = [0 0 0]; N = 1; C = 0; D = 1;
60
61 [Tc1,Rc1] = cosfil_ip(Tc,Rc); // Tc/Rc
62 [Sc2,Rc2] = cosfil_ip(Sc,Rc); // Sc/Rc
63
64 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
65 [Np,Rcp] = cosfil_ip(N,Rc); // 1/Rc
66 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
67 [Cp,Dp] = cosfil_ip(C,D); // C/D
```

This code can be downloaded from the website wwww.scilab.in This code

can be downloaded from the website wwww.scilab.in

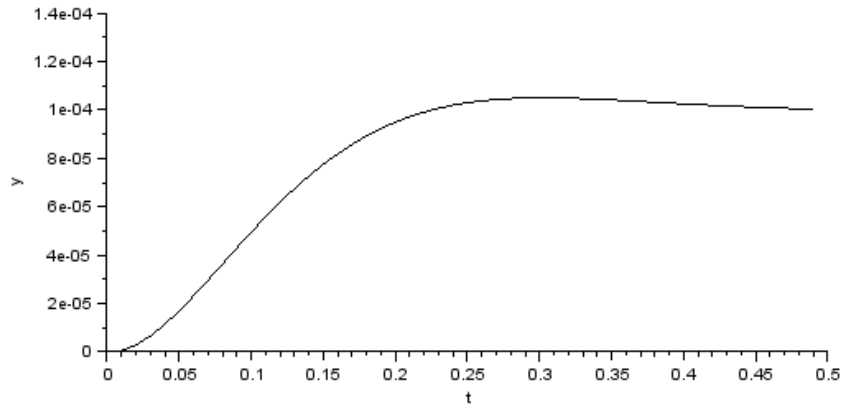

Figure 9.2: Pole placement controller for magnetically suspended ball problem

Scilab code Exa 9.2 Discretization of continuous transfer function

```
1 // Discretization of continuous transfer function.
    The result is numerator and denominator in powers
     of z^{-1} and the delay term k.
2 // 9.2
3 // function [B,A,k] = myc2d(G,Ts)
4 // Produces numerator and denominator of discrete
    transfer
5 // function in powers of z^{-1}
6 // G is continuous transfer function; time delays
    are not allowed
7 // Ts is the sampling time, all in consistent time
    units
8
9 function [B,A,k] = myc2d(G,Ts)
```

```
10 H = ss2tf(dscr(G,Ts));
11 num1 = coeff(H('num'));
12 den1 = coeff(H('den'));//---------------
13 A = den1(length(den1):-1:1);
14 num2 = num1(length(num1):-1:1);  //flip
15 nonzero = find(num1);
16 first_nz = nonzero(1);
17 B = num2(first_nz:length(num2)); //---------------
18 k = length(den1) - length(num1);
19 endfunction
```

Scilab code Exa 9.3 Procedure to split a polynomial into good and bad factors

```
1 // Procedure to split a polynomial into good and bad
     factors, as discussed in Sec. 9.2.
2 // 9.3
3 // function [goodpoly,badpoly] = polsplit2(fac,a)
4 // Splits a scalar polynomial of z^{-1} into good
    and bad
5 // factors.
6 // Input is a polynomial in increasing degree of z
    ^{-1}
7 // Optional input is a, where a <= 1.
8 // Factor that has roots of z^{-1} outside a is
    called
9 // good and the rest bad.
10 // If a is not specified, it will be assumed as
    1-1.0e-5
11
12 function [goodpoly,badpoly] = polsplit2(fac,a)
13 if argn(2) == 1, a = 1-1.0e-5; end
14 if a>1 error('good polynomial is unstable'); end
15 fac1 = poly(fac(length(fac):-1:1),'z','coeff');
16 rts1 = roots(fac1);
```

```
17 rts = rts1(length(rts1):-1:1);
18
19 // extract good and bad roots
20 badindex = find(abs(rts)>=a); // mtlb_find has been
      replaced by find
21 badpoly = coeff(poly((rts(badindex)),"z","roots"));
22 goodindex = find(abs(rts)<a); // mtlb_find has been
      replaced by find
23 goodpoly = coeff(poly(rts(goodindex),"z","roots"));
24
25 // scale by equating the largest terms
26 [m,index] = max(abs(fac));
27 goodbad = convol(goodpoly,badpoly);
28 goodbad1 = goodbad(length(goodbad):-1:1);//--
29 factor1 = fac(index)/goodbad1(index);//--
30 goodpoly = goodpoly * factor1;
31 goodpoly = goodpoly(length(goodpoly):-1:1);
32 badpoly = badpoly(length(badpoly):-1:1);
33 endfunction;
```

Scilab code Exa 9.4 Calculation of desired closed loop characteristic polynomial

```
1 // Calculation of desired closed loop characteristic
     polynomial, as discussed in Sec. 7.7.
2 // 9.4
3
4 // function [phi,dphi] = desired(Ts,rise,epsilon)
5 // Based on transient requirements,
6 // calculates closed loop characteristic polynomial
7
8 function [phi,dphi] = desired(Ts,rise,epsilon)
9 Nr = rise/Ts; omega = %pi/2/Nr; rho = epsilon^(omega
     /%pi);
10 phi = [1 -2*rho*cos(omega) rho^2]; dphi = length(phi
```

```
    ) -1;
endfunction;
```

Scilab code Exa 9.5 Design of 2 DOF pole placement controller

```
// Design of 2-DOF pole placement controller, as
   discussed in Sec. 9.2.
// 9.5

// function [Rc,Sc,Tc,gamma] = pp_basic(B,A,k,phi)
// calculates pole placement controller


function [Rc,Sc,Tc,gamm] = pp_basic(B,A,k,phi)

// Setting up and solving Aryabhatta identity
[Ag,Ab] = polsplit2(A); dAb = length(Ab) - 1;
[Bg,Bb] = polsplit2(B); dBb = length(Bb) - 1;

[zk,dzk] = zpowk(k);

[N,dN] = polmul(Bb,dBb,zk,dzk);
dphi = length(phi) - 1;

[S1,dS1,R1,dR1] = xdync(N,dN,Ab,dAb,phi,dphi);

// Determination of control law
Rc = convol(Bg,R1); Sc = convol(Ag,S1);
Tc = Ag; gamm = sum(phi)/sum(Bb);

endfunction;
```

Scilab code Exa 9.6 Evaluates z to the power k

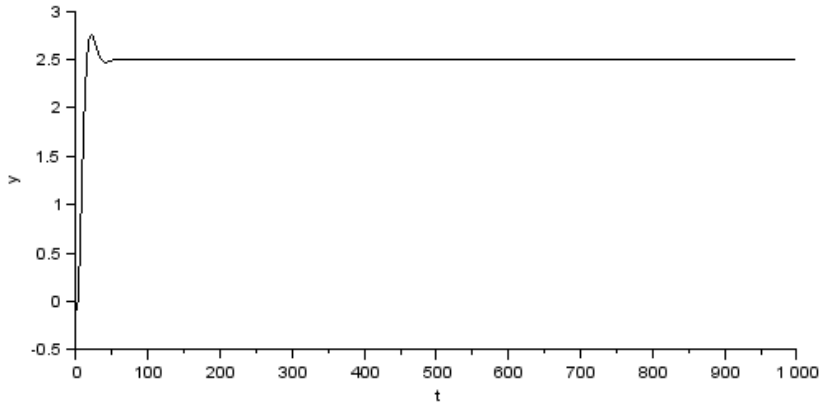

Figure 9.3: Simulation of closed loop system with an unstable controller

```
1 // Evaluates z^-k.
2 // 9.6
3
4 function [zk,dzk] = zpowk(k)
5 zk = zeros(1,k+1); zk(1,k+1) = 1;
6 dzk = k;
7 endfunction
```

This code can be downloaded from the website wwww.scilab.in This code

can be downloaded from the website wwww.scilab.in

Scilab code Exa 9.7 Simulation of closed loop system with an unstable controller

```
1 // Simulation of closed loop system with an unstable
```

```
    controller, as discussed in Example 9.5 on page
    335.
// 9.7

exec('desired.sci',-1);
exec('zpowk.sci',-1);
exec('polmul.sci',-1);
exec('polsplit2.sci',-1);
exec('polsize.sci',-1);
exec('xdync.sci',-1);
exec('rowjoin.sci',-1);
exec('left_prm.sci',-1);
exec('t1calc.sci',-1);
exec('indep.sci',-1);
exec('makezero.sci',-1);
exec('move_sci.sci',-1);
exec('colsplit.sci',-1);
exec('clcoef.sci',-1);
exec('cindep.sci',-1);
exec('seshft.sci',-1);
exec('cosfil_ip.sci',-1);
exec('pp_basic.sci',-1);

Ts = 1; B = [1 -3]; A = [1 2 -8]; k = 1;
// Since k=1, tf is of the form z^-1
[zk,dzk] = zpowk(k); // int1 = 0;//---- int1

// Transient specifications
rise = 10; epsilon = 0.1;
phi = desired(Ts,rise,epsilon);

// Controller design
[Rc,Sc,Tc,gamm] = pp_basic(B,A,k,phi);

// simulation parameters for basic_disc.xcos
//While simulating for t_final = 100, set the limit
    of Y axis of each scope
//u1: -0.2 to 3
```

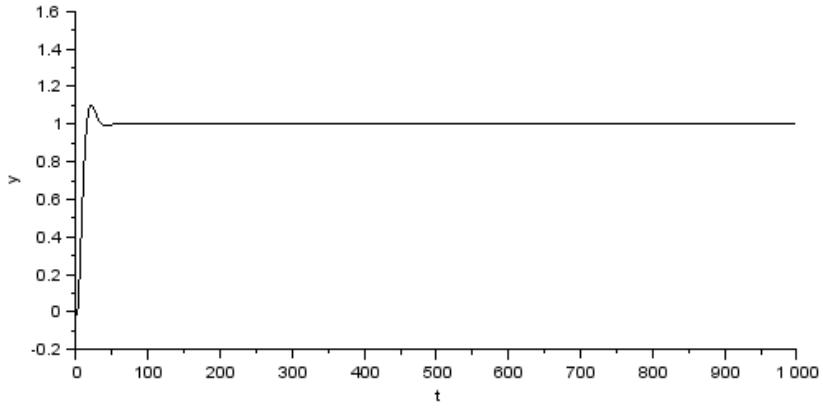

Figure 9.4: Simulation of closed loop system with an unstable controller

```
37 //y1: -0.1 to 1.2
38 st = 1.0; // Desired change in setpoint
39 t_init = 0; // Simulation start time
40 t_final = 1000; // Simulation end time
41
42 // Simulation parameters for stb_disc.xcos
43 N_var = 0; C = 0; D = 1; N = 1;
44
45 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
46 [Rcp1,Rcp2] = cosfil_ip(1,Rc); // 1/Rc
47 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
48 [Bp,Ap] = cosfil_ip(B,A); // B/A
49 [zkp1,zkp2] = cosfil_ip(zk,1); // zk/1
50 [Cp,Dp] = cosfil_ip(C,D); // C/D
51
52 [Tcp,Rcp] = cosfil_ip(Tc,Rc); // Tc/Rc
53 [Scp_b,Rcp_b] = cosfil_ip(Sc,Rc); // Sc/Rc
```

Scilab code Exa 9.8 Pole placement controller using internal model principle

```
1  // Pole placement controller using internal model
       principle, as discussed in Sec. 9.4.
2  // 9.8
3
4  // function [Rc,Sc,Tc,gamma,phit] = pp_im(B,A,k,phi,
       Delta)
5  // Calculates 2-DOF pole placement controller.
6
7  function [Rc,Sc,Tc,gamm] = pp_im(B,A,k,phi,Delta)
8
9  // Setting up and solving Aryabhatta identity
10 [Ag,Ab] = polsplit3(A); dAb = length(Ab) - 1;
11 [Bg,Bb] = polsplit3(B); dBb = length(Bb) - 1;
12
13 [zk,dzk] = zpowk(k);
14
15 [N,dN] = polmul(Bb,dBb,zk,dzk);
16 dDelta = length(Delta)-1;
17 [D,dD] = polmul(Ab,dAb,Delta,dDelta);
18 dphi = length(phi)-1;
19
20 [S1,dS1,R1,dR1] = xdync(N,dN,D,dD,phi,dphi);
21
22 // Determination of control law
23 Rc = convol(Bg,convol(R1,Delta)); Sc = convol(Ag,S1)
       ;
24 Tc = Ag; gamm = sum(phi)/sum(Bb);
25 endfunction;
```

Scilab code Exa 9.9 Pole placement controller with internal model of a step for the magnetically suspended ball problem

```
1  // Pole placement controller, with internal model of
       a step, for the magnetically suspended ball
     problem, as discussed in Example 9.8 on page 339.
2  // 9.9
3
4  // PP control with internal model for ball problem
5  exec('desired.sci',-1);
6  exec('pp_im.sci',-1);
7  exec('myc2d.sci',-1);
8  exec('polsplit3.sci',-1);
9  exec('zpowk.sci',-1);
10 exec('rowjoin.sci',-1);
11 exec('left_prm.sci',-1);
12 exec('t1calc.sci',-1);
13 exec('indep.sci',-1);
14 exec('cindep.sci',-1);
15 exec('seshft.sci',-1);
16 exec('makezero.sci',-1);
17 exec('move_sci.sci',-1);
18 exec('colsplit.sci',-1);
19 exec('clcoef.sci',-1);
20 exec('polmul.sci',-1);
21 exec('polsize.sci',-1);
22 exec('xdync.sci',-1);
23 exec('cosfil_ip.sci',-1);
24 exec('polyno.sci',-1);
25
26 // Operating conditions
27 M = 0.05; L = 0.01; R = 1; K = 0.0001; g = 9.81;
28
29 // Equilibrium conditions
30 hs = 0.01; is = sqrt(M*g*hs/K);
31
32 // State space matrices
33 a21 = K*is^2/M/hs^2; a23 = - 2*K*is/M/hs; a33 = - R/
     L;
34 b3 = 1/L;
35 a1 = [0 1 0; a21 0 a23; 0 0 a33];
```

```
36 b1 = [0; 0; b3]; c1 = [1 0 0]; d1 = 0;
37
38 // Transfer functions
39 G = syslin('c',a1,b1,c1,d1); Ts = 0.01; [B,A,k] =
      myc2d(G,Ts);
40
41 // Transient specifications
42 rise = 0.1; epsilon = 0.05;
43 phi = desired(Ts,rise,epsilon);
44
45 // Controller design
46 Delta = [1 -1];  //internal model of step used
47 [Rc,Sc,Tc,gamm] = pp_im(B,A,k,phi,Delta);
48
49 // simulation parameters for c_ss_cl.xcos
50 st = 0.0001;  //desired change in h, in m.
51 t_init = 0; // simulation start time
52 t_final = 0.5; //simulation end time
53 xInitial = [0 0 0];
54 N = 1; C = 0; D = 1; N_var = 0;
55
56 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
57 [Np,Rcp] = cosfil_ip(N,Rc); // 1/Rc
58 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
59 [Cp,Dp] = cosfil_ip(C,D); // C/D
```

This code can be downloaded from the website wwww.scilab.in

Scilab code Exa 9.10 Pole placement controller IBM Lotus Domino server

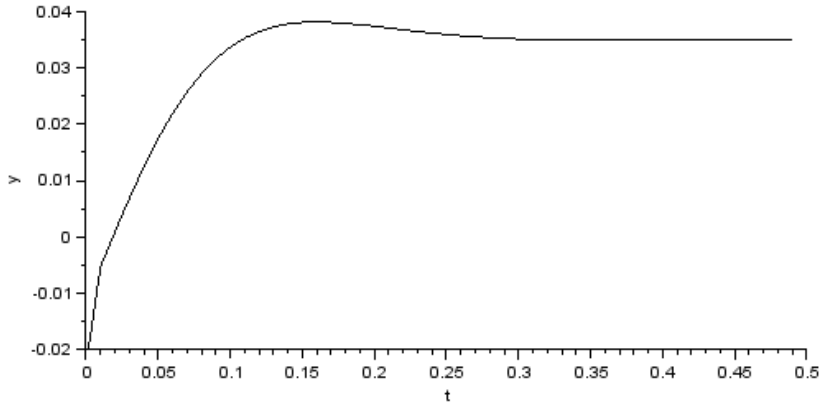

Figure 9.5: Pole placement controller with internal model of a step for the magnetically suspended ball problem

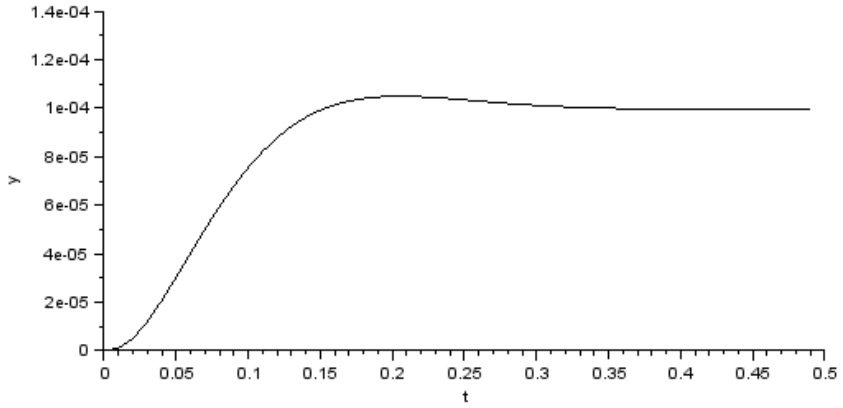

Figure 9.6: Pole placement controller with internal model of a step for the magnetically suspended ball problem

```
1  // Pole placement controller IBM Lotus Domino server
       , discussed in Example 9.9 on page 341.
2  // 9.10
3
4  exec('desired.sci',-1);
5  exec('pp_im.sci',-1);
6  exec('zpowk.sci',-1);
7  exec('cosfil_ip.sci',-1);
8  exec('polsplit3.sci',-1);
9  exec('polmul.sci',-1);
10 exec('polsize.sci',-1);
11 exec('xdync.sci',-1);
12 exec('rowjoin.sci',-1);
13 exec('left_prm.sci',-1);
14 exec('t1calc.sci',-1);
15 exec('polmul.sci',-1);
16 exec('indep.sci',-1);
17 exec('seshft.sci',-1);
18 exec('makezero.sci',-1);
19 exec('move_sci.sci',-1);
20 exec('colsplit.sci',-1);
21 exec('clcoef.sci',-1);
22 exec('cindep.sci',-1);
23 exec('polyno.sci',-1);
24
25 // Control of IBM lotus domino server
26 // Transfer function
27 B = 0.47; A = [1 -0.43]; k = 1;
28 [zk,dzk] = zpowk(k);
29
30 // Transient specifications
31 rise = 10; epsilon = 0.01; Ts = 1;
32 phi = desired(Ts,rise,epsilon);
33
34 // Controller design
35 Delta = [1 -1];  // internal model of step used
36 [Rc,Sc,Tc,gamm] = pp_im(B,A,k,phi,Delta);
37
```

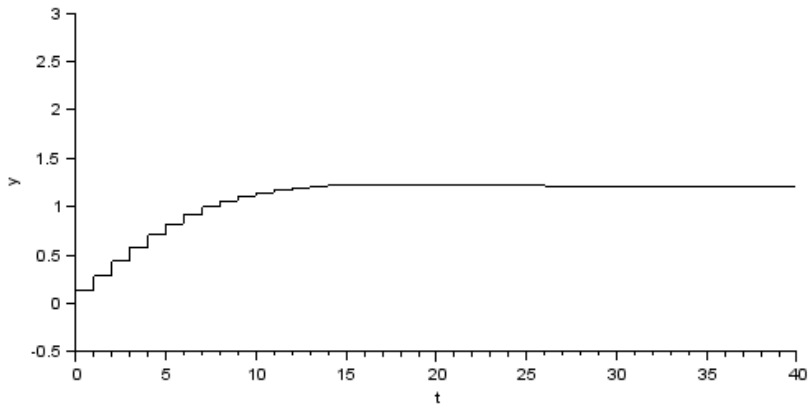

Figure 9.7: Pole placement controller IBM Lotus Domino server

```
38 // Simulation parameters for stb_disc.xcos
39 st = 1; // desired change
40 t_init = 0; // simulation start time
41 t_final = 40; // simulation end time
42 C = 0; D = 1; N_var = 0;
43
44 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
45 [Rcp1,Rcp2] = cosfil_ip(1,Rc); // 1/Rc
46 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
47 [Bp,Ap] = cosfil_ip(B,A); // B/A
48 [zkp1,zkp2] = cosfil_ip(zk,1); // zk/1
49 [Cp,Dp] = cosfil_ip(C,D); // C/D
```

This code can be downloaded from the website wwww.scilab.in

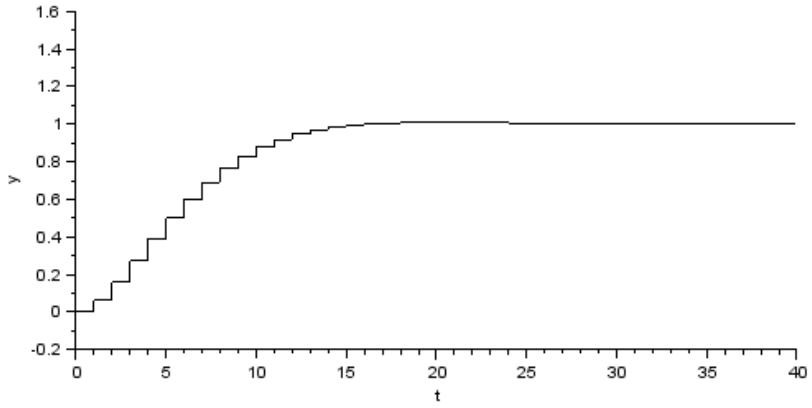

Figure 9.8: Pole placement controller IBM Lotus Domino server

This code can be downloaded from the website wwww.scilab.in

Scilab code Exa 9.11 Pole placement controller for motor problem

```
1 // Pole placement controller for motor problem,
    discussed in Example 9.10 on page 343.
2 // 9.11
3
4 exec('desired.sci',-1);
5 exec('pp_im.sci',-1);
6 exec('myc2d.sci',-1);
7 exec('cosfil_ip.sci',-1);
8 exec('polsplit3.sci',-1);
9 exec('zpowk.sci',-1);
10 exec('polmul.sci',-1);
11 exec('polsize.sci',-1);
12 exec('xdync.sci',-1);
13 exec('rowjoin.sci',-1);
```

```
14 exec('left_prm.sci',-1);
15 exec('t1calc.sci',-1);
16 exec('indep.sci',-1);
17 exec('seshft.sci',-1);
18 exec('makezero.sci',-1);
19 exec('move_sci.sci',-1);
20 exec('colsplit.sci',-1);
21 exec('clcoef.sci',-1);
22 exec('cindep.sci',-1);
23 exec('polyno.sci',-1);
24
25 // Motor control problem
26 // Transfer function
27 a1 = [-1 0; 1 0]; b1 = [1; 0]; c1 = [0 1]; d1 = 0;
28 G = syslin('c',a1,b1,c1,d1); Ts = 0.25;
29 [B,A,k] = myc2d(G,Ts);
30
31 // Transient specifications
32 rise = 3; epsilon = 0.05;
33 phi = desired(Ts,rise,epsilon);
34
35 // Controller design
36 Delta = 1; // No internal model of step used
37 [Rc,Sc,Tc,gamm] = pp_im(B,A,k,phi,Delta);
38
39 // simulation parameters for c_ss_cl.xcos
40 st = 1;  //desired change in position
41 t_init = 0;  //simulation start time
42 t_final = 10;  //simulation end time
43 xInitial = [0 0];  //initial conditions
44 N = 1; C = 0; D = 1; N_var = 0;
45
46 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
47 [Np,Rcp] = cosfil_ip(N,Rc); // 1/Rc
48 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
49 [Cp,Dp] = cosfil_ip(C,D); // C/D
```

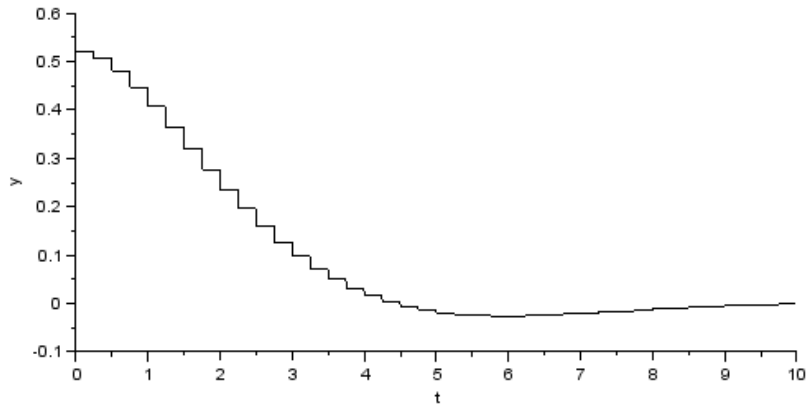

Figure 9.9: Pole placement controller for motor problem

Scilab code Exa 9.12 Procedure to split a polynomial into good and bad factors

```
1 // Procedure to split a polynomial into good and bad
     factors, as discussed in Sec. 9.5. The factors
    that have roots outside unit circle or with
    negative real parts are defined as bad.
2 // 9.12
3
4 // function [goodpoly,badpoly] = polsplit3(fac,a)
5 // Splits a scalar polynomial of z^{-1} into good
    and bad
6 // factors. Input is a polynomial in increasing
    degree of
7 // z^{-1}. Optional input is a, where a <= 1.
```

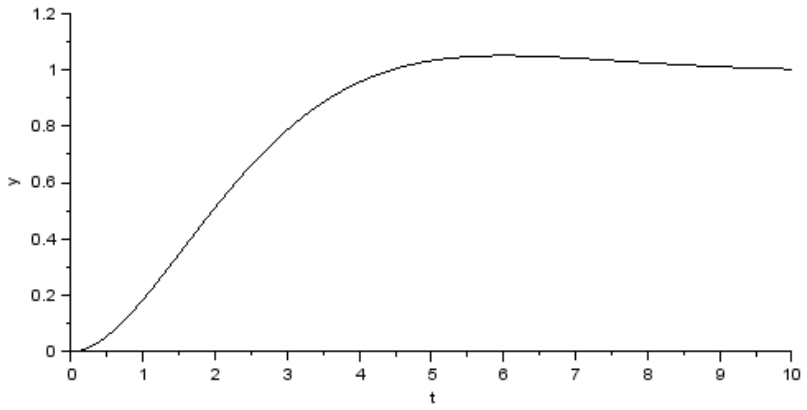

Figure 9.10: Pole placement controller for motor problem

```
8  // Factors that have roots outside a circle of
       radius a or
9  // with negative roots will be called bad and the
       rest
10 // good. If a is not specified, it will be assumed
       as 1.
11
12 function [goodpoly,badpoly] = polsplit3(fac,a)
13 if argn(2) == 1, a = 1; end
14 if a>1 error('good polynomial also is unstable');
       end
15 fac1 = poly(fac(length(fac):-1:1),'z','coeff');
16 rts = roots(fac1);
17 rts = rts(length(rts):-1:1);
18
19 // extract good and bad roots
20 badindex = mtlb_find((abs(rts)>=a-1.0e-5)|(real(rts)
       <-0.05));
21 badpoly = coeff(poly(rts(badindex),'z'));
22 goodindex = mtlb_find((abs(rts)<a-1.0e-5)&(real(rts)
       >=-0.05));
23 goodpoly = coeff(poly(rts(goodindex),'z'));
24
```

```
25 // scale by equating the largest terms
26 [m,index] = max(abs(fac));
27 goodbad = convol(goodpoly,badpoly);
28 goodbad = goodbad(length(goodbad):-1:1);
29 factor1 = fac(index)/goodbad(index);
30 goodpoly = goodpoly * factor1;
31 goodpoly = goodpoly(length(goodpoly):-1:1);
32 badpoly = badpoly(length(badpoly):-1:1);
33 endfunction;
```

Scilab code Exa 9.13 Pole placement controller without intra sample oscillations

```
1 // Pole placement controller without intra sample
     oscillations, as discussed in Sec. 9.5.
2 // 9.13
3
4 // function [Rc,Sc,Tc,gamma,phit] = pp_im2(B,A,k,phi
     ,Delta,a)
5 // 2-DOF PP controller with internal model of Delta
     and without
6 // hidden oscillations
7
8 function [Rc,Sc,Tc,gamm,phit] = pp_im2(B,A,k,phi,
     Delta,a)
9
10 if argn(2) == 5, a = 1; end
11 dphi = length(phi)-1;
12
13 // Setting up and solving Aryabhatta identity
14 [Ag,Ab] = polsplit3(A,a); dAb = length(Ab) - 1;
15 [Bg,Bb] = polsplit3(B,a); dBb = length(Bb) - 1;
16
17 [zk,dzk] = zpowk(k);
18
```

```
19 [N,dN] = polmul(Bb,dBb,zk,dzk);
20 dDelta = length(Delta)-1;
21 [D,dD] = polmul(Ab,dAb,Delta,dDelta);
22
23 [S1,dS1,R1,dR1] = xdync(N,dN,D,dD,phi,dphi);
24
25 // Determination of control law
26 Rc = convol(Bg,convol(R1,Delta)); Sc = convol(Ag,S1)
      ;
27 Tc = Ag; gamm = sum(phi)/sum(Bb);
28
29 // Total characteristic polynomial
30 phit = convol(phi,convol(Ag,Bg));
31 endfunction;
```

This code can be downloaded from the website wwww.scilab.in

Scilab code Exa 9.14 Controller design

```
1 // Controller design for the case study presented in
      Example 9.12 on page 347.
2 // 9.14
3
4 exec('tf.sci',-1);
5 exec('desired.sci',-1);
6 exec('zpowk.sci',-1);
7 exec('myc2d.sci',-1);
8 exec('polsplit3.sci',-1);
9 exec('polmul.sci',-1);
10 exec('polsize.sci',-1);
11 exec('xdync.sci',-1);
12 exec('rowjoin.sci',-1);
13 exec('left_prm.sci',-1);
14 exec('t1calc.sci',-1);
```

```
15 exec('indep.sci',-1);
16 exec('pp_im2.sci',-1);
17 exec('seshft.sci',-1);
18 exec('makezero.sci',-1);
19 exec('move_sci.sci',-1);
20 exec('colsplit.sci',-1);
21 exec('clcoef.sci',-1);
22 exec('cindep.sci',-1);
23 exec('cosfil_ip.sci',-1);
24
25 num = 200;
26 den = convol([0.05 1],[0.05 1]);
27 den = convol([10 1],den);
28 G = tf(num,den); Ts = 0.025;
29 num = G('num'); den = G('den');
30 // iodel = 0;
31 [B,A,k] = myc2d(G,Ts);
32 [zk,dzk] = zpowk(k); //int1 = 0;
33
34 // Transient specifications
35 a = 0.9; rise = 0.24; epsilon = 0.05;
36 phi = desired(Ts,rise,epsilon);
37
38 // Controller design
39 Delta = [1 -1]; // internal model of step is present
40 [Rc,Sc,Tc,gamm] = pp_im2(B,A,k,phi,Delta,a);
41
42 // margin calculation
43 Lnum = convol(Sc,convol(B,zk));
44 Lden = convol(Rc,A);
45 L = tf(Lnum,Lden,Ts);
46 Gm = g_margin(L); //---- Does not match
     ---------------- (in dB)
47 Pm = p_margin(L); //---- Convergence problem
     ---------------- (in degree)
48
49 num1 = 100; den1 = [10 1];
50 Gd = tf(num1,den1);  //-------
```

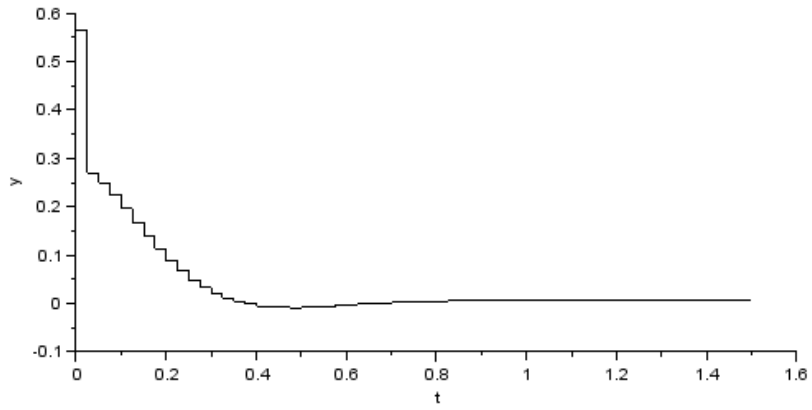

Figure 9.11: Controller design

```
51 [C,D,k1] = myc2d(Gd,Ts);
52 [zk,dzk] = zpowk(k);
53 C = convol(C,zk);
54
55 // simulation parameters g_s_cl2.xcos ----------
56 N = 1;
57 st = 1; // desired change in setpoint
58 st1 = 0; // magnitude of disturbance
59 t_init = 0; // simulation start time
60 t_final = 1.5; // simulation end time
61
62 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
63 [Np,Rcp] = cosfil_ip(N,Rc); // N/Rc
64 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
65 [Cp,Dp] = cosfil_ip(C,D); // C/D
```

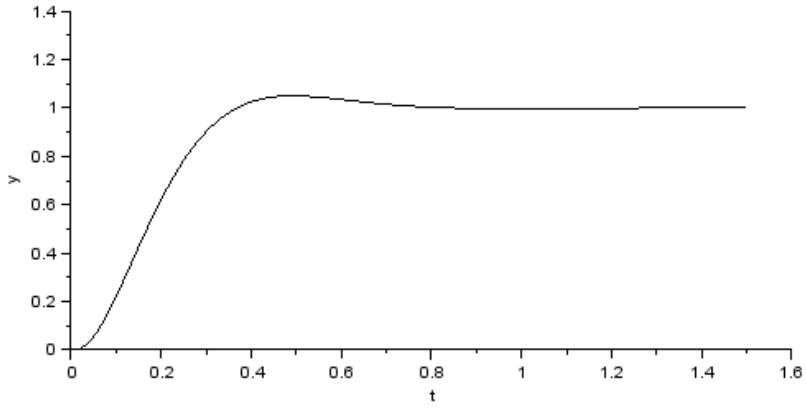

Figure 9.12: Controller design

This code can be downloaded from the website wwww.scilab.in This code

can be downloaded from the website wwww.scilab.in

Scilab code Exa 9.15 Evaluation of continuous time controller

```
1 // Evaluation of continuous time controller for the
     case study presented in Example 9.13 on page 349.
2 // 9.15
3
4 clear
5 exec('tf.sci',-1);
6 exec('myc2d.sci',-1);
7 exec('zpowk.sci',-1);
8 exec('cosfil_ip.sci',-1);
9 exec('polyno.sci',-1);
10
```

```
11 num = 200;
12 den = convol([0.05 1],[0.05 1]);
13 den = convol([10 1],den);
14 G = tf(num,den); Ts = 0.005;
15 [B,A,k] = myc2d(G,Ts);
16 [zk,dzk] = zpowk(k); //int = 0;
17
18 // Sigurd's feedback controller'
19 numb = 0.5*convol([1 2],[0.05 1]);
20 denb = convol([1 0],[0.005 1]);
21 Gb = tf(numb,denb);
22 [Sb,Rb,kb] = myc2d(Gb,Ts);
23 [zkb,dzkb] = zpowk(kb);
24 Sb = convol(Sb,zkb);
25
26 // Sigurd's feed forward controller'
27 numf = [0.5 1];
28 denf = convol([0.65 1],[0.03 1]);
29 Gf = tf(numf,denf);
30 [Sf,Rf,kf] = myc2d(Gf,Ts);
31 [zkf,dzkf] = zpowk(kf);
32 Sf = convol(Sf,zkf);
33
34 // Margins
35 simp_mode(%f);
36 L = G*Gb;
37 Gm = g_margin(L); // ------
38 Pm = p_margin(L); // ------
39 Lnum = convol(Sb,convol(zk,B));
40 Lden = convol(Rb,A);
41 L = tf(Lnum,Lden,Ts);
42 DGm = g_margin(L); // ------
43 DPm = p_margin(L); // ------
44
45 // Noise
46 num1 = 100; den1 = [10 1];
47
48 // simulation parameters for
```

```
49 // entirely continuous simulation: g_s_cl3.xcos
50 // hybrid simulation: g_s_cl6.xcos
51 st = 1; // desired change in setpoint
52 st1 = 0;
53 t_init = 0; // simulation start time
54 t_final = 5; // simulation end time
55
56 num = polyno(num,'s'); den = polyno(den,'s');
57 Numb = polyno(numb,'s'); Denb = polyno(denb,'s');
58 Numf = polyno(numf,'s'); Denf = polyno(denf,'s');
59 Num1 = polyno(num1,'s'); Den1 = polyno(den1,'s');
60
61 [Sbp,Rbp] = cosfil_ip(Sb,Rb);
62 [Sfp,Rfp] = cosfil_ip(Sf,Rf);
```

This code can be downloaded from the website wwww.scilab.in

Scilab code Exa 9.16 System type with 2 DOF controller

```
1 // System type with 2-DOF controller. It is used to
     arrive at the results Example 9.14.
2 // 9.16
3
4 exec('polsplit3.sci',-1);
5 exec('polmul.sci',-1);
6 exec('polsize.sci',-1);
7 exec('pp_im.sci',-1);
8 exec('xdync.sci',-1);
9 exec('rowjoin.sci',-1);
10 exec('left_prm.sci',-1);
11 exec('t1calc.sci',-1);
12 exec('indep.sci',-1);
13 exec('makezero.sci',-1);
14 exec('move_sci.sci',-1);
```

```
15 exec('colsplit.sci',-1);
16 exec('clcoef.sci',-1);
17 exec('cindep.sci',-1);
18 exec('seshft.sci',-1);
19 exec('zpowk.sci',-1);
20 exec('cosfil_ip.sci',-1);
21 exec('polyno.sci',-1);
22
23 B = 1; A = [1 -1]; k = 1; zk = zpowk(k); Ts = 1;
24 phi = [1 -0.5];
25
26 Delta = 1; // Choice of internal model of step
27 [Rc,Sc,Tc,gamm] = pp_im(B,A,k,phi,Delta);
28
29 // simulation parameters for stb_disc.xcos
30 st = 1; // desired step change
31 t_init = 0; // simulation start time
32 t_final = 20; // simulation end time
33 xInitial = [0 0];
34 C = 0; D = 1; N_var = 0;
35
36 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
37 [Rcp1,Rcp2] = cosfil_ip(1,Rc); // 1/Rc
38 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
39 [Bp,Ap] = cosfil_ip(B,A); // B/A
40 [zkp1,zkp2] = cosfil_ip(zk,1); // zk/1
41 [Cp,Dp] = cosfil_ip(C,D); // C/D
```

Scilab code Exa 9.17 Illustrating the benefit of cancellation

```
1 // Illustrating the benefit of cancellation. It is
     used to arrive at the results of Example 9.15.
2 // 9.17
3
4 exec('pp_im.sci',-1);
```

```
exec('pp_pid.sci',-1);
exec('zpowk.sci',-1);
exec('polmul.sci',-1);
exec('polsize.sci',-1);
exec('xdync.sci',-1);
exec('rowjoin.sci',-1);
exec('left_prm.sci',-1);
exec('t1calc.sci',-1);
exec('indep.sci',-1);
exec('seshft.sci',-1);
exec('makezero.sci',-1);
exec('move_sci.sci',-1);
exec('colsplit.sci',-1);
exec('clcoef.sci',-1);
exec('cindep.sci',-1);
exec('polyno.sci',-1);
exec('cosfil_ip.sci',-1);


// test problem to demonstrate benefits of 2_dof
// Ts = 1; B = [1  0.9]; A = conv([1  -1],[1  -0.8]); k = 1;
Ts = 1; k = 1;
B = convol([1 0.9],[1 -0.8]); A = convol([1 -1],[1 -0.5]);

// closed loop characteristic polynomial
phi = [1 -1 0.5];

Delta = 1; // Choice of internal model of step
control = 1;
if control == 1, // 1-DOF with no cancellation
   [Rc,Sc] = pp_pid(B,A,k,phi,Delta);
   Tc = Sc; gamm = 1;
else // 2-DOF
   [Rc,Sc,Tc,gamm] = pp_im(B,A,k,phi,Delta);
end

```

```
41 // simulation parameters for stb_disc.mdl
42 [zk,dzk] = zpowk(k);
43 st = 1; // desired step change
44 t_init = 0; // simulation start time
45 t_final = 20; // simulation end time
46 xInitial = [0 0];
47 C = 0; D = 1; N_var = 0;
48
49 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
50 [Rcp1,Rcp2] = cosfil_ip(1,Rc); // 1/Rc
51 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
52 [Bp,Ap] = cosfil_ip(B,A); // B/A
53 [zkp1,zkp2] = cosfil_ip(zk,1); // zk/1
54 [Cp,Dp] = cosfil_ip(C,D); // C/D
```

This code can be downloaded from the website wwww.scilab.in

Scilab code Exa 9.18 Anti windup control of IBM Lotus Domino server

```
1 // Anti windup control (AWC) of IBM Lotus Domino
    server, studied in Example 9.16 on page 357. It
    can be used for the follwoing situations: with
    and without saturation, and with and without AWC.
2 // 9.18
3
4 exec('pp_im2.sci',-1);
5 exec('desired.sci',-1);
6 exec('zpowk.sci',-1);
7 exec('cosfil_ip.sci',-1);
8 exec('polsplit3.sci',-1);
9 exec('polmul.sci',-1);
10 exec('polsize.sci',-1);
11 exec('xdync.sci',-1);
12 exec('rowjoin.sci',-1);
```

```
13 exec('left_prm.sci',-1);
14 exec('t1calc.sci',-1);
15 exec('indep.sci',-1);
16 exec('seshft.sci',-1);
17 exec('makezero.sci',-1);
18 exec('move_sci.sci',-1);
19 exec('colsplit.sci',-1);
20 exec('clcoef.sci',-1);
21 exec('polyno.sci',-1);
22 exec('cindep.sci',-1);
23 exec('poladd.sci',-1);
24
25 //  Transfer function
26 B = 0.47; A = [1 -0.43]; k = 1;
27 [zk,dzk] = zpowk(k);
28
29 // Transient specifications
30 rise = 10; epsilon = 0.01; Ts = 1;
31 phi = desired(Ts,rise,epsilon);
32
33 // Controller design
34 delta = [1 -1]; // internal model of step used
35 [Rc,Sc,Tc,gamm,F] = pp_im2(B,A,k,phi,delta);
36
37 // Study of Antiwindup Controller
38
39 key = x_choose(['Simulate without any saturation
      limits';
40           'Simulate saturation, but do not use AWC';
41           'Simulate saturation with AWC in place';
42           'Simulate with AWC, without saturation
               limits'],...
43           ['Please choose one of the following']);
44
45 if key ==0
46   disp('Invalid choice');
47   return;
48 elseif key == 1
```

```
49    U = 2; L = -2; P = 1; F = Rc; E = 0; PSc = Sc; PTc
          = Tc;
50  elseif key == 2
51    U = 1; L = -1; P = 1; F = Rc; E = 0; PSc = Sc; PTc
          = Tc;
52  else
53    if key == 3 // Antiwindup controller and with
          saturation
54       U = 1; L = -1;
55    elseif key == 4 // Antiwindup controller, but no
          saturation
56       U = 2; L = -2;
57    end
58    P = A;
59    dF = length(F) - 1;
60    PRc = convol(P,Rc); dPRc = length(PRc) - 1;
61    [E,dE] = poladd(F,dF,-PRc,dPRc);
62    PSc = convol(P,Sc); PTc = convol(P,Tc);
63  end
64
65  // Setting up simulation parameters for stb_disc_sat
66  t_init = 0; // first step begins
67  st = 1; // height of first step
68  t_init2 = 500; // second step begins
69  st2 = -2; // height of second step
70  t_final = 1000; // simulation end time
71  st1 = 0; // no disturbance input
72  C = 0; D = 1; N_var = 0;
73
74  [PTcp1,PTcp2] = cosfil_ip(PTc,1); // PTc/1
75  [Fp1,Fp2] = cosfil_ip(1,F); // 1/F
76  [Ep,Fp] = cosfil_ip(E,F); // E/F
77  [PScp1,PScp2] = cosfil_ip(PSc,1); // PSc/1
78  [Bp,Ap] = cosfil_ip(B,A); // B/A
79  [zkp1,zkp2] = cosfil_ip(zk,1); // zk/1
80  [Cp,Dp] = cosfil_ip(C,D); // C/D
```

This code can be downloaded from the website wwww.scilab.in This code

can be downloaded from the website wwww.scilab.in

Scilab code Exa 9.19 Demonstration of usefulness of negative PID parameters

```
1 // Demonstration of usefulness of negative PID
    parameters, discussed in Example 9.17 on page
    361.
2 // 9.19
3
4 exec('iodelay.sci',-1);
5 exec('delc2d.sci',-1);
6 exec('desired.sci',-1);
7 exec('pp_pid.sci',-1);
8 exec('cosfil_ip.sci',-1);
9 exec('tf.sci',-1);
10 exec('flip.sci',-1);
11 exec('zpowk.sci',-1);
12 exec('polmul.sci',-1);
13 exec('polsize.sci',-1);
14 exec('xdync.sci',-1);
15 exec('rowjoin.sci',-1);
16 exec('left_prm.sci',-1);
17 exec('t1calc.sci',-1);
18 exec('indep.sci',-1);
19 exec('seshft.sci',-1);
20 exec('makezero.sci',-1);
21 exec('move_sci.sci',-1);
22 exec('colsplit.sci',-1);
23 exec('clcoef.sci',-1);
```

```
24 exec('cindep.sci',-1);
25
26 // Discretize the continuous plant
27 num = 1; den = [2 1]; tau = 0.5;
28 G1 = tf(num,den);
29 G = iodelay(G1,tau);
30 Ts = 0.5;
31 [B,A,k] = delc2d(G,G1,Ts);
32
33 // Specify transient requirements
34 epsilon = 0.05; rise = 5;
35 phi = desired(Ts,rise,epsilon);
36
37 // Design the controller
38 Delta = [1 -1];
39 [Rc,Sc] = pp_pid(B,A,k,phi,Delta);
40
41 // parameters for simulation using g_s_cl
42 Tc = Sc; gamm = 1; N = 1;
43 C = 0; D = 1; N_var = 0;
44 st = 1; t_init = 0; t_final = 20;
45
46 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
47 [Np,Rcp] = cosfil_ip(N,Rc); // N/Rc
48 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
49 [Cp,Dp] = cosfil_ip(C,D); // C/D
50 Num = numer(G1);
51 Den = denom(G1);
```

Scilab code Exa 9.20 PID controller design

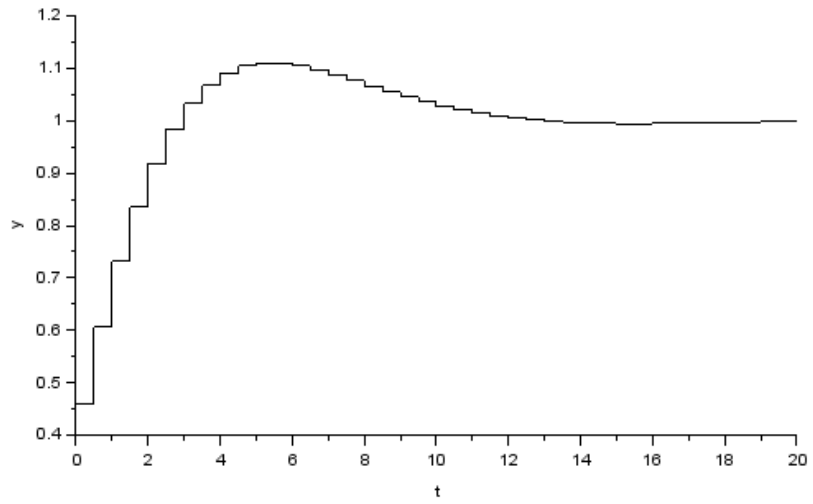

Figure 9.13: Demonstration of usefulness of negative PID parameters

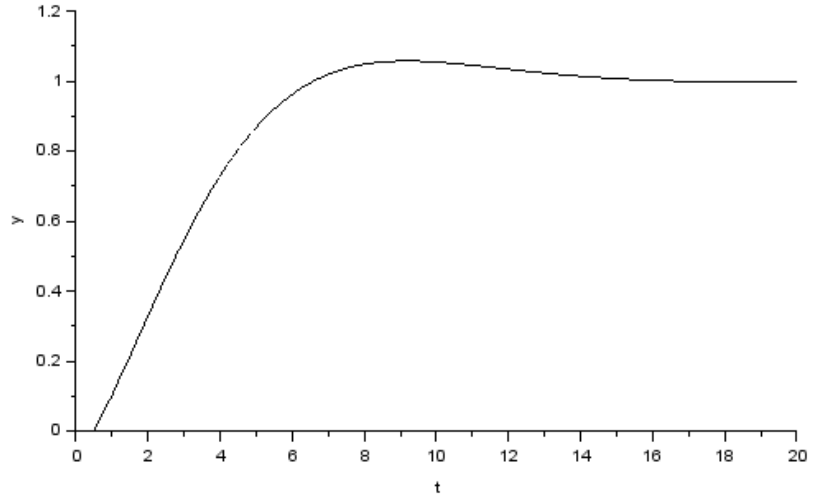

Figure 9.14: Demonstration of usefulness of negative PID parameters

```
1  // Solution to Aryabhatta's identity arising in PID
     controller design, namely Eq. 9.37 on page 363.
2  // 9.20
3
4  function [Rc,Sc] = pp_pid(B,A,k,phi,Delta)
5
6  // Setting up and solving Aryabhatta identity
7  dB = length(B) - 1; dA = length(A) - 1;
8  [zk,dzk] = zpowk(k);
9  [N,dN] = polmul(B,dB,zk,dzk);
10 dDelta = length(Delta)-1;
11 [D,dD] = polmul(A,dA,Delta,dDelta);
12 dphi = length(phi)-1;
13 [Sc,dSc,R,dR] = xdync(N,dN,D,dD,phi,dphi);
14 Rc = convol(R,Delta);
15 endfunction;
```

This code can be downloaded from the website wwww.scilab.in This code

can be downloaded from the website wwww.scilab.in

Scilab code Exa 9.21 DC motor with PID control tuned through pole placement technique

```
1  // DC motor with PID control, tuned through pole
     placement technique, as in Example 9.18.
2  // 9.21
3
4  exec('desired.sci',-1);
5  exec('pp_pid.sci',-1);
6  exec('cosfil_ip.sci',-1);
7  exec('pd.sci',-1);
8  exec('polyno.sci',-1);
```

```
exec('myc2d.sci',-1);
exec('zpowk.sci',-1);
exec('polmul.sci',-1);
exec('polsize.sci',-1);
exec('xdync.sci',-1);
exec('rowjoin.sci',-1);
exec('left_prm.sci',-1);
exec('t1calc.sci',-1);
exec('indep.sci',-1);
exec('seshft.sci',-1);
exec('makezero.sci',-1);
exec('move_sci.sci',-1);
exec('colsplit.sci',-1);
exec('clcoef.sci',-1);
exec('cindep.sci',-1);

// Motor control problem
// Transfer function

a = [-1 0; 1 0]; b = [1; 0]; c = [0 1]; d = 0;
G = syslin('c',a,b,c,d); Ts = 0.25;
[B,A,k] = myc2d(G,Ts);
[Ds,num,den] = ss2tf(G);

// Transient specifications
rise = 3; epsilon = 0.05;
phi = desired(Ts,rise,epsilon);

// Controller design
Delta = 1;  //No internal model of step used
[Rc,Sc] = pp_pid(B,A,k,phi,Delta);

// continuous time controller
[K,taud,N] = pd(Rc,Sc,Ts);
numb = K*[1 taud*(1+1/N)]; denb = [1 taud/N];
numf = 1; denf = 1;

// simulation parameters
```

```
47 st = 1;  // desired change in position
48 t_init = 0; // simulation start time
49 t_final = 20; // simulation end time
50 st1 = 0;
51
52 // continuous controller simulation: g_s_cl3.xcos
53 num1 = 0; den1 = 1;
54
55 // discrete controller simulation: g_s_cl2.xcos
56 // u1: -0.1 to 0.8
57 // y1: 0 to 1.4
58 C = 0; D = 1; N = 1; gamm = 1; Tc = Sc;
59
60 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
61 [Np,Rcp] = cosfil_ip(N,Rc); // N/Rc
62 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
63 [Cp,Dp] = cosfil_ip(C,D); // C/D
64 Numb = polyno(numb,'s');
65 Denb = polyno(denb,'s');
66 Numf = polyno(numf,'s');
67 Denf = polyno(denf,'s');
68 Num1 = polyno(num1,'s');
69 Den1 = polyno(den1,'s');
```

Scilab code Exa 9.22 PD control law from polynomial coefficients

```
1 // PD control law from polynomial coefficients, as
    explained in Sec. 9.8.
2 // 9.22
3
4 function [K,taud,N] = pd(Rc,Sc,Ts)
```

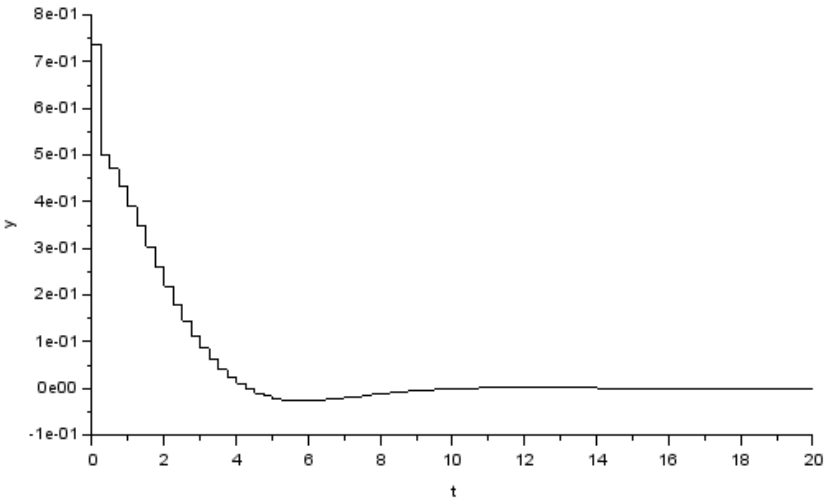

Figure 9.15: DC motor with PID control tuned through pole placement technique

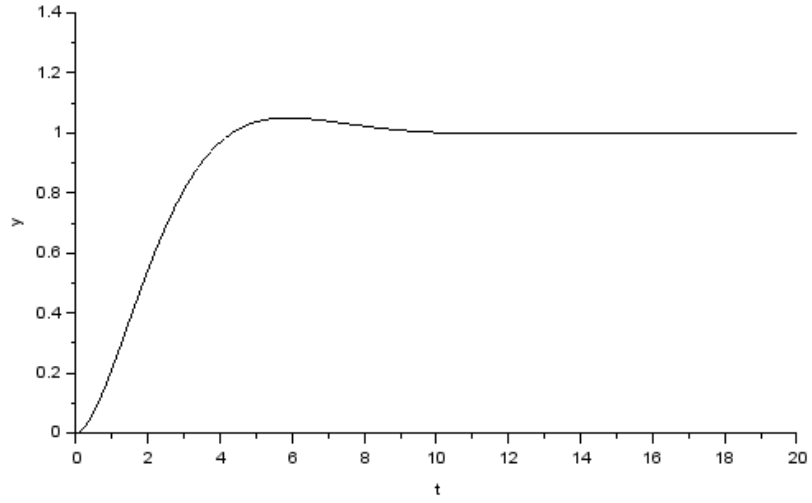

Figure 9.16: DC motor with PID control tuned through pole placement technique

```
5
6  // Both Rc and Sc have to be degree one polynomials
7
8  s0 = Sc(1); s1 = Sc(2);
9  r1 = Rc(2);
10 K = (s0+s1)/(1+r1);
11 N = (s1-s0*r1)/r1/(s0+s1);
12 taudbyN = -Ts*r1/(1+r1);
13 taud = taudbyN * N;
14 endfunction;
```

Chapter 10

Special Cases of Pole Placement Control

Scilab code Exa 10.1 Effect of delay in control performance

```
1  // Effect of delay in control performance
2  // 10.1
3
4  exec('zpowk.sci',-1);
5  exec('pp_im.sci',-1);
6  exec('cosfil_ip.sci',-1);
7  exec('polsplit3.sci',-1);
8  exec('polmul.sci',-1);
9  exec('polsize.sci',-1);
10 exec('xdync.sci',-1);
11 exec('rowjoin.sci',-1);
12 exec('left_prm.sci',-1);
13 exec('t1calc.sci',-1);
14 exec('indep.sci',-1);
15 exec('seshft.sci',-1);
16 exec('makezero.sci',-1);
17 exec('move_sci.sci',-1);
18 exec('colsplit.sci',-1);
19 exec('clcoef.sci',-1);
```

```
20 exec('cindep.sci',-1);
21 exec('polyno.sci',-1);
22
23 Ts = 1; B = 0.63; A = [1 -0.37];
24 k = input('Enter the delay as an integer: ');
25 if k<=0, k = 1; end
26 [zk,dzk] = zpowk(k);
27
28 // Desired transfer function
29 phi = [1 -0.5];
30 delta = 1; // internal model of step introduced
31
32 // Controller design
33 [Rc,Sc,Tc,gamm] = pp_im(B,A,k,phi,delta);
34
35 // simulation parameters for stb_disc.xcos
36 // y1: 0 to 1; u1: 0 to 1.2
37 st = 1.0; // desired change in setpoint
38 t_init = 0; // simulation start time
39 t_final = 20; // simulation end time
40
41 // simulation parameters for stb_disc.xcos
42 N_var = 0; C = 0; D = 1; N = 1;
43
44 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
45 [Rcp1,Rcp2] = cosfil_ip(1,Rc); // 1/Rc
46 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
47 [Bp,Ap] = cosfil_ip(B,A); // B/A
48 [zkp1,zkp2] = cosfil_ip(zk,1); // zk/1
49 [Cp,Dp] = cosfil_ip(C,D); // C/D
```

This code can be downloaded from the website wwww.scilab.in

Scilab code Exa 10.2 Smith predictor for paper machine control

```
// Smith predictor for paper machine control in
   Example 10.2 on page 385.
// 10.2

exec('zpowk.sci',-1);
exec('poladd.sci',-1);
exec('polsize.sci',-1);
exec('pp_im.sci',-1);
exec('polsplit3.sci',-1);
exec('polmul.sci',-1);
exec('xdync.sci',-1);
exec('rowjoin.sci',-1);
exec('left_prm.sci',-1);
exec('t1calc.sci',-1);
exec('indep.sci',-1);
exec('makezero.sci',-1);
exec('move_sci.sci',-1);
exec('colsplit.sci',-1);
exec('clcoef.sci',-1);
exec('cindep.sci',-1);
exec('seshft.sci',-1);
exec('cosfil_ip.sci',-1);
exec('polyno.sci',-1);

Ts = 1; B = 0.63; A = [1 -0.37]; k = 3;
Bd = convol(B,[0 1]);
kd = k - 1;
[zkd,dzkd] = zpowk(kd);
[mzkd,dmzkd] = poladd(1,0,-zkd,dzkd);

// Desired transfer function
phi = [1 -0.5]; delta = 1;

// Controller design
[Rc,Sc,Tc,gamm] = pp_im(B,A,1,phi,delta);

// simulation parameters for smith_disc.xcos
st = 1.0; // desired change in setpoint
```

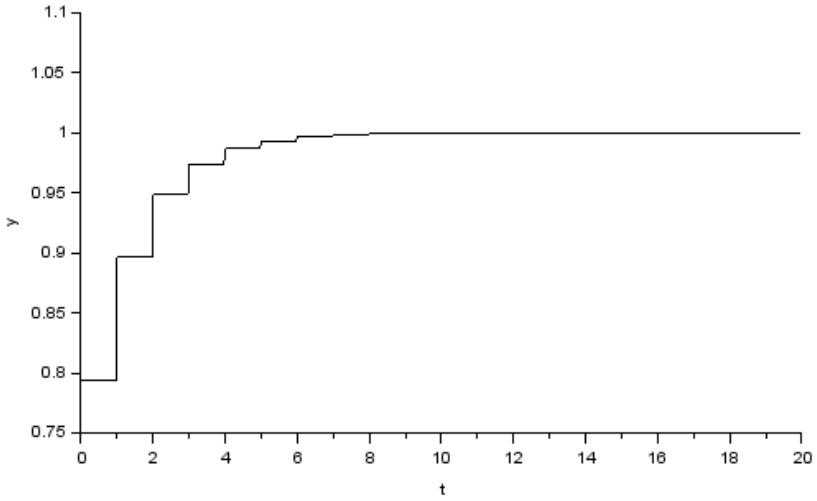

Figure 10.1: Smith predictor for paper machine control

```
38 t_init = 0; // simulation start time
39 t_final = 20; // simulation end time
40
41 // simulation parameters for smith_disc.xcos
42 N_var = 0; C = 0; D = 1; N = 1;
43
44 [Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
45 [Rcp1,Rcp2] = cosfil_ip(1,Rc); // 1/Rc
46 [Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
47 [Bdp,Ap] = cosfil_ip(Bd,A); // Bd/Ad
48 [zkdp1,zkdp2] = cosfil_ip(zkd,1); // zkd/1
49 [mzkdp1,mzkdp2] = cosfil_ip(mzkd,1); // mzkd/1
50 [Cp,Dp] = cosfil_ip(C,D); // C/D
```

This code can be downloaded from the website wwww.scilab.in

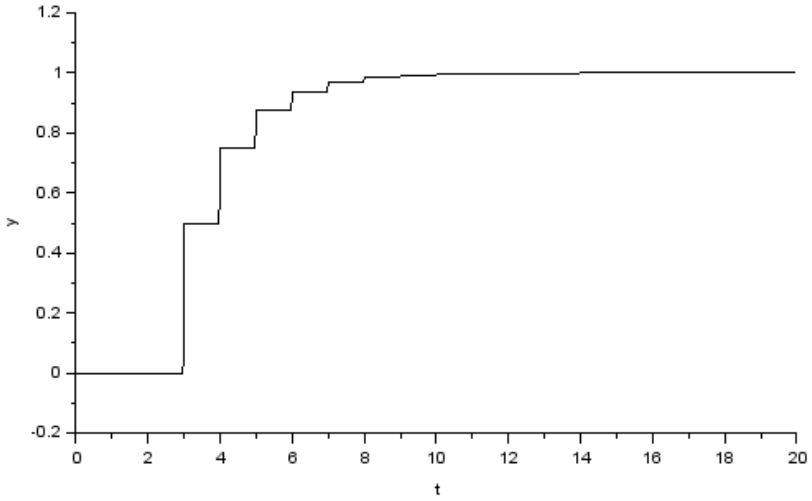

Figure 10.2: Smith predictor for paper machine control

Scilab code Exa 10.3 Splitting a polynomial B

```
1  // Splitting a polynomial B(z)
2  // 10.3
3  // Splits a polynomial B into good, nonminimum with
4  // positive real & with negative real parts.
5  // All are returned in polynomial form.
6  // Gain is returned in Kp and delay in k.
7
8  function [Kp,k,Bg,Bnmp,Bm] = imcsplit(B,polynomial)
9  k = 0;
10 Kp = 1;
11 if(polynomial)
12   rts = roots(B);
```

```
13    Kp = sum(B)/sum(coeff(poly(rts,'z')));
14  else
15    rts = B;
16  end
17  Bg = 1; Bnmp = 1; Bm = 1;
18  for i = 1:length(rts),
19      rt = rts(i);
20      if rt == 0,
21         k = k+1;
22      elseif (abs(rt)<1 & real(rt)>=0)
23        Bg = convol(Bg,[1 -rt]);
24      elseif (abs(rt)>=1 & real(rt)>=0)
25        Bnmp = convol(Bnmp,[1 -rt]);
26      else
27        Bm = convol(Bm,[1 -rt]);
28      end
29  end
```

Scilab code Exa 10.4 Design of internal model controller

```
1  // Design of internal model controller
2  // 10.4
3  // Designs Discrete Internal Model Controller
4  // for transfer function z^{-k}B(z^{-1})/A(z^{-1})
5  // Numerator and Denominator of IMC HQ are outputs
6  // Controller is also given in R,S form
7
8  function [k,HiN,HiD] = imc_stable1(B,A,k,alpha)
9
10 [Kp,d,Bg,Bnmp,Bm] = imcsplit(B,mtlb_logical(1));
11 Bg = Kp * Bg;
12 Bnmpr = flip(Bnmp);
13 Bms = sum(Bm);
14 HiN = A;
15 HiD = Bms * convol(Bg,Bnmpr);
```

```
16 k = k+d;
17 endfunction;
```

Scilab code Exa 10.5 Flipping a vector

```
1 // 10.5
2 function b = flip(a)
3 b = a(length(a):-1:1);
4 endfunction;
```

Scilab code Exa 10.6 IMC design for viscosity control problem

```
1 // IMC design for viscosity control problem
2 // 10.6
3
4 exec('imc_stable1.sci',-1);
5 exec('zpowk.sci',-1);
6 exec('imcsplit.sci',-1);
7 exec('flip.sci',-1);
8
9 B = [0.51 1.21];
10 A = [1 -0.44];
11 k = 1;
12 alpha = 0.5;
13
14 [k,GiN,GiD] = imc_stable1(B,A,k,alpha);
15
16 [zk,dzk] = zpowk(k);
17 Bp = B; Ap = A;
18 Ts = 0.1; t0 = 0; tf = 20; Nvar = 0.01;
```

Scilab code Exa 10.7 IMC design for the control of van de Vusse reactor

```
1 // IMC design for the control of van de Vusse
     reactor
2 // 10.7
3
4 exec('tf.sci');
5 exec('myc2d.sci');
6 exec('imc_stable1.sci');
7 exec('imcsplit.sci',-1);
8 exec('flip.sci',-1);
9 exec('zpowk.sci',-1);
10
11 num = [-1.117 3.1472]; den = [1 4.6429 5.3821];
12 G = tf(num,den);
13 Ts = 0.1;
14 [B,A,k] = myc2d(G,Ts);
15 alpha = 0.9;
16 [k,GiN,GiD] = imc_stable1(B,A,k,alpha);
17 [zk,dzk] = zpowk(k);
18 Bp = B; Ap = A;
19 t0 = 0; tfi = 10; st = 1; Nvar = 0;
```

Scilab code Exa 10.8 IMC design for an example by Lewin

```
1 // IMC design for Lewin's example
2 // 10.8
3
4 exec('tf.sci');
5 exec('myc2d.sci');
6 exec('imc_stable1.sci');
7 exec('zpowk.sci',-1);
8 exec('imcsplit.sci',-1);
9 exec('flip.sci',-1);
10
```

```
num = 1; den = [250 35 1]; Ts = 3;
G = tf(num,den);

[B,A,k] = myc2d(G,Ts);

alpha = 0.9;
[k,GiN,GiD] = imc_stable1(B,A,k,alpha);

[zk,dzk] = zpowk(k);
Bp = B; Ap = A;
t0 = 0; tfi = 100; st = 1; Nvar = 0;
```

Scilab code Exa 10.9 Design of conventional controller which is an equivalent of internal model controller

```
// Design of conventional controller which is an
   equivalent of internal model controller
// 10.9

// Designs Discrete Internal Model Controller
// for transfer function z^{-k}B(z^{-1})/A(z^{-1})
// Numerator and Denominator of IMC HQ are outputs
// Controller is also given in R,S form


function [k,HiN,HiD,R,S,mu] = imc_stable(B,A,k,alpha
   )

[Kp,d,Bg,Bnmp,Bm] = imcsplit(B,mtlb_logical(1));
Bg = Kp * Bg;

Bnmpr = flip(Bnmp);
Bms = sum(Bm);
HiN = A;
HiD = Bms * convol(Bg,Bnmpr);
```

```
19 k = k+d;
20
21 [zk,dzk] = zpowk(k);
22 Bf = (1-alpha);
23 Af = [1 -alpha];
24 S = convol(Bf,A);
25 R1 = convol(Af,convol(Bnmpr,Bms));
26 R2 = convol(zk,convol(Bf,convol(Bnmp,Bm)));
27
28 [R,dR] = poladd(R1,length(R1)-1,-R2,length(R2)-1);
29 R = convol(Bg,R);
30 endfunction;
```

Scilab code Exa 10.10 Design of conventional controller for van de Vusse reactor problem

```
1 // Design of conventional controller for van de
     Vusse reactor problem
2 // 10.10
3
4 exec('tf.sci');
5 exec('myc2d.sci');
6 exec('imcsplit.sci',-1);
7 exec('imc_stable.sci');
8 exec('zpowk.sci',-1);
9 exec('flip.sci',-1);
10 exec('poladd.sci',-1);
11 exec('polsize.sci',-1);
12
13 num = [-1.117 3.1472]; den = [1 4.6429 5.3821];
14 G = tf(num,den);
15 Ts = 0.1;
16 [B,A,k] = myc2d(G,Ts);
17 alpha = 0.5;
18 [k,HiN,HiD,R,S] = imc_stable(B,A,k,alpha);
```

```
19 [zk,dzk] = zpowk(k);
20 Bp = B; Ap = A;
```

Chapter 11

Minimum Variance Control

Scilab code Exa 11.1 Recursive computation of Ej and Fj

```
1  // Recursive computation of Ej and Fj
2  // 11.1
3
4  function [Fj,dFj,Ej,dEj] = recursion(A,dA,C,dC,j)
5  Fo = C; dFo = dC;
6  Eo = 1; dEo = 0;
7  A_z = A(2:dA+1); dA_z = dA-1;
8  zi = 1; dzi = 0;
9  for i = 1:j-1
10     if (dFo == 0)
11         Fn1 = 0;
12     else
13         Fn1 = Fo(2:(dFo+1));
14     end
15     dFn1 = max(dFo-1,0);
16     Fn2 = -Fo(1)*A_z;  dFn2 = dA-1;
17     [Fn,dFn] = poladd(Fn1,dFn1,Fn2,dFn2);
18     zi = convol(zi,[0,1]); dzi = dzi + 1;
19     En2 = Fn(1)*zi; dEn2 = dzi;
20     [En,dEn] = poladd(Eo,dEo,En2,dEn2);
21     Eo = En; Fo = Fn;
```

```
22      dEo = dEn; dFo = dFn;
23  end
24  if (dFo == 0)
25      Fn1 = 0;
26  else
27  Fn1 = Fo(2:(dFo+1));
28  end;
29  dFn1 = max(dFo-1,0);
30  Fn2 = -Fo(1)*A_z;  dFn2 = dA-1;
31  [Fn,dFn] = poladd(Fn1,dFn1,Fn2,dFn2);
32  Fj = Fn; dFj = dFn;
33  Ej = Eo; dEj = dEo;
34  endfunction;
```

Scilab code Exa 11.2 Recursive computation of Ej and Fj for the system presented in Example

```
1  // Recursive computation of Ej and Fj for the system
       presented in Example 11.2 on page 408.
2  // 11.2
3
4  exec('poladd.sci',-1);
5  exec('polsize.sci',-1);
6  exec('recursion.sci',-1);
7
8  C = [1 0.5]; dC = 1;
9  A = [1 -0.6 -0.16]; dA = 2;
10 j = 2;
11 [Fj,dFj,Ej,dEj] = recursion(A,dA,C,dC,j)
```

Scilab code Exa 11.3 Solution of Aryabhatta identity

```
1  // Solution of Aryabhatta's identity Eq. 11.8, as
       discussed in Example 11.3 on page 409.
2  // 11.3
3
4  exec('xdync.sci',-1);
5  exec('rowjoin.sci',-1);
6  exec('polsize.sci',-1);
7  exec('left_prm.sci',-1);
8  exec('t1calc.sci',-1);
9  exec('indep.sci',-1);
10 exec('seshft.sci',-1);
11 exec('makezero.sci',-1);
12 exec('move_sci.sci',-1);
13 exec('colsplit.sci',-1);
14 exec('clcoef.sci',-1);
15 exec('cindep.sci',-1);
16
17 C = [1 0.5]; dC = 1; j=2;
18 A = [1 -0.6 -0.16]; dA = 2;
19 zj = zeros(1,j+1); zj(j+1) = 1;
20 [Fj,dFj,Ej,dEj] = xdync(zj,j,A,dA,C,dC)
```

Scilab code Exa 11.4 1st control problem by MacGregor

```
1  // MacGregor's first control problem, discussed in
       Example 11.4 on page 213.
2  // 11.4
3
4  exec('mv.sci',-1);
5  exec('cl.sci',-1);
6  exec('cosfil_ip.sci',-1);
7  exec('zpowk.sci',-1);
8  exec('xdync.sci',-1);
9  exec('rowjoin.sci',-1);
10 exec('polsize.sci',-1);
```

```
exec('left_prm.sci',-1);
exec('t1calc.sci',-1);
exec('indep.sci',-1);
exec('seshft.sci',-1);
exec('makezero.sci',-1);
exec('move_sci.sci',-1);
exec('colsplit.sci',-1);
exec('clcoef.sci',-1);
exec('cindep.sci',-1);
exec('polmul.sci',-1);
exec('poladd.sci',-1);
exec('tfvar.sci',-1);
exec('l2r.sci',-1);
exec('transp.sci',-1);
exec('tf.sci',-1);
exec('covar_m.sci',-1);
exec('polyno.sci',-1);

// MacGregor's first control problem
A = [1 -1.4 0.45]; dA = 2; C = [1 -0.5]; dC = 1;
B = 0.5*[1 -0.9]; dB = 1; k = 1; int1 = 0;
[Sc,dSc,Rc,dRc] = mv(A,dA,B,dB,C,dC,k,int1);
[Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] = ...
cl(A,dA,B,dB,C,dC,k,Sc,dSc,Rc,dRc,int1);

// Simulation parameters for stb_disc.xcos
Tc = Sc; gamm = 1; [zk,dzk] = zpowk(k);
D = 1; N_var = 1; Ts = 1; st = 0;
t_init = 0; t_final = 1000;

[Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
[Rcp1,Rcp2] = cosfil_ip(1,Rc); // 1/Rc
[Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
[Bp,Ap] = cosfil_ip(B,A); // B/A
[zkp1,zkp2] = cosfil_ip(zk,1); // zk/1
[Cp,Dp] = cosfil_ip(C,D); // C/D
```

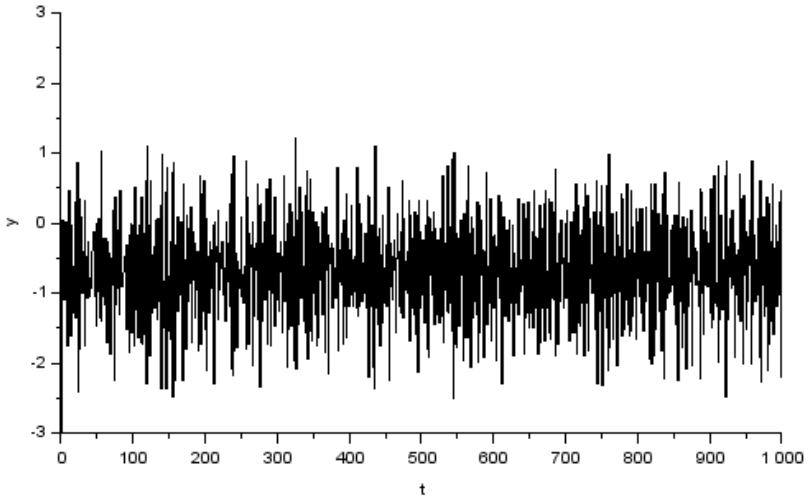

Figure 11.1: 1st control problem by MacGregor

This code can be downloaded from the website wwww.scilab.in

Scilab code Exa 11.5 Minimum variance control law design

```
1 // Minimum variance control law design, given by Eq.
     11.40 on page 413.
2 // 11.5
3
4 // function [S,dS,R,dR] = mv(A,dA,B,dB,C,dC,k,int)
5 // implements the minimum variance controller
6 // if int>=1, integrated noise is assumed; otherwise
    ,
```

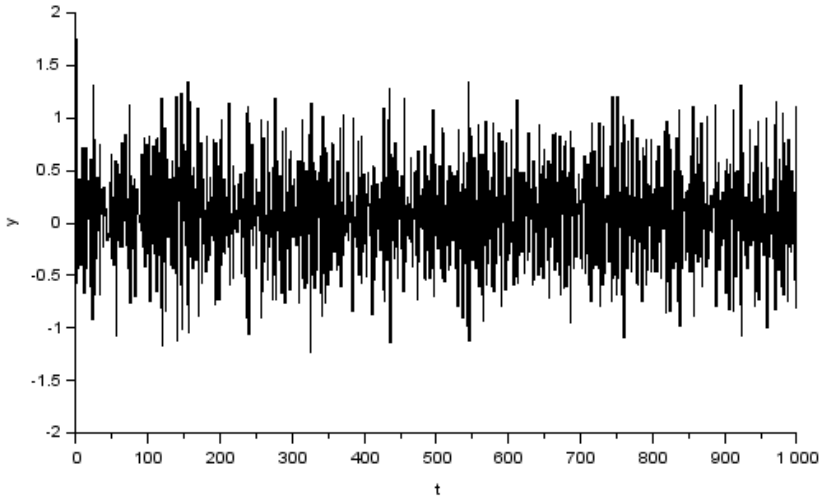

Figure 11.2: 1st control problem by MacGregor

```
7  // it is not integrated noise
8
9  function [S,dS,R,dR] = mv(A,dA,B,dB,C,dC,k,int1)
10 zk = zeros(1,k+1); zk(k+1) = 1;
11 if int1>=1, [A,dA] = polmul([1 -1],1,A,dA); end
12 [Fk,dFk,Ek,dEk] = xdync(zk,k,A,dA,C,dC);
13
14 [Gk,dGk] = polmul(Ek,dEk,B,dB);
15 S = Fk; dS = dFk; R = Gk; dR = dGk;
16 endfunction;
```

Scilab code Exa 11.6 Calculation of closed loop transfer functions

```
1 // Calculation of closed loop transfer functions
2 // 11.6
3
4 // function [Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar]
    = ...
```

```
5  //     cl(A,dA,B,dB,C,dC,k,S,dS,R,dR,int)
6  // int>=1 means integrated noise and control law:
7  // delta u = - (S/R)y
8  // Evaluates the closed loop transfer function and
9  // variances of input and output
10
11 function [Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] =
       ...
12     cl(A,dA,B,dB,C,dC,k,S,dS,R,dR,int1)
13 [zk,dzk] = zpowk(k);
14
15 [BS,dBS] = polmul(B,dB,S,dS);
16 [zBS,dzBS] = polmul(zk,dzk,BS,dBS);
17 [RA,dRA] = polmul(R,dR,A,dA);
18 if int1>=1, [RA,dRA] = polmul(RA,dRA,[1 -1],1); end
19
20 [D,dD] = poladd(RA,dRA,zBS,dzBS);
21
22 [Ny,dNy] = polmul(C,dC,R,dR);
23 [Nu,dNu] = polmul(C,dC,S,dS);
24
25 [Nu,dNu,Du,dDu,uvar] = tfvar(Nu,dNu,D,dD);
26 [Ny,dNy,Dy,dDy,yvar] = tfvar(Ny,dNy,D,dD);
27
28 endfunction;
```

Scilab code Exa 11.7 Cancellation of common factors and determination of covariance

```
1  // Cancellation of common factors and determination
       of covariance
2  // 11.7
3
4  // function [N,dN,D,dD,yvar] = tfvar(N,dN,D,dD)
5  // N and D polynomials in z^{-1} form; discrete case
```

```
6
7  function [N,dN,D,dD,yvar] = tfvar(N,dN,D,dD)
8
9  [N,dN,D,dD] = l2r(N,dN,D,dD);
10 N = N/D(1); D = D/D(1);
11 LN = length(N); LD = length(D);
12 D1 = D;
13 if LD<LN, D1 = [D zeros(1,LN-LD)]; dD1 = dD+LN-LD;
      end
14 H = tf(N,D1,1);//TS=1 (sampling time) has been taken
       constant in tfvar
15 yvar = covar_m(H,1);
16 endfunction;
```

Scilab code Exa 11.8 Computing sum of squares

```
1  // Computing sum of squares, as presented in Example
       11.5 on page 415.
2  // 11.8
3
4  exec('tf.sci',-1);
5  exec('covar_m.sci',-1);
6
7  Y = tf([1 0],[1 -0.9],-1);
8  covar_m(Y,1)
```

Scilab code Exa 11.9 Minimum variance control for nonminimum phase systems

```
1  // Minimum variance control for nonminimum phase
       systems
2  // 11.9
3
```

```
4 // function [Sc,dSc,Rc,dRc] = mv_mv(A,dA,B,dB,C,dC,k
    ,int)
5 // implements the minimum variance controller
6 // if int>=1, integrated noise is assumed; otherwise
    ,
7 // it is not integrated noise
8
9 function [Sc,dSc,Rc,dRc] = mv_nm(A,dA,B,dB,C,dC,k,
    int1)
10 if int1>=1, [A,dA] = polmul([1 -1],1,A,dA); end
11 [zk,dzk] = zpowk(k);
12 [Bzk,dBzk] = polmul(B,dB,zk,dzk);
13 [Bg,Bb] = polsplit3(B); Bbr = flip(Bb);
14 RHS = convol(C,convol(Bg,Bbr)); dRHS = length(RHS)
    -1;
15 [Sc,dSc,Rc,dRc] = xdync(Bzk,dBzk,A,dA,RHS,dRHS);
16 endfunction;
```

Scilab code Exa 11.10 Minimum variance control for nonminimum phase example

```
1 // Minimum variance control for nonminimum phase
    example of Example 11.6 on page 416.
2 // 11.10
3
4 exec('mv_nm.sci',-1);
5 exec('cl.sci',-1);
6 exec('zpowk.sci',-1);
7 exec('polmul.sci',-1);
8 exec('polsize.sci',-1);
9 exec('polsplit3.sci',-1);
10 exec('flip.sci',-1);
11 exec('xdync.sci',-1);
12 exec('rowjoin.sci',-1);
13 exec('left_prm.sci',-1);
```

```
14 exec('t1calc.sci',-1);
15 exec('indep.sci',-1);
16 exec('seshft.sci',-1);
17 exec('makezero.sci',-1);
18 exec('move_sci.sci',-1);
19 exec('colsplit.sci',-1);
20 exec('clcoef.sci',-1);
21 exec('cindep.sci',-1);
22 exec('poladd.sci',-1);
23 exec('tfvar.sci',-1);
24 exec('l2r.sci',-1);
25 exec('transp.sci',-1);
26 exec('tf.sci',-1);
27 exec('covar_m.sci',-1);
28
29 A = convol([1 -1],[1 -0.7]); dA = 2;
30 B = [0.9 1]; dB = 1; k = 1;
31 C = [1 -0.7]; dC = 1; int1 = 0;
32 [Sc,dSc,Rc,dRc] = mv_nm(A,dA,B,dB,C,dC,k,int1);
33 [Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] = ...
34 cl(A,dA,B,dB,C,dC,k,Sc,dSc,Rc,dRc,int1);
```

Scilab code Exa 11.11 Minimum variance control of viscosity control problem

```
1 // Minimum variance control of viscosity control
    problem
2 // 11.11
3
4 // Viscosity control problem of MacGregor
5
6 exec('mv_nm.sci',-1);
7 exec('polmul.sci',-1);
8 exec('polsize.sci',-1);
9 exec('zpowk.sci',-1);
```

```
10 exec('polsplit3.sci',-1);
11 exec('flip.sci',-1);
12 exec('xdync.sci',-1);
13 exec('rowjoin.sci',-1);
14 exec('left_prm.sci',-1);
15 exec('t1calc.sci',-1);
16 exec('indep.sci',-1);
17 exec('seshft.sci',-1);
18 exec('makezero.sci',-1);
19 exec('move_sci.sci',-1);
20 exec('colsplit.sci',-1);
21 exec('clcoef.sci',-1);
22 exec('cindep.sci',-1);
23 exec('cl.sci',-1);
24 exec('poladd.sci',-1);
25 exec('tfvar.sci',-1);
26 exec('l2r.sci',-1);
27 exec('transp.sci',-1);
28 exec('tf.sci',-1);
29 exec('covar_m.sci',-1);
30
31 A = [1 -0.44]; dA = 1; B = [0.51 1.21]; dB = 1;
32 C = [1 -0.44]; dC = 1; k = 1; int1 = 1;
33 [Sc,dSc,Rc,dRc] = mv_nm(A,dA,B,dB,C,dC,k,int1);
34 [Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] = ...
35 cl(A,dA,B,dB,C,dC,k,Sc,dSc,Rc,dRc,int1);
```

Scilab code Exa 11.12 General minimum variance controller design

```
1 // General minimum variance controller design, as
    given by Eq. 11.66 on page 421 and Eq. 11.70 on
    page 422.
2 // 11.12
3
4 // function [Sc,dSc,Rc,dRc] = gmv(A,dA,B,dB,C,dC,k,
```

```
    rho,int)
// implements the generalized minimum variance
    controller
// if int>=1, integrated noise is assumed; otherwise
    ,
// it is not integrated noise

function [Sc,dSc,R,dR] = gmv(A,dA,B,dB,C,dC,k,rho,
    int1)
zk = zeros(1,k+1); zk(k+1) = 1;
if int1>=1, [A,dA] = polmul([1 -1],1,A,dA); end
[Fk,dFk,Ek,dEk] = xdync(zk,k,A,dA,C,dC);
[Gk,dGk] = polmul(Ek,dEk,B,dB);
alpha0 = Gk(1)/C(1);
Sc = alpha0 * Fk; dSc = dFk;
[R,dR] = poladd(alpha0*Gk,dGk,rho*C,dC);
endfunction;
```

Scilab code Exa 11.13 GMVC design of first example by MacGregor

```
// GMVC design of MacGregor's first example, as
    discussed in Example 11.9 on page 421.
// 11.13

// MacGregor's first control problem by gmv

exec('gmv.sci',-1);
exec('cl.sci',-1);
exec('xdync.sci',-1);
exec('rowjoin.sci',-1);
exec('polsize.sci',-1);
exec('left_prm.sci',-1);
exec('t1calc.sci',-1);
exec('indep.sci',-1);
exec('seshft.sci',-1);
```

```
15 exec('makezero.sci',-1);
16 exec('move_sci.sci',-1);
17 exec('colsplit.sci',-1);
18 exec('clcoef.sci',-1);
19 exec('cindep.sci',-1);
20 exec('polmul.sci',-1);
21 exec('zpowk.sci',-1);
22 exec('poladd.sci',-1);
23 exec('tfvar.sci',-1);
24 exec('l2r.sci',-1);
25 exec('transp.sci',-1);
26 exec('tf.sci',-1);
27 exec('covar_m.sci',-1);
28
29 A = [1 -1.4 0.45]; dA = 2; C = [1 -0.5]; dC = 1;
30 B = 0.5*[1 -0.9]; dB = 1; k = 1; int1 = 0;
31 rho = 1;
32 [Sc,dSc,Rc,dRc] = gmv(A,dA,B,dB,C,dC,k,rho,int1);
33 [Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] = ...
34          cl(A,dA,B,dB,C,dC,k,Sc,dSc,Rc,dRc,int1);
```

Scilab code Exa 11.14 GMVC design of viscosity problem

```
1 // GMVC design of viscosity problem, as described in
      Example 11.10 on page 423.
2 // 11.14
3
4 // MacGregor's Viscosity control problem by gmv
5
6 exec('gmv.sci',-1);
7 exec('cl.sci',-1);
8 exec('polmul.sci',-1);
9 exec('polsize.sci',-1);
10 exec('xdync.sci',-1);
11 exec('rowjoin.sci',-1);
```

```
12 exec('left_prm.sci',-1);
13 exec('t1calc.sci',-1);
14 exec('indep.sci',-1);
15 exec('seshft.sci',-1);
16 exec('makezero.sci',-1);
17 exec('move_sci.sci',-1);
18 exec('colsplit.sci',-1);
19 exec('clcoef.sci',-1);
20 exec('cindep.sci',-1);
21 exec('poladd.sci',-1);
22 exec('zpowk.sci',-1);
23 exec('tfvar.sci',-1);
24 exec('l2r.sci',-1);
25 exec('transp.sci',-1);
26 exec('tf.sci',-1);
27 exec('covar_m.sci',-1);
28
29 A = [1 -0.44]; dA = 1; B = [0.51 1.21]; dB = 1;
30 C = [1 -0.44]; dC = 1; k = 1; int1 = 1;
31 rho = 1;
32 [Sc,dSc,R1,dR1] = gmv(A,dA,B,dB,C,dC,k,rho,int1);
33 [Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] = ...
34          cl(A,dA,B,dB,C,dC,k,Sc,dSc,R1,dR1,int1);
```

Scilab code Exa 11.15 PID tuning through GMVC law

```
1 // PID tuning through GMVC law, as discussed in
     Example 11.11.
2 // 11.15
3
4 exec('gmvc_pid.sci',-1);
5 exec('zpowk.sci',-1);
6 exec('ch_pol.sci',-1);
7 exec('polmul.sci',-1);
8 exec('polsize.sci',-1);
```

```
exec('xdync.sci',-1);
exec('rowjoin.sci',-1);
exec('left_prm.sci',-1);
exec('t1calc.sci',-1);
exec('indep.sci',-1);
exec('seshft.sci',-1);
exec('makezero.sci',-1);
exec('move_sci.sci',-1);
exec('colsplit.sci',-1);
exec('clcoef.sci',-1);
exec('cindep.sci',-1);
exec('filtval.sci',-1);
exec('polyno.sci',-1);

// GMVC PID tuning of example given by Miller et al.
// Model
A = [1 -1.95 0.935]; B = -0.015; k = 1; Ts = 1;

// Transient specifications
N = 15; epsilon = 0.1;
T = ch_pol(N,epsilon);

// Controller Design
[Kc,tau_i,tau_d,L] = gmvc_pid(A,B,k,T,Ts);
L1 = filtval(L,1);
zk = zpowk(k);
```

Scilab code Exa 11.16 Value of polynomial p evaluated at x

```
// Value of polynomial p(x), evaluated at x
// 11.16

// finds the value of a polynomial in powers of z
   ^{-1}
// function Y = filtval(P,z)
```

```
6
7  function Y = filtval(P,z)
8  N = length(P)-1;
9  Q = polyno(P,'x');
10 Y = horner(Q,z)/z^N;
11 endfunction;
```

Scilab code Exa 11.17 PID tuning through GMVC law

```
1  // PID tuning through GMVC law
2  // 11.17
3
4  // function [Kc,tau_i,tau_d,L] = gmvc_pid(A,B,k,T,Ts
       )
5  // Determines p,i,d tuning parameters using GMVC
6  // Plant model: Integrated white noise
7  // A, B in discrete time form
8
9  function [Kc,tau_i,tau_d,L] = gmvc_pid(A,B,k,T,Ts)
10
11 dA = length(A)-1; dB = length(B)-1;
12 dT = length(T)-1;
13 if dA > 2,
14    disp('degree of A cannot be more than 2')
15    exit
16 elseif dB > 1,
17    disp('degree of B cannot be more than 1')
18    exit
19 elseif dT > 2,
20    disp('degree of T cannot be more than 2')
21    exit
22 end
23 delta = [1 -1]; ddelta = 1;
24
25 [Adelta,dAdelta] = polmul(A,dA,delta,ddelta);
```

```
26
27 [Q,dQ,P,dP] = ...
28 xdync(Adelta,dAdelta,B,dB,T,dT);
29 PAdelta = P(1)*Adelta;
30
31 [zk,dzk] = zpowk(k);
32 [E,degE,F,degF] = ...
33 xdync(PAdelta,dAdelta,zk,dzk,P,dP);
34 nu = P(1)*E(1)*B(1);
35 Kc = -1/nu*(F(2)+2*F(3));
36 tau_i = -(F(2)+2*F(3))/(F(1)+F(2)+F(3))*Ts;
37 tau_d = -F(3)/(F(2)+2*F(3))*Ts;
38 L(1) = 1+Ts/tau_i+tau_d/Ts;
39 L(2) = -(1+2*tau_d/Ts);
40 L(3) = tau_d/Ts;
41 L = Kc * L';
42 endfunction;
```

Chapter 12

Model Predictive Control

Scilab code Exa 12.1 Model derivation for GPC design

```
1 // Model derivation for GPC design in Example 12.1
    on page 439.
2 // 12.1
3
4 exec('xdync.sci',-1);
5 exec('polmul.sci',-1);
6 exec('flip.sci',-1);
7 exec('rowjoin.sci',-1);
8 exec('polsize.sci',-1);
9 exec('left_prm.sci',-1);
10 exec('t1calc.sci',-1);
11 exec('indep.sci',-1);
12 exec('seshft.sci',-1);
13 exec('makezero.sci',-1);
14 exec('move_sci.sci',-1);
15 exec('colsplit.sci',-1);
16 exec('clcoef.sci',-1);
17 exec('cindep.sci',-1);
18
19 // Camacho and Bordon's GPC example; model formation
20
```

```
A=[1 -0.8]; dA=1; B=[0.4 0.6]; dB=1; N=3; k=1;
D=[1 -1]; dD=1; AD=convol(A,D); dAD=dA+1; Nu=N+k;
zj = 1; dzj = 0; G = zeros(Nu);
H1 = zeros(Nu,k-1+dB); H2 = zeros(Nu,dA+1);

for j = 1:Nu,
    zj = convol(zj,[0,1]); dzj = dzj + 1;
    [Fj,dFj,Ej,dEj] = xdync(zj,dzj,AD,dAD,1,0);
    [Gj,dGj] = polmul(B,dB,Ej,dEj);
    G(j,1:dGj) = flip(Gj(1:dGj));
    H1(j,1:k-1+dB) = Gj(dGj+1:dGj+k-1+dB);
    H2(j,1:dA+1) = Fj;
end

G,H1,H2
```

Scilab code Exa 12.2 Calculates the GPC law

```
// Calculates the GPC law given by Eq. 12.19 on page
    441.
// 12.2

function [K,KH1,KH2,Tc,dTc,Sc,dSc,R1,dR1] = ...
gpc_bas(A,dA,B,dB,N,k,rho)
D=[1 -1]; dD=1; AD=convol(A,D); dAD=dA+1; Nu=N+1;
zj = 1; dzj = 0; G = zeros(Nu,Nu);
H1 = zeros(Nu,k-1+dB); H2 = zeros(Nu,dA+1);
for j = 1:Nu,
    zj = convol(zj,[0,1]); dzj = dzj + 1;
    [Fj,dFj,Ej,dEj] = xdync(zj,dzj,AD,dAD,1,0);
    [Gj,dGj] = polmul(B,dB,Ej,dEj);
    G(j,1:dGj) = flip(Gj(1:dGj));
    H1(j,1:k-1+dB) = Gj(dGj+1:dGj+k-1+dB);
    H2(j,1:dA+1) = Fj;
end
```

```
17 K = inv(G'*G+rho*eye(Nu,Nu))*G';
18 // Note: inverse need not be calculated
19 KH1 = K * H1; KH2 = K * H2;
20 R1 = [1 KH1(1,:)]; dR1 = length(R1)-1;
21 Sc = KH2(1,:); dSc = length(Sc)-1;
22 Tc = K(1,:); dTc = length(Tc)-1;
23 endfunction;
```

Scilab code Exa 12.3 GPC design for the problem discussed on page 441

```
1 // GPC design for the problem discussed in Example
     12.2 on page 441.
2 // 12.3
3
4 exec('gpc_bas.sci',-1);
5 exec('xdync.sci',-1);
6 exec('rowjoin.sci',-1);
7 exec('polsize.sci',-1);
8 exec('left_prm.sci',-1);
9 exec('t1calc.sci',-1);
10 exec('indep.sci',-1);
11 exec('seshft.sci',-1);
12 exec('makezero.sci',-1);
13 exec('move_sci.sci',-1);
14 exec('colsplit.sci',-1);
15 exec('clcoef.sci',-1);
16 exec('cindep.sci',-1);
17 exec('polmul.sci',-1);
18 exec('flip.sci',-1);
19 exec('filtval.sci',-1);
20
21 // Camacho and Bordon's GPC example; Control law
22 A=[1 -0.8]; dA=1; B=[0.4 0.6]; dB=1; N=3; k=1; rho
     =0.8;
23 [K,KH1,KH2,Tc,dTc,Sc,dSc,R1,dR1] = ...
```

```
24 gpc_bas(A,dA,B,dB,N,k,rho)
25 // C=1; dC=0; [K,KH1,KH2,Tc,dTc,Sc,dSc,R1,dR1] = ...
26 //  gpc_col(A,dA,B,dB,C,dC,N,k,rho)
```

Scilab code Exa 12.4 GPC design

```
1  // GPC design for the problem discussed in Example
       12.3.
2  // 12.4
3
4  exec('gpc_N.sci',-1);
5  exec('xdync.sci',-1);
6  exec('rowjoin.sci',-1);
7  exec('polsize.sci',-1);
8  exec('left_prm.sci',-1);
9  exec('t1calc.sci',-1);
10 exec('indep.sci',-1);
11 exec('seshft.sci',-1);
12 exec('makezero.sci',-1);
13 exec('move_sci.sci',-1);
14 exec('colsplit.sci',-1);
15 exec('clcoef.sci',-1);
16 exec('cindep.sci',-1);
17 exec('polmul.sci',-1);
18 exec('flip.sci',-1);
19
20 A=[1 -0.8]; dA=1; B=[0.4 0.6]; dB=1;
21 rho = 0.8; k = 1;
22 N1 = 0; N2 = 3; Nu = 2;
23
24 [K,KH1,KH2,Tc,dTc,Sc,dSc,R1,dR1] = ...
25 gpc_N(A,dA,B,dB,k,N1,N2,Nu,rho)
```

Scilab code Exa 12.5 Calculates the GPC law

```
1  // Calculates the GPC law given by Eq. 12.36 on page
       446.
2  // 12.5
3
4  function [K,KH1,KH2,Tc,dTc,Sc,dSc,R1,dR1] = ...
5  gpc_N(A,dA,B,dB,k,N1,N2,Nu,rho)
6  D=[1 -1]; dD=1; AD=convol(A,D); dAD=dA+1;
7  zj = 1; dzj = 0;
8  for i = 1:N1+k-1
9      zj = convol(zj,[0,1]); dzj = dzj + 1;
10 end
11 G = zeros(N2-N1+1,Nu+1);
12 H1 = zeros(N2-N1+1,k-1+dB); H2 = zeros(N2-N1+1,dA+1)
       ;
13 for j = k+N1:k+N2
14     zj = convol(zj,[0,1]); dzj = dzj + 1;
15     [Fj,dFj,Ej,dEj] = xdync(zj,dzj,AD,dAD,1,0);
16     [Gj,dGj] = polmul(B,dB,Ej,dEj);
17     if (j-k >= Nu)
18     G(j-(k+N1-1),1:Nu+1) = flip(Gj(j-k-Nu+1:j-k+1));
19 else
20     G(j-(k+N1-1),1:j-k+1) = flip(Gj(1:j-k+1));
21 end
22     H1(j-(k+N1-1),1:k-1+dB) = Gj(j-k+2:j+dB);
23     H2(j-(k+N1-1),1:dA+1) = Fj;
24 end
25 K = inv(G'*G+rho*eye(Nu+1,Nu+1))*G';
26 // Note: inverse need not be calculated
27 KH1 = K * H1; KH2 = K * H2;
28 R1 = [1 KH1(1,:)]; dR1 = length(R1)-1;
29 Sc = KH2(1,:); dSc = length(Sc)-1;
30 Tc = K(1,:); dTc = length(Tc)-1;
31 endfunction;
```

Scilab code Exa 12.6 Calculates the GPC law

```
1  // Calculates the GPC law given by Eq. 12.36 on page
       446.
2  // 12.6
3
4  function [K,KH1,KH2,Tc,dTc,Sc,dSc,R1,dR1] = ...
5  gpc_col(A,dA,B,dB,C,dC,N,k,rho)
6  D=[1 -1]; dD = 0; AD=convol(A,D); dAD=dA+1; zj=1;
       dzj=0;
7  Nu = N+1; G=zeros(Nu,Nu); H1=zeros(Nu,2*k+N-2+dB);
8  H2 = zeros(Nu,k+N+dA);
9  for j = 1:Nu,
10     zj = convol(zj,[0,1]); dzj = dzj + 1;
11     [Fj,dFj,Ej,dEj] = ...
12         xdync(zj,dzj,AD,dAD,C,dC);
13     [Nj,dNj,Mj,dMj] = ...
14         xdync(zj,dzj,C,dC,1,0);
15     [Gj,dGj] = polmul(Mj,dMj,Ej,dEj);
16     [Gj,dGj] = polmul(Gj,dGj,B,dB);
17     [Pj,dPj] = polmul(Mj,dMj,Fj,dFj);
18     [Pj,dPj] = poladd(Nj,dNj,Pj,dPj);
19     j,Fj,Ej,Mj,Nj,Gj,Pj
20     G(j,1:j) = flip(Gj(1:j));
21     H1(j,1:dGj-j+1) = Gj(j+1:dGj+1);
22     H2(j,1:dPj+1) = Pj;
23 end
24 K = inv(G'*G+rho*eye(Nu,Nu))*G'
25 // Note: inverse need not be calculated
26 KH1 = K * H1; KH2 = K * H2;
27 R1 = [1 KH1(1,:)]; dR1 = length(R1)-1;
28 Sc = KH2(1,:); dSc = length(Sc)-1;
29 Tc = K(1,:); dTc = length(Tc)-1;
30 endfunction;
```

Scilab code Exa 12.7 GPC design for viscosity control

```
1 // GPC design for viscosity control in Example 12.4
     on page 446.
2 // 12.7
3
4 exec('gpc_col.sci',-1);
5 exec('poladd.sci',-1);
6 exec('xdync.sci',-1);
7 exec('rowjoin.sci',-1);
8 exec('polsize.sci',-1);
9 exec('left_prm.sci',-1);
10 exec('t1calc.sci',-1);
11 exec('indep.sci',-1);
12 exec('seshft.sci',-1);
13 exec('makezero.sci',-1);
14 exec('move_sci.sci',-1);
15 exec('colsplit.sci',-1);
16 exec('clcoef.sci',-1);
17 exec('cindep.sci',-1);
18 exec('polmul.sci',-1);
19 exec('flip.sci',-1);
20
21 // GPC control of viscosity problem
22 A=[1 -0.44]; dA=1; B=[0.51 1.21]; dB=1; N=2; k=1;
23 C = [1 -0.44]; dC = 1; rho = 1;
24
25 [K,KH1,KH2,Tc,dTc,Sc,dSc,R1,dR1] = ...
26 gpc_col(A,dA,B,dB,C,dC,N,k,rho)
```

Scilab code Exa 12.8 GPC design

```
// GPC design for the problem discussed in Example
   12.3.
// 12.8

exec('gpc_Nc.sci',-1);
exec('xdync.sci',-1);
exec('rowjoin.sci',-1);
exec('polsize.sci',-1);
exec('left_prm.sci',-1);
exec('t1calc.sci',-1);
exec('indep.sci',-1);
exec('seshft.sci',-1);
exec('makezero.sci',-1);
exec('move_sci.sci',-1);
exec('colsplit.sci',-1);
exec('clcoef.sci',-1);
exec('cindep.sci',-1);
exec('polmul.sci',-1);
exec('poladd.sci',-1);
exec('flip.sci',-1);

A=[1 -0.44]; dA=1; B=[0.51 1.21]; dB=1;
C = [1 -0.44]; dC = 1;
k=1; N1 = 0; N2 = 2; Nu = 0; rho = 1;

[K,KH1,KH2,Tc,dTc,Sc,dSc,R1,dR1] = ...
gpc_Nc(A,dA,B,dB,C,dC,k,N1,N2,Nu,rho)
```

Scilab code Exa 12.9 Calculates the GPC law

```
// Calculates the GPC law for different prediction
   and control horizons
// 12.9

function [K,KH1,KH2,Tc,dTc,Sc,dSc,R1,dR1] = ...
```

```
gpc_Nc(A,dA,B,dB,C,dC,k,N1,N2,Nu,rho)
D=[1 -1]; dD=1; AD=convol(A,D); dAD=dA+1;
zj = 1; dzj = 0;
for i = 1:N1+k-1
    zj = convol(zj,[0,1]); dzj = dzj + 1;
end
M = 2*k+N2-2+dB;  P = max(k+N2+dA-1,dC-1)
G = zeros(N2-N1+1,Nu+1); H1 = zeros(N2-N1+1,M);
H2 = zeros(N2-N1+1,P+1);
for j = k+N1:k+N2
    zj = convol(zj,[0,1]); dzj = dzj + 1;
    [Fj,dFj,Ej,dEj] = xdync(zj,dzj,AD,dAD,C,dC);
    [Nj,dNj,Mj,dMj] = xdync(zj,dzj,C,dC,1,0);
    [Gj,dGj] = polmul(Mj,dMj,Ej,dEj);
    [Gj,dGj] = polmul(Gj,dGj,B,dB);
    [Pj,dPj] = polmul(Mj,dMj,Fj,dFj);
    [Pj,dPj] = poladd(Nj,dNj,Pj,dPj);
    if (j-k >= Nu)
    G(j-(k+N1-1),1:Nu+1) = flip(Gj(j-k-Nu+1:j-k+1));
else
    G(j-(k+N1-1),1:j-k+1) = flip(Gj(1:j-k+1));
end
    H1(j-(k+N1-1),1:j+k-2+dB) = Gj(j-k+2:2*j+dB-1);
    dPj = max(j-1+dA,dC-1);
    H2(j-(k+N1-1),1:dPj+1) = Pj;
end
K = inv(G'*G+rho*eye(Nu+1,Nu+1))*G';
// Note: inverse need not be calculated
KH1 = K * H1; KH2 = K * H2;
R1 = [1 KH1(1,:)]; dR1 = length(R1)-1;
Sc = KH2(1,:); dSc = length(Sc)-1;
Tc = K(1,:); dTc = length(Tc)-1;
endfunction;
```

Scilab code Exa 12.10 PID controller tuned with GPC

```
1  // PID controller, tuned with GPC, as discussed in
       Example 12.5 on page 452.
2  // 12.10
3
4  exec('gpc_pid.sci',-1);
5  exec('zpowk.sci',-1);
6  exec('xdync.sci',-1);
7  exec('rowjoin.sci',-1);
8  exec('polsize.sci',-1);
9  exec('left_prm.sci',-1);
10 exec('t1calc.sci',-1);
11 exec('indep.sci',-1);
12 exec('seshft.sci',-1);
13 exec('makezero.sci',-1);
14 exec('move_sci.sci',-1);
15 exec('colsplit.sci',-1);
16 exec('clcoef.sci',-1);
17 exec('cindep.sci',-1);
18
19 A = [1 -1.95 0.935];
20 B=-0.015;
21 C=1;
22 degA=2;
23 degB=0;
24 degC=0;
25 N1=1;
26 N2=5;
27 Nu=2;
28 gamm=0.05;
29 gamma_y=1;
30 lambda=0.02;
31
32 [Kp,Ki,Kd] = ...
33 gpc_pid(A,degA,B,degB,C,degC,N1,N2,Nu,lambda,gamm,
       gamma_y)
```

Scilab code Exa 12.11 Predictive PID tuned with GPC

```
1  // Predictive PID, tuned with GPC, as explained in
       Sec. 12.2.3.
2  // 12.11
3
4  function [Kp,Ki,Kd] = ...
5  gpc_pid(A,dA,B,dB,C,dC,N1,N2,Nu,lambda,gamm,gamma_y)
6  Adelta=convol(A,[1 -1]); G=[];
7  for i=N1:N2
8      zi=zpowk(i);
9      [E,dE,F,dF]=xdync(Adelta,dA+1,zi,i,C,dC);
10     [Gtilda,dGtilda,Gbar,dGbar] = ...
11       xdync(C,dC,zi,i,E*B,dE+dB);
12     for j = 1:i, Gtilda1(j)=Gtilda(i+1-j); end
13     Gtilda2 = Gtilda1.'; // Added because Scilab
         forms a column vecor
14     // while Matlab forms a row vector, by default
15         if i<=Nu-1
16        G=[G;[Gtilda2,zeros(1,Nu-i)]];
17     else
18        G=[G;Gtilda2(1:Nu)];
19     end
20 end
21 es=sum(C)/sum(A); gs=sum(B)/sum(A); F_s=es*A; G_s
     =[];
22 for i=1:Nu
23     if ((Nu - i) == 0)
24     row=gs*ones(1,i);
25     else
26     row=gs*ones(1,i); row=[row,zeros(Nu-i,Nu-i)];
27     end;
28     G_s=[G_s;row];
29 end
```

```
30 lambda_mat=lambda*(diag(ones(1,Nu)));
31 gamma_mat=gamm*(diag(ones(1,Nu)));
32 gamma_y_mat=gamma_y*(diag(ones(1,N2-N1+1)));
33 mat1=inv(G'*gamma_y_mat*G+lambda_mat+G_s'*gamma_mat*
      G_s);
34 mat2=mat1*(G'*gamma_y_mat);
35 mat2_s=mat1*(G_s'*gamma_mat);
36 h_s=sum(mat2_s(1,:)); h=mat2(1,:);
37 T=C; R=C*(sum(h(:))+h_s); S=0;
38 for i=N1:N2
39     zi=zpowk(i);
40     [E,dE,F,dF]=xdync(Adelta,dA+1,zi,i,C,dC);
41     [Gtilda,dGtilda,Gbar,dGbar]=...
42       xdync(C,dC,zi,i,E*B,dE+dB);
43     S=S+F*h(i);
44 end
45 S=S+F_s*h_s;
46 if length(A)==3
47    Kp=S(1)-R-S(3); Ki=R; Kd=S(3);
48 else
49    Kp=S(1)-R; Ki=R; Kd=0;
50 end
51
52 endfunction;
```

Chapter 13

Linear Quadratic Gaussian Control

Scilab code Exa 13.1 Spectral factorization

```
1 // Spectral factorization , as discussed in Example
     13.3 on page 467.
2 // 13.1
3
4 exec('spec1.sci',-1);
5 exec('flip.sci',-1);
6 exec('polmul.sci',-1);
7 exec('polsize.sci',-1);
8 exec('poladd.sci',-1);
9
10 A = convol([-0.5 1],[-0.9 1]); dA = 2;
11 B = 0.5*[-0.9 1]; dB = 1; rho = 1;
12 [r,beta1,sigma] = spec1(A,dA,B,dB,rho)
```

Scilab code Exa 13.2 Function to implement spectral factorization

```
// Function to implement spectral factorization, as
   discussed in sec. 13.1.
// 13.2

function [r,b,rbbr] = spec1(A,dA,B,dB,rho)
AA = rho * convol(A,flip(A));
BB = convol(B,flip(B));
diff1 = dA - dB;
dBB = 2*dB;
for i = 1:diff1
    [BB,dBB] = polmul(BB,dBB,[0 1],1);
end
[rbbr,drbbr] = poladd(AA,2*dA,BB,dBB);
rts = roots(rbbr);  // roots in descending order of
   magnitude
rts = flip(rts);
rtsin = rts(dA+1:2*dA);
b = 1;
for i = 1:dA,
    b = convol(b,[1 -rtsin(i)]);
end
br = flip(b);
bbr = convol(b,br);
r = rbbr(1) / bbr(1);
endfunction;
```

Scilab code Exa 13.3 Spectral factorization

```
// Spectral factorization, to solve Eq. 13.47 on
   page 471.
// 13.3

// function [r,b,dAFW] = ...
//     specfac(A,degA,B,degB,rho,V,degV,W,degW,F,degF
   )
```

```
6  // Implements the spectral factorization for use
       with LQG control
7  // design method of Ahlen and Sternard
8
9  function [r,b,dAFW] = ...
10     specfac(A,degA,B,degB,rho,V,degV,W,degW,F,degF)
11 AFW = convol(A,convol(W,F));
12 dAFW = degA + degF + degW;
13 AFWWFA = rho * convol(AFW,flip(AFW));
14 BV = convol(B,V);
15 dBV = degB + degV;
16 BVVB = convol(BV,flip(BV));
17 diff1 = dAFW - dBV;
18 dBVVB = 2*dBV;
19 for i = 1:diff1
20     [BVVB,dBVVB] = polmul(BVVB,dBVVB,[0 1],1);
21 end
22 [rbb,drbb] = poladd(AFWWFA,2*dAFW,BVVB,dBVVB);
23 Rbb = polyno(rbb,'z');
24 rts = roots(Rbb);
25 rtsin = rts(dAFW+1:2*dAFW);
26 b = 1;
27 for i = 1:dAFW,
28     b = convol(b,[1 -rtsin(i)]);
29 end
30 b = real(b);
31 br = flip(b);
32 bbr = convol(b,br);
33 r = rbb(1) / bbr(1);
34 endfunction;
```

Scilab code Exa 13.4 LQG control design by polynomial method

```
1  // LQG control design by polynomial method, to solve
       Eq. 13.51 on page 472.
```

```
2  // 13.4
3
4  // LQG controller design by method of Ahlen and
       Sternad
5  // function [R,degR,S,degS] = ...
6  // lqg(A,degA,B,degB,C,degC,k,rho,V,degV,W,degW,F,
       degF)
7
8  function [R,degR,S,degS] = ...
9  lqg1(A,degA,B,degB,C,degC,k,rho,V,degV,W,degW,F,degF
       )
10
11 [r,b,degb] = ...
12 specfac(A,degA,B,degB,rho,V,degV,W,degW,F,degF);
13
14 WFA = flip(convol(A,convol(F,W)));
15 dWFA = degW + degF + degA;
16
17 [rhs1,drhs1] = polmul(W,degW,WFA,dWFA);
18 [rhs1,drhs1] = polmul(rhs1,drhs1,C,degC);
19 rhs1 = rho * rhs1;
20 rhs2 = convol(C,convol(V,flip(convol(B,V))));
21 drhs2 = degC + 2*degV + degB;
22 for i = 1:degb-degB-degV,
23   rhs2 = convol(rhs2,[0,1]);
24 end
25 drhs2 = drhs2 + degb-degB-degV;
26 C1 = zeros(1,2);
27
28 [C1,degC1] = putin(C1,0,rhs1,drhs1,1,1);
29 [C1,degC1] = putin(C1,degC1,rhs2,drhs2,1,2);
30 rbf = r * flip(b);
31 D1 = zeros(2,2);
32 [D1,degD1] = putin(D1,0,rbf,degb,1,1);
33 for i = 1:k,
34     rbf = convol(rbf,[0 1]);
35 end
36 [D1,degD1] = putin(D1,degD1,rbf,degb+k,2,2);
```

```
37 N = zeros(1,2);
38 [N,degN] = putin(N,0,-B,degB,1,1);
39 [AF,dAF] = polmul(A,degA,F,degF);
40 [N,degN] = putin(N,degN,AF,dAF,1,2);
41
42 [Y,degY,X,degX] = xdync(N,degN,D1,degD1,C1,degC1);
43
44 [R,degR] = ext(X,degX,1,1);
45 [S,degS] = ext(X,degX,1,2);
46 X = flip(Y);
47
48 endfunction;
```

Scilab code Exa 13.5 LQG design

```
1 // LQG design for the problem discussed in Example
     13.4 on page 472.
2 // 13.5
3
4 // MacGregor's first control problem
5
6 exec('lqg1.sci',-1);
7 exec('cl.sci',-1);
8 exec('specfac.sci',-1);
9 exec('flip.sci',-1);
10 exec('polmul.sci',-1);
11 exec('polsize.sci',-1);
12 exec('poladd.sci',-1);
13 exec('polyno.sci',-1);
14 exec('putin.sci',-1);
15 exec('xdync.sci',-1);
16 exec('rowjoin.sci',-1);
17 exec('left_prm.sci',-1);
18 exec('t1calc.sci',-1);
19 exec('indep.sci',-1);
```

```
exec('seshft.sci',-1);
exec('makezero.sci',-1);
exec('move_sci.sci',-1);
exec('colsplit.sci',-1);
exec('clcoef.sci',-1);
exec('cindep.sci',-1);
exec('ext.sci',-1);
exec('zpowk.sci',-1);
exec('tfvar.sci',-1);
exec('l2r.sci',-1);
exec('transp.sci',-1);
exec('tf.sci',-1);
exec('covar_m.sci',-1);

A = [1 -1.4 0.45]; dA = 2; C = [1 -0.5]; dC = 1;
B = 0.5*[1 -0.9]; dB = 1; k = 1; int1 = 0; F = 1; dF
    = 0;
V = 1; W = 1; dV = 0; dW = 0;
rho = 1;
[R1,dR1,Sc,dSc] = lqg1(A,dA,B,dB,C,dC,k,rho,V,dV,W,
   dW,F,dF)
[Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] = ...
       cl(A,dA,B,dB,C,dC,k,Sc,dSc,R1,dR1,int1);
```

Scilab code Exa 13.6 LQG control design for viscosity control problem

```
// LQG control design for viscosity control problem
   discussed in Example 13.5.
// 13.6


exec('lqg1.sci',-1);
exec('cl.sci',-1);
exec('specfac.sci',-1);
exec('flip.sci',-1);
```

```
exec('polmul.sci',-1);
exec('polsize.sci',-1);
exec('poladd.sci',-1);
exec('polyno.sci',-1);
exec('putin.sci',-1);
exec('xdync.sci',-1);
exec('rowjoin.sci',-1);
exec('left_prm.sci',-1);
exec('t1calc.sci',-1);
exec('indep.sci',-1);
exec('seshft.sci',-1);
exec('makezero.sci',-1);
exec('move_sci.sci',-1);
exec('colsplit.sci',-1);
exec('clcoef.sci',-1);
exec('cindep.sci',-1);
exec('ext.sci',-1);
exec('zpowk.sci',-1);
exec('tfvar.sci',-1);
exec('l2r.sci',-1);
exec('transp.sci',-1);
exec('tf.sci',-1);
exec('covar_m.sci',-1);

// Viscosity control problem of MacGregor
A = [1 -0.44]; dA = 1; B = [0.51 1.21]; dB = 1;
C = [1 -0.44]; dC = 1; k = 1; int1 = 1; F = [1 -1];
   dF = 1;
V = 1; W = 1; dV = 0; dW = 0;
rho = 1;
[R1,dR1,Sc,dSc] = lqg1(A,dA,B,dB,C,dC,k,rho,V,dV,W,
   dW,F,dF);
[Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] = ...
    cl(A,dA,B,dB,C,dC,k,Sc,dSc,R1,dR1,int1);
```

Scilab code Exa 13.7 Simplified LQG control design

```
1 // Simplified LQG control design, obtained by the
     solution of Eq. 13.53 on page 476.
2 // 13.7
3
4 // LQG controller simple design by method of Ahlen
     and Sternad
5 // function [R1,dR1,S,dS] = ...
6 // lqg_simple(A,dA,B,dB,C,dC,k,rho,V,dV,W,dW,F,dF)
7
8 function [R1,dR1,S,dS] = ...
9 lqg_simple(A,dA,B,dB,C,dC,k,rho,V,dV,W,dW,F,dF)
10 [r,b,db] = specfac(A,dA,B,dB,rho,V,dV,W,dW,F,dF);
11 [D,dD] = polmul(A,dA,F,dF);
12 [zk,dzk] = zpowk(k);
13 [N,dN] = polmul(zk,dzk,B,dB);
14 [RHS,dRHS] = polmul(C,dC,b,db);
15 [S,dS,R1,dR1] = xdync(N,dN,D,dD,RHS,dRHS);
16 endfunction;
```

Scilab code Exa 13.8 LQG control design

```
1 // LQG control design for the problem discussed in
     Example 13.6 on page 474.
2 // 13.8
3
4 // Solves Example 3.1 of Ahlen and Sternad in Hunt's
     book
5 exec('lqg1.sci',-1);
6 exec('specfac.sci',-1);
7 exec('flip.sci',-1);
8 exec('polmul.sci',-1);
9 exec('polsize.sci',-1);
10 exec('poladd.sci',-1);
```

```
11 exec('polyno.sci',-1);
12 exec('putin.sci',-1);
13 exec('clcoef.sci',-1);
14 exec('xdync.sci',-1);
15 exec('rowjoin.sci',-1);
16 exec('left_prm.sci',-1);
17 exec('t1calc.sci',-1);
18 exec('indep.sci',-1);
19 exec('seshft.sci',-1);
20 exec('makezero.sci',-1);
21 exec('move_sci.sci',-1);
22 exec('colsplit.sci',-1);
23 exec('cindep.sci',-1);
24 exec('ext.sci',-1);
25
26 A = [1 -0.9]; dA = 1; B = [0.1 0.08]; dB = 1;
27 k = 2; rho = 0.1; C = 1; dC = 0;
28 V = 1; dV = 0; F = 1; dF = 0; W = [1 -1]; dW = 1;
29 [R1,dR1,Sc,dSc] = ...
30 lqg1(A,dA,B,dB,C,dC,k,rho,V,dV,W,dW,F,dF)
```

Scilab code Exa 13.9 Performance curve for LQG control design of viscosity problem

```
1 // Performance curve for LQG control design of
     viscosity problem
2 // 13.9
3
4 exec('lqg1.sci',-1);
5 exec('specfac.sci',-1);
6 exec('flip.sci',-1);
7 exec('polmul.sci',-1);
8 exec('polsize.sci',-1);
9 exec('poladd.sci',-1);
10 exec('polyno.sci',-1);
```

```
exec('putin.sci',-1);
exec('clcoef.sci',-1);
exec('xdync.sci',-1);
exec('rowjoin.sci',-1);
exec('left_prm.sci',-1);
exec('t1calc.sci',-1);
exec('indep.sci',-1);
exec('seshft.sci',-1);
exec('makezero.sci',-1);
exec('move_sci.sci',-1);
exec('colsplit.sci',-1);
exec('cindep.sci',-1);
exec('ext.sci',-1);
exec('cl.sci',-1);
exec('zpowk.sci',-1);
exec('tfvar.sci',-1);
exec('l2r.sci',-1);
exec('transp.sci',-1);
exec('tf.sci',-1);
exec('covar_m.sci',-1);

// MacGregor's Viscosity control problem
A = [1 -0.44]; dA = 1; B = [0.51 1.21]; dB = 1;
C = [1 -0.44]; dC = 1; k = 1; int1 = 1; F = [1 -1];
   dF = 1;
V = 1; W = 1; dV = 0; dW = 0;
u_lqg = []; y_lqg =[]; uy_lqg = [];

for rho = 0.001:0.1:3,
    [R1,dR1,Sc,dSc] = lqg1(A,dA,B,dB,C,dC,k,rho,V,dV
       ,W,dW,F,dF);
    [Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] = ...
        cl(A,dA,B,dB,C,dC,k,Sc,dSc,R1,dR1,int1);
    u_lqg = [u_lqg uvar]; y_lqg = [y_lqg yvar];
    uy_lqg = [uy_lqg; [rho uvar yvar]];
end
plot(u_lqg,y_lqg,'g')
save('lqg_visc.dat','uy_lqg');
```

Scilab code Exa 13.10 Performance curve for GMVC design of first control problem by MacGregor

```
1  // Performance curve for GMVC design of MacGregor's
       first control problem
2  // 13.10
3
4  exec('gmv.sci',-1);
5  exec('xdync.sci',-1);
6  exec('rowjoin.sci',-1);
7  exec('polsize.sci',-1);
8  exec('left_prm.sci',-1);
9  exec('t1calc.sci',-1);
10 exec('indep.sci',-1);
11 exec('seshft.sci',-1);
12 exec('makezero.sci',-1);
13 exec('move_sci.sci',-1);
14 exec('colsplit.sci',-1);
15 exec('clcoef.sci',-1);
16 exec('cindep.sci',-1);
17 exec('polmul.sci',-1);
18 exec('poladd.sci',-1);
19 exec('cl.sci',-1);
20 exec('zpowk.sci',-1);
21 exec('tfvar.sci',-1);
22 exec('l2r.sci',-1);
23 exec('transp.sci',-1);
24 exec('tf.sci',-1);
25 exec('covar_m.sci',-1);
26
27 // MacGregor's first control problem
28 A = [1 -1.4 0.45]; dA = 2; C = [1 -0.5]; dC = 1;
29 B = 0.5*[1 -0.9]; dB = 1; k = 1; int1 = 0;
30 u_gmv = []; y_gmv = []; uy_gmv = [];
```

```
31
32 for rho = 0:0.1:10,
33     [S,dS,R,dR] = gmv(A,dA,B,dB,C,dC,k,rho,int1);
34     [Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] = ...
35         cl(A,dA,B,dB,C,dC,k,S,dS,R,dR,int1);
36     u_gmv = [u_gmv uvar]; y_gmv = [y_gmv yvar];
37     uy_gmv = [uy_gmv; [rho uvar yvar]];
38 end
39 plot(u_gmv,y_gmv,'b')
40 save('gmv_mac1.dat','uy_gmv');
```

Chapter 14

State Space Techniques in Controller Design

Scilab code Exa 14.1 Pole placement controller for inverted pendulum

```
1 // Pole placement controller for inverted pendulum,
     discussed in Example 14.1 on page 490. 2.1 should
      be executed before starting this code
2 // 14.1
3
4 exec('pol2cart.sci',-1);
5
6 C = eye(4,4);
7 D = zeros(4,1);
8 Ts = 0.01;
9 G = syslin('c',A,B,C,D);
10 H = dscr(G,Ts);
11 [a,b,c,d] = H(2:5);
12 rise = 5; epsilon = 0.1;
13 N = rise/Ts;
14 omega = %pi/2/N;
15 r = epsilon^(omega/%pi);
16 r1 = r; r2 = 0.9*r;
17 [x1,y1] = pol2cart(omega,r1);
```

```
18 [x2,y2] = pol2cart(omega,r2);
19 p1 = x1+%i*y1;
20 p2 = x1-%i*y1;
21 p3 = x2+%i*y2;
22 p4 = x2-%i*y2;
23 P = [p1;p2;p3;p4];
24 K = ppol(a,b,P)
```

Scilab code Exa 14.2 Compensator calculation

```
1  // Compensator calculation for Example 14.6 on page
   507.
2  // 14.2
3
4  exec('polyno.sci',-1);
5  exec('polmul.sci',-1);
6  exec('polsize.sci',-1);
7
8  A = [1 2; 0 3]; c = [1 0];
9  p = roots(polyno([1 -0.5 0.5],'z'));
10 b = [0; 1];
11 K = ppol(A,b,p);
12
13 p1=0.1+0.1*%i; p2=0.1-0.1*%i;
14 phi = real(convol([1 -p1],[1 -p2]));
15 Obs = [c; c*A];
16 alphae = A^2-0.2*A+0.02*eye(2,2);
17 Lp = alphae*inv(Obs)*[0; 1];
18 Lp = ppol([1 0;2 3], ...
19 [1; 0],[0.1+0.1*%i 0.1-0.1*%i]);
20 Lp = Lp';
21
22 C = [1 0 0.5 2;0 1 -4.71 2.8];
23 dC = 1;
24
```

```
25 [HD,dHD] = polmul(K,0,C,dC);
26 [HD,dHD] = polmul(HD,dHD,Lp,0);
```

Scilab code Exa 14.3 Kalman filter example of estimating a constant

```
1 // Kalman filter example of estimating a constant,
     discussed in Example 14.7.
2 // 14.3
3
4 exec('kal_ex.sci',-1);
5
6 x = 5; xhat = 2; P = 1; xvec = x;
7 xhat_vec = xhat; Pvec = P; yvec = x;
8 for i = 1:200,
9     xline = xhat; M = P;
10    [xhat,P,y] = kal_ex(x,xline,M);
11    xvec = [xvec;x];
12    xhat_vec = [xhat_vec;xhat];
13    Pvec = [Pvec;P]; yvec = [yvec;y];
14 end
15 n = 1:201;
16 plot(Pvec);
17 xtitle('','n');
18 halt();
19 clf();
20 plot(n,xhat_vec',n,yvec',n,xvec');
21 xtitle('','n');
```

Scilab code Exa 14.4 Kalman filter example of estimating a constant

```
1 // Kalman filter example of estimating a constant
2 // 14.4
3
```

```
4  function [xhat,P,y] = kal_ex(x,xline,M)
5  y = x + rand();
6  Q = 0; R = 1;
7  xhat_ = xline;
8  P_ = M + Q;
9  K = P_/(P_+R);
10 P = (1-K)*P_;
11 xhat = xhat_ + K*(y-xhat_);
12 endfunction;
```