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

September 12, 2014

[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

```
// Model of inverted pendulum
// 2.1

Km = 0.00767;
Kg = 3.7;
Rm = 2.6;
r = 0.00635;
M = 0.522;
m = 0.231;
g = 9.81;
L = 0.305;
J = 0;

D1 = (J+m*L^2)*(M+m)-m^2*L^2;
alpha = m*g*L*(M+m)/D1;
beta1 = m*L/D1;
gamma1 = m^2*g*L^2/D1;
delta = (J+m*L^2)/D1;

```

```
alpha1 = Km*Kg/Rm/r;
alpha2 = Km^2*Kg^2/Rm/r^2;

A = zeros(4,4);
A(1,3) = 1;
A(2,4) = 1;
A(3,2) = -gamma1;
A(3,3) = -alpha2*delta;
A(4,2) = alpha;
A(4,3) = alpha2*beta1;

B = zeros(4,1);
B(3) = alpha1*delta;
B(4) = -alpha1*beta1;
```

Scilab code Exa 2.2 Exponential of the matrix

```
// Exponential of the matrix
// 2.2

F = [-1 0;1 0];
expm(F)
```

Scilab code Exa 2.3 ZOH equivalent state space system

```
// ZOH equivalent state space system
// 2.3

F = [-1 0;1 0]; G = [1; 0];
C = [0 1]; D = 0; Ts=1;
sys = syslin('c',F,G,C,D);
sysd = dscr(sys,Ts)
```

Chapter 3

Linear Systems

Scilab code Exa 3.1 Energy of a signal

```
// Energy of a signal
// 3.1

u = [4 5 6];
Eu = norm(u)^2;
ruu = xcorr(u);
Lu = length(ruu);
Eu = ruu(ceil(Lu/2));
```

Scilab code Exa 3.2 Convolution of two sequences

```
// Convolution of two sequences
// 3.2

h = [1 2 3];
u = [4 5 6];
y = convol(u,h)
```

Chapter 4

Z Transform

Scilab code Exa 4.1 To produce a sequence

```
// To produce a^n 1(n)
// 4.1

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)>=0,
       y(i) = a^n(i);
    end
end
stem(n,y)
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

```
// Discrete transfer function of the continuous
   state space system
// 4.4

F = [0 0; 1 -0.1]; G = [0.1; 0];
C = [0 1]; dt = 0.2;
sys = syslin('c',F,G,C);
sysd = dscr(sys,dt);
H = ss2tf(sysd);
```

Scilab code Exa 4.5 Computation of residues

```
// Computation of residues
// 4.5
// Numerator and denominator coefficients
// are passed in decreasing powers of z(say)

function [res,pol,q] = respol(num,den)
len = length(num);
if num(len) == 0
    num = num(1:len-1);
end

[resi,q] = pfe(num,den);
res = resi(:,2);
res = int(res) + (clean(res - int(res),1.d-04));
pol = resi(:,1);
pol = int(pol) + (clean(pol - int(pol),1.d-04));
endfunction;
```

```
//////////////////////////////////////////////////////
// Partial fraction expansion

function [resid1,q] = pfe(num,den)
x = poly(0,'x');
s = %s;

num2 = flip(num);
den2 = flip(den);
num = poly(num2,'s','coeff');
den = poly(den2,'s','coeff');
[fac,g] = factors(den);
polf = polfact(den);
n = 1;

r = clean(real(roots(den)),1.d-5);
//disp(r);
l = length(r);
r = gsort(r,'g','i');
r = [r; 0];
j = 1;
t1 = 1; q = [];
rr = 0;
rr1 = [0 0];
m = 1;

  for i = j:l
     if abs(r(i)- r(i+1)) < 0.01 then
       r(i);
       r(i+1);
       n = n+1;
       m = n;
       //pause
       t1 = i;
       //disp('Repeated roots')
     else
       m = n;
```

```
       //pause
       n = 1;
    end
    i;
    if n == 1 then
      rr1 = [rr1; r(i) m];
      //pause
    end;
       j = t1 + 1;
  end;
rr2 = rr1(2:$,:);
[r1,c1] = size(rr2);
den1 = 1;

for i = 1:r1
  den1 = den1 * ((s-rr2(i,1))^(rr2(i,2)));
end;
[rem,quo] = pdiv(num,den);
q = quo;
if quo ~= 0
  num = rem;
end

tf = num/den1;
res1 = 0;
res3 = [s 0];
res5 = [0 0];
for i = 1:r1
  j = rr2(i,2) + 1;
  tf1 = tf; //strictly proper
  k = rr2(i,2);
  tf2 = ((s-rr2(i,1))^k)*tf1;
  rr2(i,1);
  res1 = horner(tf2,rr2(i,1));
  res2 = [(s - rr2(i,1))^(rr2(i,2)) res1];
  res4 = [rr2(i,1) res1];
  res3 = [res3; res2];
  res5 = [res5; res4];
```

```
  res = res1;
    for m = 2:j-1
    k;
    rr2(i,1);
      tf1 = derivat(tf2)/factorial(m-1); //ith derivative
      res = horner(tf1,rr2(i,1));
      res2 = [(s - rr2(i,1))^(j-m) res];
      res4 = [rr2(i,1) res];
      res5 = [res5; res4];
      res3 = [res3; res2];
      tf2 = tf1;
    end;
end;
resid = res3(2:$,:); //with s terms
resid1 = res5(2:$,:); //only poles(in decreasing no. of repetitions)
endfunction;
// ////////////////////////////////////////////////////////////
```

Scilab code Exa 4.6 Partial fraction expansion

```
// Partial fraction expansion for Example 4.24
// 4.6

//               2z^2 + 2z
// G(z)   =   ---------------
//             z^2 + 2z - 3

exec('respol.sci',-1);
exec('flip.sci',-1);

num = [2 2 0];
```

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

Scilab code Exa 4.7 Partial fraction expansion

```
// Partial fraction expansion for Example 4.26
// 4.7

//              z^2 + z            A             B              C
// G(z) = ———————————— = ——————— + ————————— + —————————
//              (z - 1)^3        (z - 1)    (z - 1)^2    (z - 1)^3

exec('respol.sci',-1);
exec('flip.sci',-1);

num = [1 1 0];
den = convol([1 -1],convol([1 -1],[1 -1])); // poly multiplication
[res,pol] = respol(num,den)

// Output interpretation
// res =
//C = 2
//B = 1
//A = 0

// pol =
// 1      (z - 1)^3
// 1      (z - 1)^2
// 1      (z - 1)
```

Scilab code Exa 4.8 Partial fraction expansion

```
// Partial fraction expansion for Example 4.27
// 4.8

//           11z^2 - 15z + 6         A1            A2           B
// G(z) = ----------------- = ------- + --------- + --------
//          (z - 2) (z - 1)^2    (z - 1)    (z - 1)^2    (z - 2)

exec('respol.sci',-1);
exec('flip.sci',-1);

num = [11 -15 6];
den = convol([1 -2],convol([1 -1],[1 -1]));
[res,pol] = respol(num,den) //User defined function
```

Scilab code Exa 4.9 Partial fraction expansion

```
// Partial fraction expansion for Example 4.29
// 4.9

//                    z^2 + 2z
// G(z)  =  -------------------
//             (z + 1)^2 (z - 2)

exec('respol.sci',-1);
exec('flip.sci',-1);

num = [1 2 0];
```

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

Scilab code Exa 4.10 Partial fraction expansion

```
// Partial fraction expansion for Example 4.30
// 4.10

//                    3 - (5/6)z^-1
                             3z^2 - (5/6)z
// G(z) = ------------------------------ =
   ------------------------------
//               (1-(1/4)z^-1)(1-(1/3)z^-1)          (z -
   (1/4))(z - (1/3))

// No equivalent of residuez

exec('respol.sci',-1);
exec('flip.sci',-1);

num = [3 -5/6 0];
den = convol([1 -1/4],[1 -1/3]);
[res,pol,q] = respol(num,den)
```

Scilab code Exa 4.11 Long division of problems

```
// Long division of problems discussed in Example
   4.32 on page 102
// 4.11

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

```

```
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

```
// Sinusoidal plots for increasing frequency
// 5.1

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

n=0:16;
subplot(2,2,1), stem(n,cos(n*%pi/8))
xgrid,xtitle('','n','cos(n*pi/8)')
subplot(2,2,2), stem(n,cos(n*%pi/4))
xgrid,xtitle('','n','cos(n*pi/4)')
subplot(2,2,3), stem(n,cos(n*%pi/2))
xgrid,xtitle('','n','cos(n*pi/2)')
subplot(2,2,4), stem(n,cos(n*%pi))
xgrid,xtitle('','n','cos(n*pi)')
```

Scilab code Exa 5.2 Bode plots

```
// 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]);
```

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

Scilab code Exa 5.4 Bode plot of the differencing filter

```
// Bode plot of the differencing filter, discussed
    in Example 5.6 on page 130
// 5.4

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

w = 0.01:0.01:%pi;
G = 1-exp(-%i*w);
subplot(2,1,1)
plot2d1("gll",w,abs(G),style = 2);
label('',4,' ','Magnitude',4);
subplot(2,1,2)
plot2d1("gln",w,phasemag(G),style = 2);
label('',4,'w','Phase',4)
```

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

```
// Bode plot of minimum and nonminimum phase filters
    , discussed in Example 5.9 on page 145
// 5.5

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

omega = linspace(0,%pi);
ejw = exp(-%i*omega);
G1 = 1.5*(1-0.4*ejw);
mag1 = abs(G1); angle1 = phasemag(G1);
G2 = -0.6*(1-2.5*ejw);
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);
```

Chapter 6

Identification

Scilab code Exa 6.1 Least squares solution

```
// Least squares solution of the simple problem
   discussed in Example 6.4 on page 164
// 6.1

Mag = 10; V = 10; No_pts = 100; theta = 2;
Phi = Mag * (1-2*rand(No_pts,1));
E = V * (1-2*rand(No_pts,1));
Z = Phi*theta + E;
LS = Phi \ Z
Max = max(Z ./ Phi), Min = min(Z ./ Phi)
```

Scilab code Exa 6.2 ACF calculation

```
// ACF calculation for the problem discussed in
   Example 6.5 on page 167
// 6.2

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);
```

```
v1 = arsimul(m1,xi);
M1 = armax1(0,0,2,v1,zeros(1,10000))
disp(M1)

// Simulation and estimation of second model
m2 = armac(1,0,[1,-0.9,0.2],1,1,1);
v2 = arsimul(m2,xi);
M2 = armax1(0,0,2,v2,zeros(1,10000))
disp(M2)

// ACF and PACF of both models
plotacf(v1,1,11,1);
xset('window',1), plotacf(v2,1,11,1);
xset('window',2), pacf(v1,11);
xset('window',3), pacf(v2,11);
```

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

```
// Estimation with a larger order model results in
   large uncertainty, as discussed in Example 6.15
   on page 185.
// 6.8

m = armac(1,0,[1 -0.9 0.2],1,1,1);
xi = 0.1*rand(1,10000);
v = arsimul(m,xi);
M1 = armax1(0,0,2,v,zeros(1,10000))
disp(M1)
M2 = armax1(0,0,3,v,zeros(1,10000))
disp(M2)
```

Scilab code Exa 6.9 Determination of order of AR process

```
// Determination of order of AR(p) process , as discussed in Example 6.18 on page 189.
// 6.9

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

// Define model and generate data
m = armac([1,-1,0.5],0,1,1,1,1);
xi = 0.1*rand(1,10000,'normal');
v = arsimul(m,xi);

// Plot noise , plant output and PACF
subplot(2,1,1), plot(v(1:500));
label('',6,'','v',6);
subplot(2,1,2), plot(xi(1:500));
label('',6,'n','xi',6);
xset('window',1)
pacf(v,10);
```

Scilab code Exa 6.10 Determination of the PACF of AR process

```
// Determination of the PACF of AR(p) process , as explained in Sec. 6.4.5.
// 6.10

function [ajj] = pacf(v,M)
exec('label.sci',-1);
rvvn = xcorr(v,'coeff');
len = length(rvvn);
zero = (len+1)/2;
rvvn0 = rvvn(zero);
rvvn_one_side = rvvn(zero+1:len);
ajj = [];
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
```

```
  end
  b(i) = -rvv_rest(k+i-1);
end
a = A\b;
ajj = a(p);
endfunction;
```

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

```
// PACF plot of an MA process decays slowly, as
   discussed in Example 6.19 on page 190.
// 6.12

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

m = armac(1,0,[1,-0.9,0.2],1,1,1);
xi = 0.1*rand(1,10000);
v = arsimul(m,xi);
plotacf(v,1,11,1);
xset('window',1);
pacf(v,11);
```

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

```
// Implementation of trial and error procedure to
   determine ARMA(1,1) process, presented in Example
    6.20 on page 191.
// 6.13

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

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

// Set up the model for simulation
arma_mod = armac([1 -0.8],0,[1 -0.3],1,1,1);

// Generate the inputs for simulation
// Deterministic Input can be anything
u = zeros(1,2048);
e = rand(1,2048,'normal');

// Simulate the model
v = arsimul(arma_mod,[u e]);

// Plot ACF and PACF for 10 lags
plotacf(v,1e-03,11,1);
xset('window',1), pacf(v,10);

// Estimate AR(1) model and present it
// compute the residuals
[mod_est1,err_mod1] = armax1(1,0,0,v,zeros(1,length(
    v)));
disp(mod_est1);

// Plot ACF and PACF for 10 lags
xset('window',2), plotacf(err_mod1,1e-03,11,1);
xset('window',3), pacf(err_mod1,10);

// Check ACF and PACF of residuals
[mod_est2,err_mod2] = armax1(1,0,1,v,zeros(1,length(
    v)));
disp(mod_est2);

// Plot ACF and PACF for 10 lags
xset('window',4), plotacf(err_mod2,1e-03,11,1);
xset('window',5), pacf(err_mod2,10);
```

Scilab code Exa 6.14 Determination of FIR parameters

```
// Determination of FIR parameters as described in
   Example 6.22 on page 200.
// 6.14

exec('cra.sci',-1);
exec('filt.sci',-1);
exec('covf.sci',-1);

sig = 0.05;
process_mod = armac([1 -0.5],[0 0.6 -0.2],1,1,1,sig)
   ;

u = prbs_a(2225,40);
xi = rand(1,2225,'normal');
y = arsimul(process_mod,[u xi]);
u = [u zeros(1,length(y)-length(u))];
z = [y' u'];

// Plot y as a function of u and xi
exec('label.sci',-1)
subplot(3,1,1), plot(y(1:500)),
label('',4,'','y',4)
subplot(3,1,2), plot(u(1:500))
label('',4,'','u',4)
subplot(3,1,3), plot(sig*xi(1:500))
label('',4,'n','xi',4)

xset('window',1);
[ir,r,cl_s] = cra(detrend(z,'constant'));
ir_act = filt([0 0.6 -0.2],[1 -0.5],...
              [1 zeros(1,9)]);
replot([0,min(min(ir),min(ir_act)) - 0.1,9,max(max(
```

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

Scilab code Exa 6.15 Determination of ARX parameters

```
// Determination of ARX parameters as described in
    Example 6.25 on page 203.
// 6.15

exec('armac1.sci',-1);
exec('cra.sci',-1);
exec('arx.sci',-1);
exec('filt.sci',-1);
exec('covf.sci',-1);
exec('stem.sci',-1);

process_arx = armac1([1 -0.5],[0 0 0.6
    -0.2],1,1,1,0.05);
u = prbs_a(5000,250);
xi = rand(1,5000,'normal');
y = arsimul(process_arx,[u xi]);
z = [y(1:length(u))' u'];
zd = detrend(z,'constant');

// Compute IR for time-delay estimation
[ir,r,cl_s] = cra(detrend(z,'constant'));

// Time-delay = 2 samples
// Estimate ARX model (assume known orders)
na = 1; nb = 2; nk = 2;
[theta_arx,cov_arx,nvar,resid] = arx(zd,na,nb,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');
```

Scilab code Exa 6.16 Determination of ARMAX parameters

```
// Determination of ARMAX parameters as described in
    Example 6.27 on page 206.
// 6.16

exec('cra.sci',-1);
exec('stem.sci',-1);
exec('filt.sci',-1);
exec('covf.sci',-1);

process_armax = armac([1 -0.5],[0 0 0.6 -0.2],[1
    -0.3],1,1,0.05);
u = prbs_a(5000,250);
xi = rand(1,5000);
y = arsimul(process_armax,[u xi]);
z = [y(1:length(u))' u'];
zd = detrend(z,'constant');

//Compute IR for time-delay estimation
[ir,r,cl_s] = cra(detrend(z,'constant'));

```

```
//Estimate ARMAX model (assume known orders)
na = 1; nb = 3; nc = 1; nk = 2;
[theta_armax,resid] = armax1(na,nb,nc,zd(:,1)',zd
    (:,2)',1);
disp(theta_armax)

// 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');
```

Scilab code Exa 6.17 Determination of OE parameters

```
// Determination of OE parameters as described in
    Example 6.28 on page 209.
// 6.17

exec('armac1.sci',-1);
exec('oe.sci',-1);
exec('cra.sci',-1);
exec('stem.sci',-1);
exec('filt.sci',-1);
exec('covf.sci',-1);
exec('deconvol.sci',-1);

b = [0 0 0.6 -0.2];
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

```
// Procedure to draw root locus for the problem
   discussed in Example 7.1 on page 247.
// 7.1

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

H = tf(1,[1 -1 0],-1);
evans(H)
```

Scilab code Exa 7.2 Procedure to draw the Nyquist plot

```
// Procedure to draw the Nyquist plot, as discussed
   in Example 7.2 on page 250.
// 7.2

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

H = tf(1,[1 -1 0],-1);
nyquist(H,0.1,0.5)
```

Scilab code Exa 7.3 Procedure to draw Bode plots

```
// Procedure to draw Bode plots in Fig. 7.11 on page
    255.
// 7.3

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

pol1 = [1 -0.9]; pol2 = [1 -0.8];
G1 = tf(pol1,[1 0],-1);
G2 = tf(pol2,[1 0],-1);
w = linspace(0.001,0.5,1000);
xset('window',1);
bode([G1;G2],w);
G = tf(pol1,pol2,-1);
xset('window',2);
bode(G,w);
```

Scilab code Exa 7.4 A procedure to design lead controllers

```
// A procedure to design lead controllers, as
   explained in Fig. 7.12 on page 257.
// 7.4

exec('tf.sci',-1)

w = linspace(0.001,%pi,1000);
a = linspace(0.001,0.999,100);
lena = length(a);
omega = []; lead = [];
for i = 1:lena,
```

```
    zero = a(i);
    pole = 0.9*zero;
    sys = tf([1 -zero],[1 -pole],-1);
    frq = w/(2*%pi);
    [frq,repf]=repfreq(sys, frq);
    [db,phase] =dbphi(repf);
    [y,j] = max(phase);
    omega = [omega w(j)];
    lead = [lead y];
    comega = (pole+zero)/(pole*zero+1);
    clead = zero-pole;
    clead1 = sqrt((1-zero^2)*(1-pole^2));
    clead = clead/clead1;
//    [w(j) acos(comega) y atan(clead)*180/pi]
end
subplot(2,1,1), plot(lead,omega)
xtitle('','','Frequency, in radians'), xgrid;
halt;
subplot(2,1,2), plot(lead,a)
xtitle('','Lead generated, in degrees','Zero
   location'), xgrid;
```

Scilab code Exa 7.5 Bode plot of a lead controller

```
// Bode plot of a lead controller, as shown in Fig.
   7.13 on page 257.
// 7.5

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

w = linspace(0.001,0.5,1000);
G = tf([1 -0.8],[1 -0.24],-1);
bode(G,w)
```

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

```
// Verification of performance of lead controller on antenna system, as discussed in Example 7.3.
// 7.6

// Continuous time antenna model
a = 0.1;
F = [0 1;0 -a]; g = [0; a]; c = [1 0]; d = 0;
Ga = syslin('c',F,g,c,d); [ds,num,den] = ss2tf(Ga);
Num = clean(num); Den = clean(den);
Ts = 0.2;
G = dscr(Ga,Ts);

// lead controller
beta1 = 0.8;
N = [1 -0.9802]*(1-beta1)/(1-0.9802); Rc = [1 -beta1];

// simulation parameters using g_s_cl2.cos
gamm = 1; Sc = 1; Tc = 1; C = 0; D = 1;
st = 1; st1 = 0;
t_init = 0; t_final = 20;

// u1: -4 to 11
// y1: 0 to 1.4
exec('cosfil_ip.sci',-1);
[Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
[Np,Rcp] = cosfil_ip(N,Rc); // N/Rc
[Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
[Cp,Dp] = cosfil_ip(C,D); // C/D
```

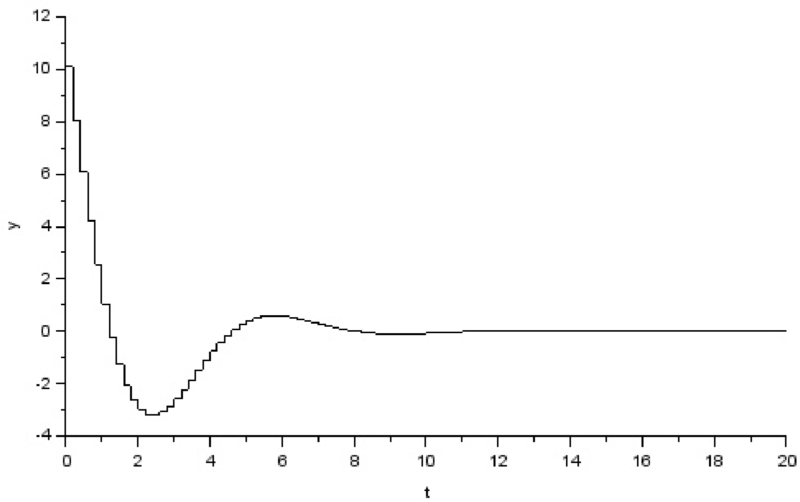

Figure 7.1: Verification of performance of lead controller on antenna system

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

```
// Illustration of system type, as explained in
   Example 7.10 on page 275.
// 7.7

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

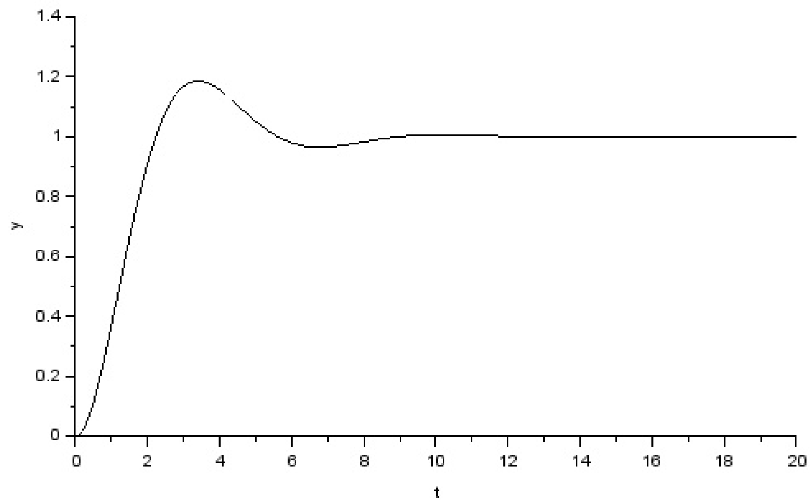

Figure 7.2: Verification of performance of lead controller on antenna system

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

// Plant
B = 1; A = [1 -1]; zk = [0 1]; Ts = 1; k = 1;
// Value of k absent in original code
// Specify closed loop characteristic polynomial
```

```
phi = [1 -0.5];

// Design the controller
reject_ramps = 1;
  if reject_ramps == 1,
    Delta = [1 -1]; // to reject ramps another Delta
  else
    Delta = 1; // steps can be rejected by plant
       itself
  end
[Rc,Sc] = pp_pid(B,A,k,phi,Delta);

// parameters for simulation using stb_disc.mdl
Tc = Sc; gamm = 1; N = 1;
C = 0; D = 1; N_var = 0;
st = 1; t_init = 0; t_final = 20;

[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

// Give appropriate path
//xcos('stb_disc.xcos');
```

Scilab code Exa 7.8 Solution to Aryabhatta identity

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

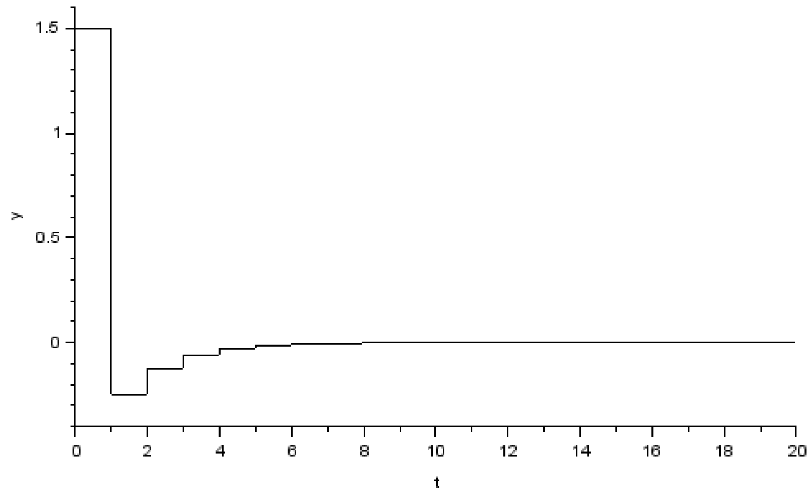

Figure 7.3: Illustration of system type

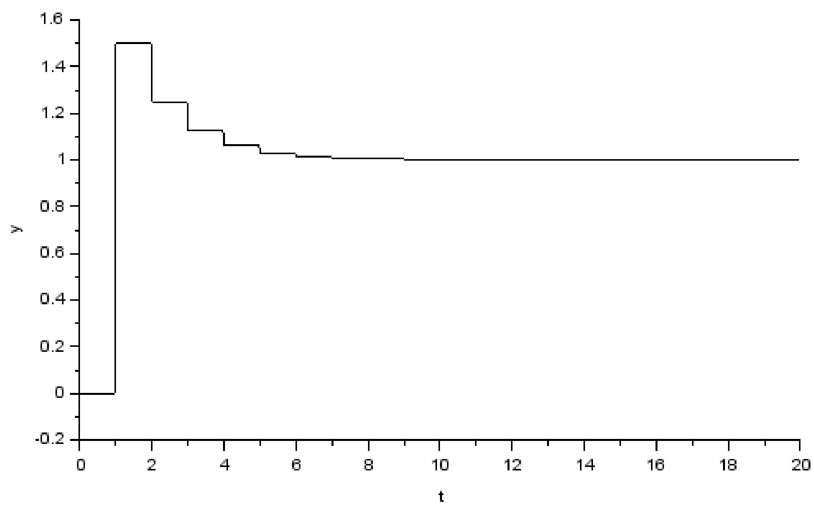

Figure 7.4: Illustration of system type

```
// 7.8

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

N = convol([0 1],[1 1]);
D = convol([1 -4],[1 -1]);
dN = 2; dD = 2;
C = [1 -1 0.5];
dC = 2;
[Y,dY,X,dX,B,dB,A,dA] = xdync(N,dN,D,dD,C,dC)
```

Scilab code Exa 7.9 Left coprime factorization

```
// Left coprime factorization as discussed in
   Example 7.13 on page 295.
// 7.9

exec('rowjoin.sci',-1);
exec('makezero.sci',-1);
exec('colsplit.sci',-1);
exec('clcoef.sci',-1);
exec('polsize.sci',-1);
exec('seshft.sci',-1);
exec('indep.sci',-1);
```

```
exec('move_sci.sci',-1);
exec('t1calc.sci',-1);
exec('left_prm.sci',-1);

D = [1 0 0 0 0 0
0 1 0 1 0 0
0 0 1 1 1 0];
N = [
1 0 0
0 1 0
0 0 1];
dD = 1;
dN = 0;
[B,dB,A,dA] = left_prm(N,dN,D,dD)
```

Scilab code Exa 7.10 Solution to polynomial equation

```
// Solution to polynomial equation, as discussed in
   Example 7.14 on page 295.
// 7.10

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

N = [0  4 0 1
```

```
        -1 8 0 3];
dN = 1;
D = [0 0  1 4 0 1
        0 0 -1 0 0 0];
dD = 2;
C = [1 0 1 1
      0 2 0 1];
dC = 1;
[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

```
// Continuous to discrete time transfer function
// 8.1

exec('tf.sci');

sys = tf(10,[5 1]);
sysd = ss2tf(dscr(sys,0.5));
```

Chapter 9

Pole Placement Controllers

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

```
// Pole placement controller for magnetically
   suspended ball problem, discussed in Example 9.3
   on page 331.
// 9.1

exec('myc2d.sci',-1);
exec('desired.sci',-1);
exec('zpowk.sci',-1);
exec('polsplit2.sci',-1);
exec('polsize.sci',-1);
exec('t1calc.sci',-1);
exec('indep.sci',-1);
exec('move_sci.sci',-1);
exec('colsplit.sci',-1);
exec('clcoef.sci',-1);
exec('cindep.sci',-1);
exec('polmul.sci',-1);
exec('seshft.sci',-1);
exec('makezero.sci',-1);
exec('xdync.sci',-1);
```

```
exec('left_prm.sci',-1);
exec('rowjoin.sci',-1);
exec('pp_basic.sci',-1);
exec('polyno.sci',-1);
exec('cosfil_ip.sci',-1);

// Magnetically suspended ball problem
// Operating conditions
M = 0.05; L = 0.01; R = 1; K = 0.0001; g = 9.81;

//Equilibrium conditions
hs = 0.01; is = sqrt(M*g*hs/K);

// State space matrices
a21 = K*is^2/M/hs^2; a23 = - 2*K*is/M/hs; a33 = - R/L;
b3 = 1/L;
a1 = [0 1 0; a21 0 a23; 0 0 a33];
b1 = [0; 0; b3]; c1 = [1 0 0]; d1 = 0;

// Transfer functions
G = syslin('c',a1,b1,c1,d1); Ts = 0.01;
[B,A,k] = myc2d(G,Ts);

//polynomials are returned
[Ds,num,den] = ss2tf(G);
num = clean(num); den = clean(den);

// Transient specifications
rise = 0.15; epsilon = 0.05;
phi = desired(Ts,rise,epsilon);

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

// Setting up simulation parameters for basic.xcos
st = 0.0001; // desired change in h, in m.
t_init = 0; // simulation start time
```

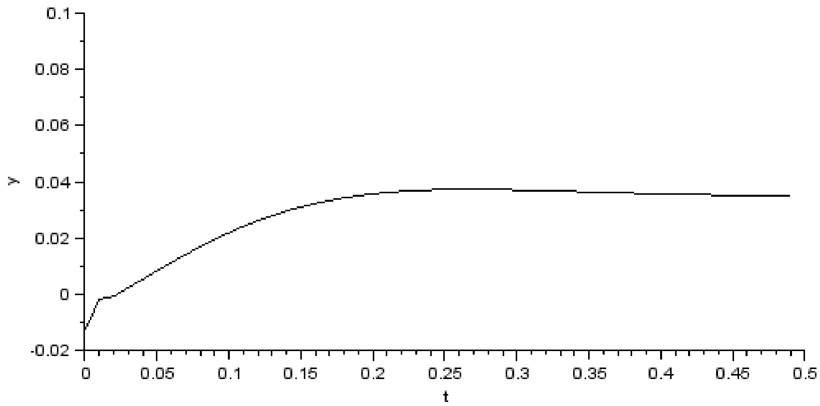

Figure 9.1: Pole placement controller for magnetically suspended ball problem

```
t_final = 0.5; // simulation end time

// Setting up simulation parameters for c_ss_cl.xcos
N_var = 0; xInitial = [0 0 0]; N = 1; C = 0; D = 1;

[Tc1,Rc1] = cosfil_ip(Tc,Rc); // Tc/Rc
[Sc2,Rc2] = cosfil_ip(Sc,Rc); // Sc/Rc

[Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
[Np,Rcp] = cosfil_ip(N,Rc); // 1/Rc
[Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
[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

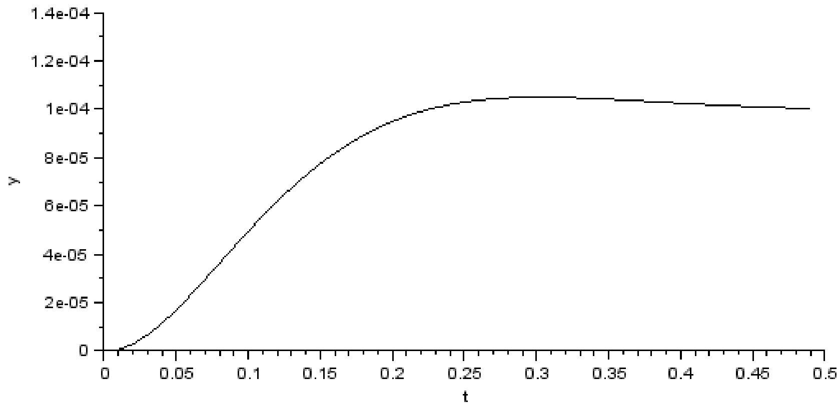

Figure 9.2: Pole placement controller for magnetically suspended ball problem

Scilab code Exa 9.2 Discretization of continuous transfer function

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

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

```
H = ss2tf(dscr(G,Ts));
num1 = coeff(H('num'));
den1 = coeff(H('den'));//−−−−−−−−−−−−−
A = den1(length(den1):-1:1);
num2 = num1(length(num1):-1:1);  //flip
nonzero = find(num1);
first_nz = nonzero(1);
B = num2(first_nz:length(num2)); //−−−−−−−−−−−−−
k = length(den1) - length(num1);
endfunction
```

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

```
// Procedure to split a polynomial into good and bad
   factors, as discussed in Sec. 9.2.
// 9.3
// function [goodpoly,badpoly] = polsplit2(fac,a)
// Splits a scalar polynomial of z^{−1} into good
   and bad
// factors.
// Input is a polynomial in increasing degree of z
   ^{−1}
// Optional input is a, where a <= 1.
// Factor that has roots of z^{−1} outside a is
   called
// good and the rest bad.
// If a is not specified, it will be assumed as
   1−1.0e−5

function [goodpoly,badpoly] = polsplit2(fac,a)
if argn(2) == 1, a = 1-1.0e-5; end
if a>1 error('good polynomial is unstable'); end
fac1 = poly(fac(length(fac):-1:1),'z','coeff');
rts1 = roots(fac1);
```

```
rts = rts1(length(rts1):-1:1);

// extract good and bad roots
badindex = find(abs(rts)>=a); // mtlb_find has been
   replaced by find
badpoly = coeff(poly((rts(badindex)),"z","roots"));
goodindex = find(abs(rts)<a); // mtlb_find has been
   replaced by find
goodpoly = coeff(poly(rts(goodindex),"z","roots"));

// scale by equating the largest terms
[m,index] = max(abs(fac));
goodbad = convol(goodpoly,badpoly);
goodbad1 = goodbad(length(goodbad):-1:1);//--
factor1 = fac(index)/goodbad1(index);//--
goodpoly = goodpoly * factor1;
goodpoly = goodpoly(length(goodpoly):-1:1);
badpoly = badpoly(length(badpoly):-1:1);
endfunction;
```

Scilab code Exa 9.4 Calculation of desired closed loop characteristic polynomial

```
// Calculation of desired closed loop characteristic
    polynomial, as discussed in Sec. 7.7.
// 9.4

// function [phi,dphi] = desired(Ts,rise,epsilon)
// Based on transient requirements,
// calculates closed loop characteristic polynomial

function [phi,dphi] = desired(Ts,rise,epsilon)
Nr = rise/Ts; omega = %pi/2/Nr; rho = epsilon^(omega
   /%pi);
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

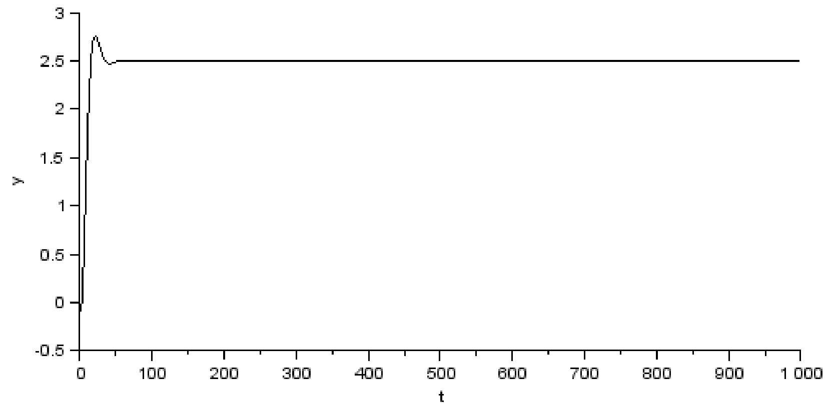

Figure 9.3: Simulation of closed loop system with an unstable controller

```
// Evaluates z^-k.
// 9.6

function [zk,dzk] = zpowk(k)
zk = zeros(1,k+1); zk(1,k+1) = 1;
dzk = k;
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

```
// 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
```

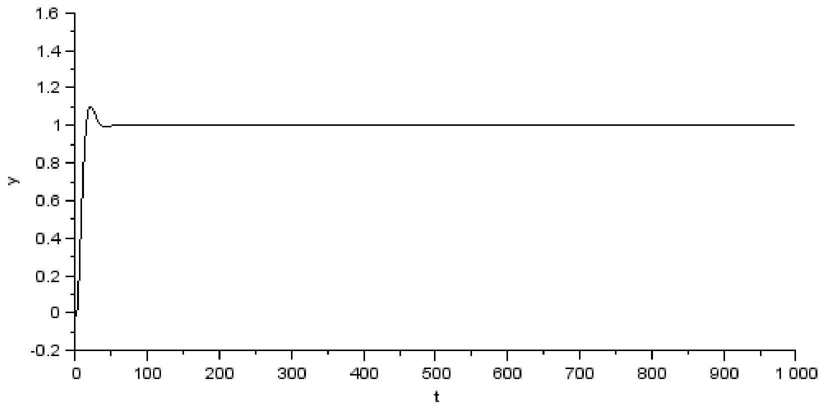

Figure 9.4: Simulation of closed loop system with an unstable controller

```
//y1: -0.1 to 1.2
st = 1.0; // Desired change in setpoint
t_init = 0; // Simulation start time
t_final = 1000; // Simulation end time

// Simulation parameters for stb_disc.xcos
N_var = 0; C = 0; D = 1; N = 1;

[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

[Tcp,Rcp] = cosfil_ip(Tc,Rc); // Tc/Rc
[Scp_b,Rcp_b] = cosfil_ip(Sc,Rc); // Sc/Rc
```

Scilab code Exa 9.8 Pole placement controller using internal model principle

```
// Pole placement controller using internal model
    principle , as discussed in Sec. 9.4.
// 9.8

// function [Rc,Sc,Tc,gamma,phit] = pp_im(B,A,k,phi,
    Delta)
// Calculates 2-DOF pole placement controller.

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

// Setting up and solving Aryabhatta identity
[Ag,Ab] = polsplit3(A); dAb = length(Ab) - 1;
[Bg,Bb] = polsplit3(B); dBb = length(Bb) - 1;

[zk,dzk] = zpowk(k);

[N,dN] = polmul(Bb,dBb,zk,dzk);
dDelta = length(Delta)-1;
[D,dD] = polmul(Ab,dAb,Delta,dDelta);
dphi = length(phi)-1;

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

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

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

```
// Pole placement controller, with internal model of a step, for the magnetically suspended ball problem, as discussed in Example 9.8 on page 339.
// 9.9

// PP control with internal model for ball problem
exec('desired.sci',-1);
exec('pp_im.sci',-1);
exec('myc2d.sci',-1);
exec('polsplit3.sci',-1);
exec('zpowk.sci',-1);
exec('rowjoin.sci',-1);
exec('left_prm.sci',-1);
exec('t1calc.sci',-1);
exec('indep.sci',-1);
exec('cindep.sci',-1);
exec('seshft.sci',-1);
exec('makezero.sci',-1);
exec('move_sci.sci',-1);
exec('colsplit.sci',-1);
exec('clcoef.sci',-1);
exec('polmul.sci',-1);
exec('polsize.sci',-1);
exec('xdync.sci',-1);
exec('cosfil_ip.sci',-1);
exec('polyno.sci',-1);

// Operating conditions
M = 0.05; L = 0.01; R = 1; K = 0.0001; g = 9.81;

// Equilibrium conditions
hs = 0.01; is = sqrt(M*g*hs/K);

// State space matrices
a21 = K*is^2/M/hs^2; a23 = - 2*K*is/M/hs; a33 = - R/L;
b3 = 1/L;
a1 = [0 1 0; a21 0 a23; 0 0 a33];
```

```
b1 = [0; 0; b3]; c1 = [1 0 0]; d1 = 0;

// Transfer functions
G = syslin('c',a1,b1,c1,d1); Ts = 0.01; [B,A,k] =
    myc2d(G,Ts);

// Transient specifications
rise = 0.1; epsilon = 0.05;
phi = desired(Ts,rise,epsilon);

// Controller design
Delta = [1 -1];  //internal model of step used
[Rc,Sc,Tc,gamm] = pp_im(B,A,k,phi,Delta);

// simulation parameters for c_ss_cl.xcos
st = 0.0001;  //desired change in h, in m.
t_init = 0; // simulation start time
t_final = 0.5; //simulation end time
xInitial = [0 0 0];
N = 1; C = 0; D = 1; N_var = 0;

[Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
[Np,Rcp] = cosfil_ip(N,Rc); // 1/Rc
[Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
[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

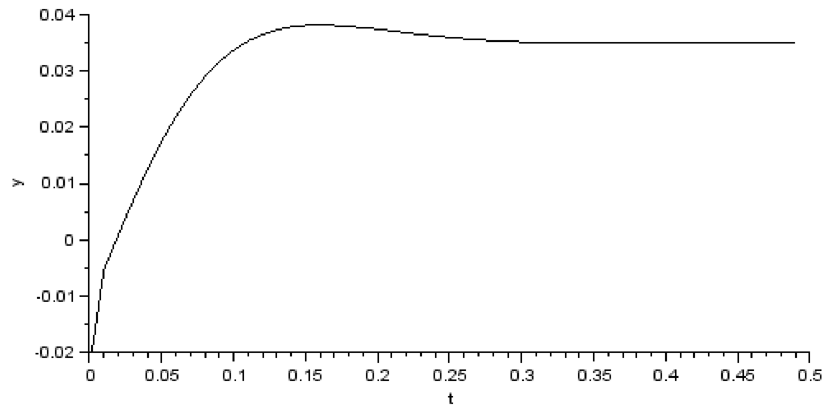

Figure 9.5: Pole placement controller with internal model of a step for the magnetically suspended ball problem

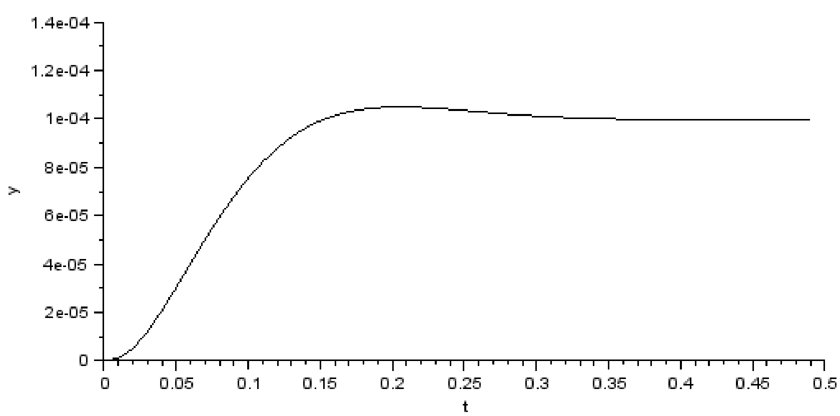

Figure 9.6: Pole placement controller with internal model of a step for the magnetically suspended ball problem

```
// Pole placement controller IBM Lotus Domino server
    , discussed in Example 9.9 on page 341.
// 9.10

exec('desired.sci',-1);
exec('pp_im.sci',-1);
exec('zpowk.sci',-1);
exec('cosfil_ip.sci',-1);
exec('polsplit3.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('polmul.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);

// Control of IBM lotus domino server
// Transfer function
B = 0.47; A = [1 -0.43]; k = 1;
[zk,dzk] = zpowk(k);

// Transient specifications
rise = 10; epsilon = 0.01; Ts = 1;
phi = desired(Ts,rise,epsilon);

// Controller design
Delta = [1 -1];  // internal model of step used
[Rc,Sc,Tc,gamm] = pp_im(B,A,k,phi,Delta);

```

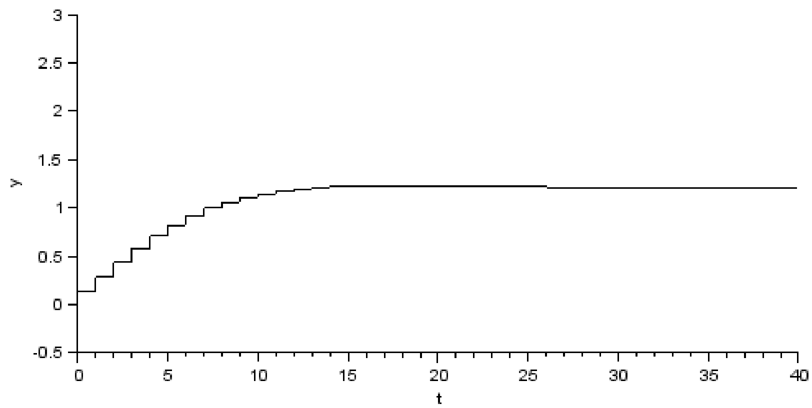

Figure 9.7: Pole placement controller IBM Lotus Domino server

```
// Simulation parameters for stb_disc.xcos
st = 1; // desired change
t_init = 0; // simulation start time
t_final = 40; // simulation end time
C = 0; D = 1; N_var = 0;

[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
```

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

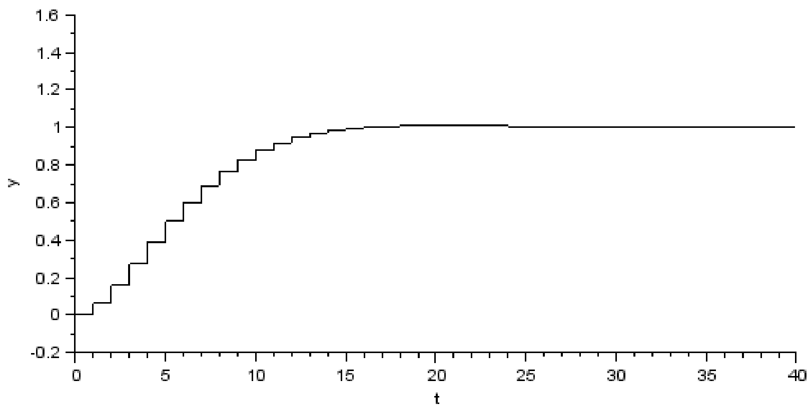

Figure 9.8: Pole placement controller IBM Lotus Domino server

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

Scilab code Exa 9.11 Pole placement controller for motor problem

```
// Pole placement controller for motor problem,
    discussed in Example 9.10 on page 343.
// 9.11

exec('desired.sci',-1);
exec('pp_im.sci',-1);
exec('myc2d.sci',-1);
exec('cosfil_ip.sci',-1);
exec('polsplit3.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);

// Motor control problem
// Transfer function
a1 = [-1 0; 1 0]; b1 = [1; 0]; c1 = [0 1]; d1 = 0;
G = syslin('c',a1,b1,c1,d1); Ts = 0.25;
[B,A,k] = myc2d(G,Ts);

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

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

// simulation parameters for c_ss_cl.xcos
st = 1;  //desired change in position
t_init = 0;  //simulation start time
t_final = 10;  //simulation end time
xInitial = [0 0];  //initial conditions
N = 1; C = 0; D = 1; N_var = 0;

[Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
[Np,Rcp] = cosfil_ip(N,Rc); // 1/Rc
[Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
[Cp,Dp] = cosfil_ip(C,D); // C/D
```

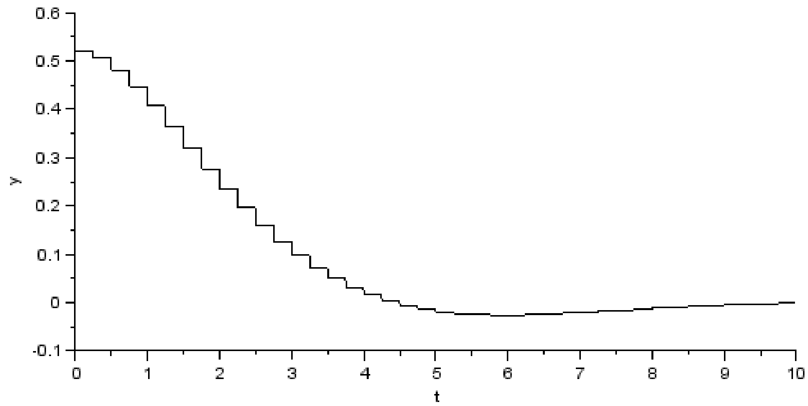

Figure 9.9: Pole placement controller for motor problem

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

```
// 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.
// 9.12

// function [goodpoly,badpoly] = polsplit3(fac,a)
// Splits a scalar polynomial of z^{-1} into good
   and bad
// factors. Input is a polynomial in increasing
   degree of
// z^{-1}.  Optional input is a, where a <= 1.
```

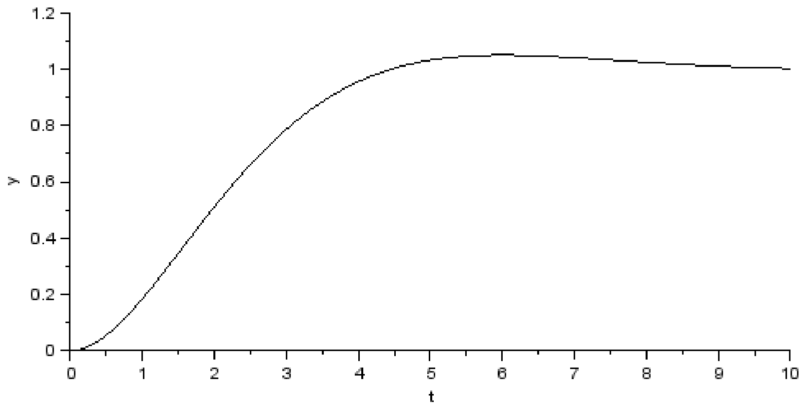

Figure 9.10: Pole placement controller for motor problem

```
// Factors that have roots outside a circle of radius a or
// with negative roots will be called bad and the rest
// good. If a is not specified, it will be assumed as 1.

function [goodpoly,badpoly] = polsplit3(fac,a)
if argn(2) == 1, a = 1; end
if a>1 error('good polynomial also is unstable'); end
fac1 = poly(fac(length(fac):-1:1),'z','coeff');
rts = roots(fac1);
rts = rts(length(rts):-1:1);

// extract good and bad roots
badindex = mtlb_find((abs(rts)>=a-1.0e-5)|(real(rts)<-0.05));
badpoly = coeff(poly(rts(badindex),'z'));
goodindex = mtlb_find((abs(rts)<a-1.0e-5)&(real(rts)>=-0.05));
goodpoly = coeff(poly(rts(goodindex),'z'));

```

```
// scale by equating the largest terms
[m,index] = max(abs(fac));
goodbad = convol(goodpoly,badpoly);
goodbad = goodbad(length(goodbad):-1:1);
factor1 = fac(index)/goodbad(index);
goodpoly = goodpoly * factor1;
goodpoly = goodpoly(length(goodpoly):-1:1);
badpoly = badpoly(length(badpoly):-1:1);
endfunction;
```

Scilab code Exa 9.13 Pole placement controller without intra sample oscillations

```
// Pole placement controller without intra sample
    oscillations, as discussed in Sec. 9.5.
// 9.13

// function [Rc,Sc,Tc,gamma,phit] = pp_im2(B,A,k,phi
    ,Delta,a)
// 2-DOF PP controller with internal model of Delta
    and without
// hidden oscillations

function [Rc,Sc,Tc,gamm,phit] = pp_im2(B,A,k,phi,
    Delta,a)

if argn(2) == 5, a = 1; end
dphi = length(phi)-1;

// Setting up and solving Aryabhatta identity
[Ag,Ab] = polsplit3(A,a); dAb = length(Ab) - 1;
[Bg,Bb] = polsplit3(B,a); dBb = length(Bb) - 1;

[zk,dzk] = zpowk(k);

```

```
[N,dN] = polmul(Bb,dBb,zk,dzk);
dDelta = length(Delta)-1;
[D,dD] = polmul(Ab,dAb,Delta,dDelta);

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

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

// Total characteristic polynomial
phit = convol(phi,convol(Ag,Bg));
endfunction;
```

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

Scilab code Exa 9.14 Controller design

```
// Controller design for the case study presented in
    Example 9.12 on page 347.
// 9.14

exec('tf.sci',-1);
exec('desired.sci',-1);
exec('zpowk.sci',-1);
exec('myc2d.sci',-1);
exec('polsplit3.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('pp_im2.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('cosfil_ip.sci',-1);

num = 200;
den = convol([0.05 1],[0.05 1]);
den = convol([10 1],den);
G = tf(num,den); Ts = 0.025;
num = G('num'); den = G('den');
// iodel = 0;
[B,A,k] = myc2d(G,Ts);
[zk,dzk] = zpowk(k); //int1 = 0;

// Transient specifications
a = 0.9; rise = 0.24; epsilon = 0.05;
phi = desired(Ts,rise,epsilon);

// Controller design
Delta = [1 -1]; // internal model of step is present
[Rc,Sc,Tc,gamm] = pp_im2(B,A,k,phi,Delta,a);

// margin calculation
Lnum = convol(Sc,convol(B,zk));
Lden = convol(Rc,A);
L = tf(Lnum,Lden,Ts);
Gm = g_margin(L); //---- Does not match
   ---------------- (in dB)
Pm = p_margin(L); //---- Convergence problem
   ---------------- (in degree)

num1 = 100; den1 = [10 1];
Gd = tf(num1,den1);  //--------
```

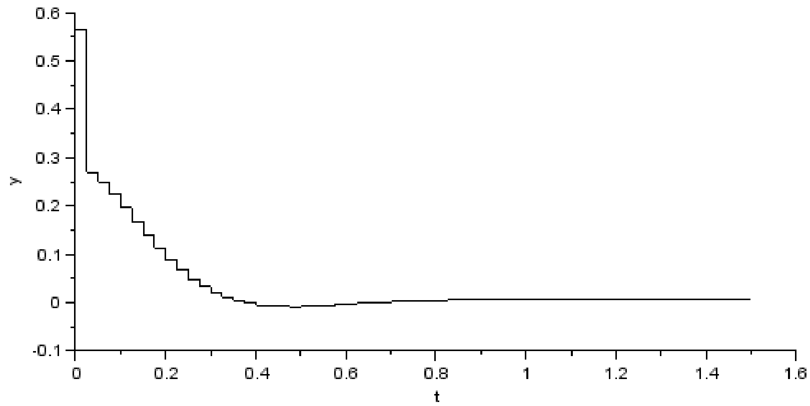

Figure 9.11: Controller design

```
[C,D,k1] = myc2d(Gd,Ts);
[zk,dzk] = zpowk(k);
C = convol(C,zk);

// simulation parameters g_s_cl2.xcos -------------
N = 1;
st = 1; // desired change in setpoint
st1 = 0; // magnitude of disturbance
t_init = 0; // simulation start time
t_final = 1.5; // simulation end time

[Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
[Np,Rcp] = cosfil_ip(N,Rc); // N/Rc
[Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
[Cp,Dp] = cosfil_ip(C,D); // C/D
```

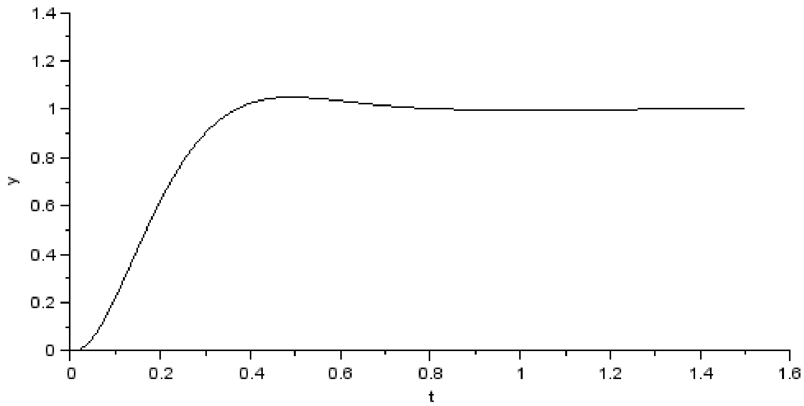

Figure 9.12: Controller design

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

```
// Evaluation of continuous time controller for the
   case study presented in Example 9.13 on page 349.
// 9.15

clear
exec('tf.sci',-1);
exec('myc2d.sci',-1);
exec('zpowk.sci',-1);
exec('cosfil_ip.sci',-1);
exec('polyno.sci',-1);

```

```
num = 200;
den = convol([0.05 1],[0.05 1]);
den = convol([10 1],den);
G = tf(num,den); Ts = 0.005;
[B,A,k] = myc2d(G,Ts);
[zk,dzk] = zpowk(k); //int = 0;

// Sigurd's feedback controller'
numb = 0.5*convol([1 2],[0.05 1]);
denb = convol([1 0],[0.005 1]);
Gb = tf(numb,denb);
[Sb,Rb,kb] = myc2d(Gb,Ts);
[zkb,dzkb] = zpowk(kb);
Sb = convol(Sb,zkb);

// Sigurd's feed forward controller'
numf = [0.5 1];
denf = convol([0.65 1],[0.03 1]);
Gf = tf(numf,denf);
[Sf,Rf,kf] = myc2d(Gf,Ts);
[zkf,dzkf] = zpowk(kf);
Sf = convol(Sf,zkf);

// Margins
simp_mode(%f);
L = G*Gb;
Gm = g_margin(L); // -------
Pm = p_margin(L); // -------
Lnum = convol(Sb,convol(zk,B));
Lden = convol(Rb,A);
L = tf(Lnum,Lden,Ts);
DGm = g_margin(L); // -------
DPm = p_margin(L); // -------

// Noise
num1 = 100; den1 = [10 1];

// simulation parameters for
```

```
// entirely continuous simulation: g_s_cl3.xcos
// hybrid simulation: g_s_cl6.xcos
st = 1; // desired change in setpoint
st1 = 0;
t_init = 0; // simulation start time
t_final = 5; // simulation end time

num = polyno(num,'s'); den = polyno(den,'s');
Numb = polyno(numb,'s'); Denb = polyno(denb,'s');
Numf = polyno(numf,'s'); Denf = polyno(denf,'s');
Num1 = polyno(num1,'s'); Den1 = polyno(den1,'s');

[Sbp,Rbp] = cosfil_ip(Sb,Rb);
[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

```
// System type with 2-DOF controller. It is used to
    arrive at the results Example 9.14.
// 9.16

exec('polsplit3.sci',-1);
exec('polmul.sci',-1);
exec('polsize.sci',-1);
exec('pp_im.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('zpowk.sci',-1);
exec('cosfil_ip.sci',-1);
exec('polyno.sci',-1);

B = 1; A = [1 -1]; k = 1; zk = zpowk(k); Ts = 1;
phi = [1 -0.5];

Delta = 1; // Choice of internal model of step
[Rc,Sc,Tc,gamm] = pp_im(B,A,k,phi,Delta);

// simulation parameters for stb_disc.xcos
st = 1; // desired step change
t_init = 0; // simulation start time
t_final = 20; // simulation end time
xInitial = [0 0];
C = 0; D = 1; N_var = 0;

[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
```

Scilab code Exa 9.17 Illustrating the benefit of cancellation

```
// Illustrating the benefit of cancellation. It is
   used to arrive at the results of Example 9.15.
// 9.17

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

```

```
// simulation parameters for stb_disc.mdl
[zk,dzk] = zpowk(k);
st = 1; // desired step change
t_init = 0; // simulation start time
t_final = 20; // simulation end time
xInitial = [0 0];
C = 0; D = 1; N_var = 0;

[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
```

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

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

```
// 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.
// 9.18

exec('pp_im2.sci',-1);
exec('desired.sci',-1);
exec('zpowk.sci',-1);
exec('cosfil_ip.sci',-1);
exec('polsplit3.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('polyno.sci',-1);
exec('cindep.sci',-1);
exec('poladd.sci',-1);

//  Transfer function
B = 0.47; A = [1 -0.43]; k = 1;
[zk,dzk] = zpowk(k);

// Transient specifications
rise = 10; epsilon = 0.01; Ts = 1;
phi = desired(Ts,rise,epsilon);

// Controller design
delta = [1 -1]; // internal model of step used
[Rc,Sc,Tc,gamm,F] = pp_im2(B,A,k,phi,delta);

// Study of Antiwindup Controller

key = x_choose(['Simulate without any saturation limits';
         'Simulate saturation, but do not use AWC';
         'Simulate saturation with AWC in place';
         'Simulate with AWC, without saturation limits'],...
         ['Please choose one of the following']);

if key ==0
  disp('Invalid choice');
  return;
elseif key == 1
```

```
  U = 2; L = -2; P = 1; F = Rc; E = 0; PSc = Sc; PTc = Tc;
elseif key == 2
  U = 1; L = -1; P = 1; F = Rc; E = 0; PSc = Sc; PTc = Tc;
else
  if key == 3 // Antiwindup controller and with saturation
    U = 1; L = -1;
  elseif key == 4 // Antiwindup controller, but no saturation
    U = 2; L = -2;
  end
  P = A;
  dF = length(F) - 1;
  PRc = convol(P,Rc); dPRc = length(PRc) - 1;
  [E,dE] = poladd(F,dF,-PRc,dPRc);
  PSc = convol(P,Sc); PTc = convol(P,Tc);
end

// Setting up simulation parameters for stb_disc_sat
t_init = 0; // first step begins
st = 1; // height of first step
t_init2 = 500; // second step begins
st2 = -2; // height of second step
t_final = 1000; // simulation end time
st1 = 0; // no disturbance input
C = 0; D = 1; N_var = 0;

[PTcp1,PTcp2] = cosfil_ip(PTc,1); // PTc/1
[Fp1,Fp2] = cosfil_ip(1,F); // 1/F
[Ep,Fp] = cosfil_ip(E,F); // E/F
[PScp1,PScp2] = cosfil_ip(PSc,1); // PSc/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
```

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

```
// Demonstration of usefulness of negative PID
   parameters, discussed in Example 9.17 on page
   361.
// 9.19

exec('iodelay.sci',-1);
exec('delc2d.sci',-1);
exec('desired.sci',-1);
exec('pp_pid.sci',-1);
exec('cosfil_ip.sci',-1);
exec('tf.sci',-1);
exec('flip.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);

// Discretize the continuous plant
num = 1; den = [2 1]; tau = 0.5;
G1 = tf(num,den);
G = iodelay(G1,tau);
Ts = 0.5;
[B,A,k] = delc2d(G,G1,Ts);

// Specify transient requirements
epsilon = 0.05; rise = 5;
phi = desired(Ts,rise,epsilon);

// Design the controller
Delta = [1 -1];
[Rc,Sc] = pp_pid(B,A,k,phi,Delta);

// parameters for simulation using g_s_cl
Tc = Sc; gamm = 1; N = 1;
C = 0; D = 1; N_var = 0;
st = 1; t_init = 0; t_final = 20;

[Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
[Np,Rcp] = cosfil_ip(N,Rc); // N/Rc
[Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
[Cp,Dp] = cosfil_ip(C,D); // C/D
Num = numer(G1);
Den = denom(G1);
```

Scilab code Exa 9.20 PID controller design

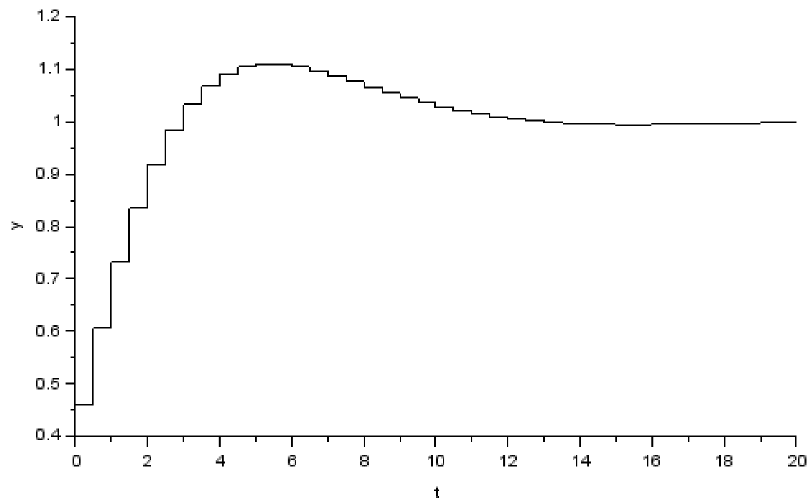

Figure 9.13: Demonstration of usefulness of negative PID parameters

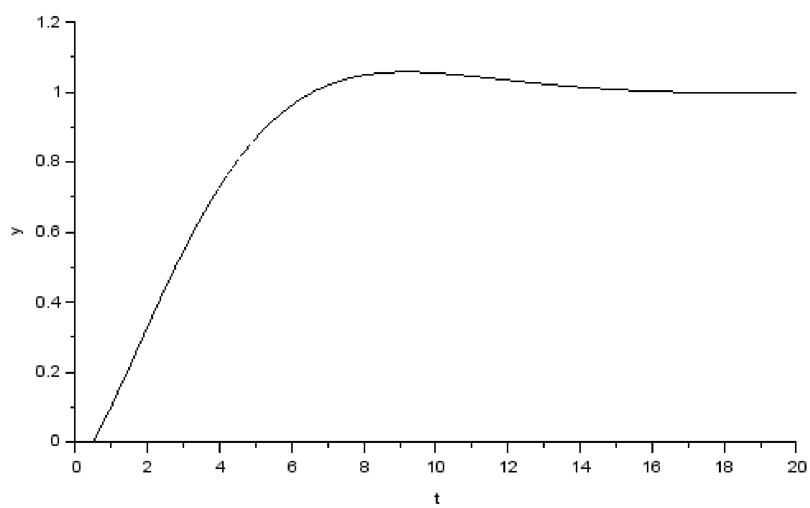

Figure 9.14: Demonstration of usefulness of negative PID parameters

```
// Solution to Aryabhatta's identity arising in PID controller design, namely Eq. 9.37 on page 363.
// 9.20

function [Rc,Sc] = pp_pid(B,A,k,phi,Delta)

// Setting up and solving Aryabhatta identity
dB = length(B) - 1; dA = length(A) - 1;
[zk,dzk] = zpowk(k);
[N,dN] = polmul(B,dB,zk,dzk);
dDelta = length(Delta)-1;
[D,dD] = polmul(A,dA,Delta,dDelta);
dphi = length(phi)-1;
[Sc,dSc,R,dR] = xdync(N,dN,D,dD,phi,dphi);
Rc = convol(R,Delta);
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

```
// DC motor with PID control, tuned through pole placement technique, as in Example 9.18.
// 9.21

exec('desired.sci',-1);
exec('pp_pid.sci',-1);
exec('cosfil_ip.sci',-1);
exec('pd.sci',-1);
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
```

```
st = 1;  // desired change in position
t_init = 0; // simulation start time
t_final = 20; // simulation end time
st1 = 0;

// continuous controller simulation: g_s_cl3.xcos
num1 = 0; den1 = 1;

// discrete controller simulation: g_s_cl2.xcos
// u1: -0.1 to 0.8
// y1: 0 to 1.4
C = 0; D = 1; N = 1; gamm = 1; Tc = Sc;

[Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
[Np,Rcp] = cosfil_ip(N,Rc); // N/Rc
[Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
[Cp,Dp] = cosfil_ip(C,D); // C/D
Numb = polyno(numb,'s');
Denb = polyno(denb,'s');
Numf = polyno(numf,'s');
Denf = polyno(denf,'s');
Num1 = polyno(num1,'s');
Den1 = polyno(den1,'s');
```

Scilab code Exa 9.22 PD control law from polynomial coefficients

```
// PD control law from polynomial coefficients, as
   explained in Sec. 9.8.
// 9.22

function [K,taud,N] = pd(Rc,Sc,Ts)
```

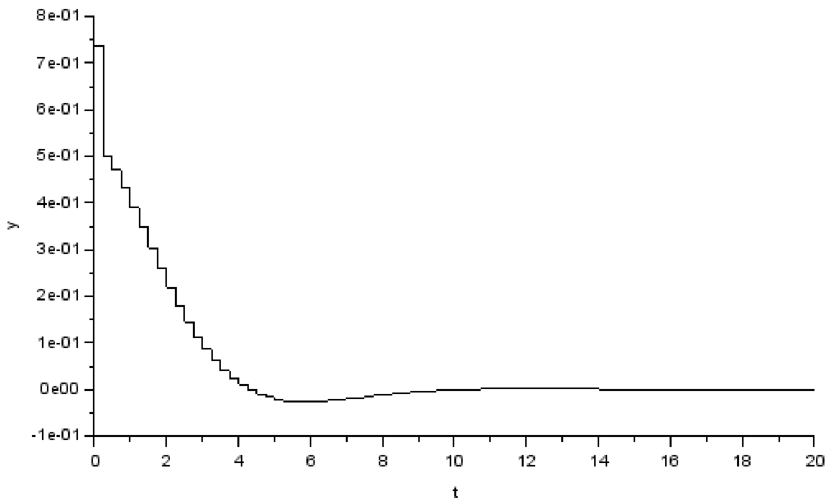

Figure 9.15: DC motor with PID control tuned through pole placement technique

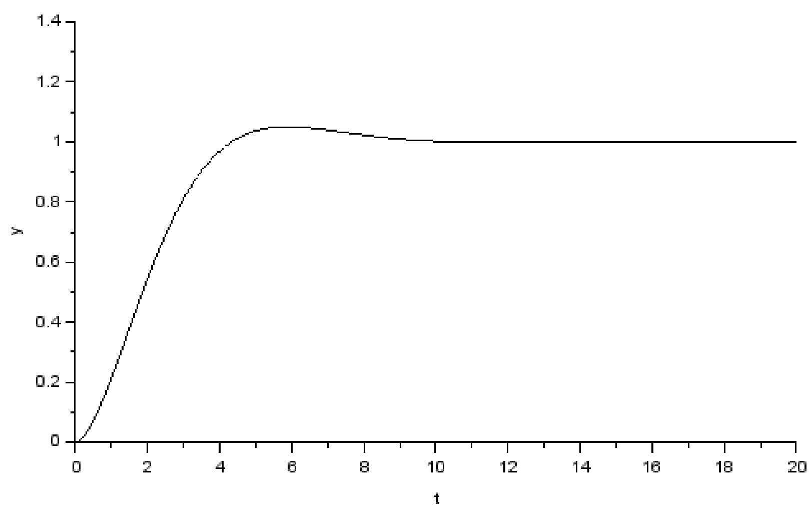

Figure 9.16: DC motor with PID control tuned through pole placement technique

```

// Both Rc and Sc have to be degree one polynomials

s0 = Sc(1); s1 = Sc(2);
r1 = Rc(2);
K = (s0+s1)/(1+r1);
N = (s1-s0*r1)/r1/(s0+s1);
taudbyN = -Ts*r1/(1+r1);
taud = taudbyN * N;
endfunction;
```

Chapter 10

Special Cases of Pole Placement Control

Scilab code Exa 10.1 Effect of delay in control performance

```
// Effect of delay in control performance
// 10.1

exec('zpowk.sci',-1);
exec('pp_im.sci',-1);
exec('cosfil_ip.sci',-1);
exec('polsplit3.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);

Ts = 1; B = 0.63; A = [1 -0.37];
k = input('Enter the delay as an integer: ');
if k<=0, k = 1; end
[zk,dzk] = zpowk(k);

// Desired transfer function
phi = [1 -0.5];
delta = 1; // internal model of step introduced

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

// simulation parameters for stb_disc.xcos
// y1: 0 to 1; u1: 0 to 1.2
st = 1.0; // desired change in setpoint
t_init = 0; // simulation start time
t_final = 20; // simulation end time

// simulation parameters for stb_disc.xcos
N_var = 0; C = 0; D = 1; N = 1;

[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
```

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
```

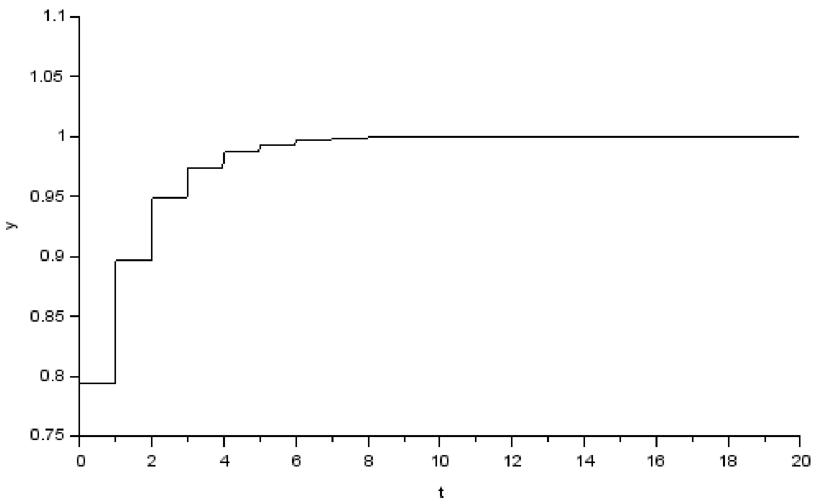

Figure 10.1: Smith predictor for paper machine control

```
t_init = 0; // simulation start time
t_final = 20; // simulation end time

// simulation parameters for smith_disc.xcos
N_var = 0; C = 0; D = 1; N = 1;

[Tcp1,Tcp2] = cosfil_ip(Tc,1); // Tc/1
[Rcp1,Rcp2] = cosfil_ip(1,Rc); // 1/Rc
[Scp1,Scp2] = cosfil_ip(Sc,1); // Sc/1
[Bdp,Ap] = cosfil_ip(Bd,A); // Bd/Ad
[zkdp1,zkdp2] = cosfil_ip(zkd,1); // zkd/1
[mzkdp1,mzkdp2] = cosfil_ip(mzkd,1); // mzkd/1
[Cp,Dp] = cosfil_ip(C,D); // C/D
```

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

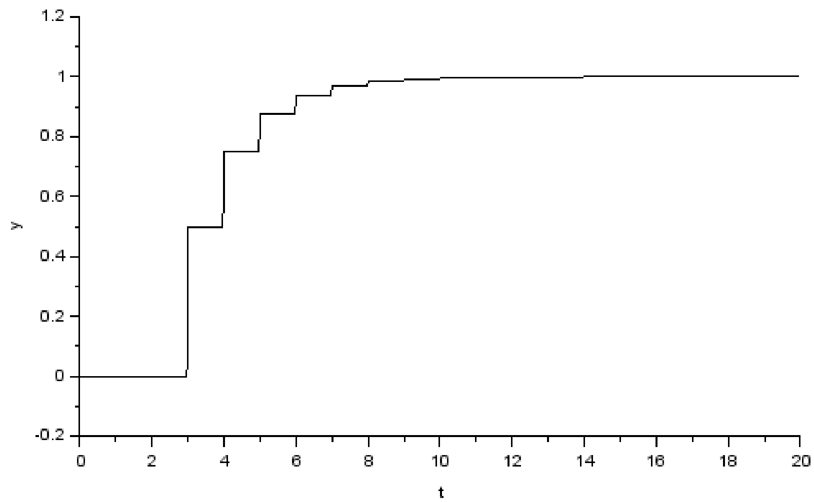

Figure 10.2: Smith predictor for paper machine control

Scilab code Exa 10.3 Splitting a polynomial B

```
// Splitting a polynomial B(z)
// 10.3
// Splits a polynomial B into good, nonminimum with
// positive real & with negative real parts.
// All are returned in polynomial form.
// Gain is returned in Kp and delay in k.

function [Kp,k,Bg,Bnmp,Bm] = imcsplit(B,polynomial)
k = 0;
Kp = 1;
if(polynomial)
  rts = roots(B);
```

```
  Kp = sum(B)/sum(coeff(poly(rts,'z')));
else
  rts = B;
end
Bg = 1; Bnmp = 1; Bm = 1;
for i = 1:length(rts),
    rt = rts(i);
    if rt == 0,
       k = k+1;
    elseif (abs(rt)<1 & real(rt)>=0)
      Bg = convol(Bg,[1 -rt]);
    elseif (abs(rt)>=1 & real(rt)>=0)
      Bnmp = convol(Bnmp,[1 -rt]);
    else
      Bm = convol(Bm,[1 -rt]);
    end
end
```

Scilab code Exa 10.4 Design of internal model controller

```
// Design of internal model controller
// 10.4
// 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] = imc_stable1(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);
```

```
k = k+d;
endfunction;
```

Scilab code Exa 10.5 Flipping a vector

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

Scilab code Exa 10.6 IMC design for viscosity control problem

```
// IMC design for viscosity control problem
// 10.6

exec('imc_stable1.sci',-1);
exec('zpowk.sci',-1);
exec('imcsplit.sci',-1);
exec('flip.sci',-1);

B = [0.51 1.21];
A = [1 -0.44];
k = 1;
alpha = 0.5;

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

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

```
// IMC design for the control of van de Vusse
   reactor
// 10.7

exec('tf.sci');
exec('myc2d.sci');
exec('imc_stable1.sci');
exec('imcsplit.sci',-1);
exec('flip.sci',-1);
exec('zpowk.sci',-1);

num = [-1.117 3.1472]; den = [1 4.6429 5.3821];
G = tf(num,den);
Ts = 0.1;
[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 = 10; st = 1; Nvar = 0;
```

Scilab code Exa 10.8 IMC design for an example by Lewin

```
// IMC design for Lewin's example
// 10.8

exec('tf.sci');
exec('myc2d.sci');
exec('imc_stable1.sci');
exec('zpowk.sci',-1);
exec('imcsplit.sci',-1);
exec('flip.sci',-1);

```

```
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);
```

```
k = k+d;

[zk,dzk] = zpowk(k);
Bf = (1-alpha);
Af = [1 -alpha];
S = convol(Bf,A);
R1 = convol(Af,convol(Bnmpr,Bms));
R2 = convol(zk,convol(Bf,convol(Bnmp,Bm)));

[R,dR] = poladd(R1,length(R1)-1,-R2,length(R2)-1);
R = convol(Bg,R);
endfunction;
```

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

```
// Design of conventional controller for van de
   Vusse reactor problem
// 10.10

exec('tf.sci');
exec('myc2d.sci');
exec('imcsplit.sci',-1);
exec('imc_stable.sci');
exec('zpowk.sci',-1);
exec('flip.sci',-1);
exec('poladd.sci',-1);
exec('polsize.sci',-1);

num = [-1.117 3.1472]; den = [1 4.6429 5.3821];
G = tf(num,den);
Ts = 0.1;
[B,A,k] = myc2d(G,Ts);
alpha = 0.5;
[k,HiN,HiD,R,S] = imc_stable(B,A,k,alpha);
```

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

Chapter 11

Minimum Variance Control

Scilab code Exa 11.1 Recursive computation of Ej and Fj

```
// Recursive computation of Ej and Fj
// 11.1

function [Fj,dFj,Ej,dEj] = recursion(A,dA,C,dC,j)
Fo = C; dFo = dC;
Eo = 1; dEo = 0;
A_z = A(2:dA+1); dA_z = dA-1;
zi = 1; dzi = 0;
for i = 1:j-1
    if (dFo == 0)
        Fn1 = 0;
    else
        Fn1 = Fo(2:(dFo+1));
    end
    dFn1 = max(dFo-1,0);
    Fn2 = -Fo(1)*A_z;  dFn2 = dA-1;
    [Fn,dFn] = poladd(Fn1,dFn1,Fn2,dFn2);
    zi = convol(zi,[0,1]); dzi = dzi + 1;
    En2 = Fn(1)*zi; dEn2 = dzi;
    [En,dEn] = poladd(Eo,dEo,En2,dEn2);
    Eo = En; Fo = Fn;
```

```
    dEo = dEn; dFo = dFn;
end
if (dFo == 0)
    Fn1 = 0;
else
Fn1 = Fo(2:(dFo+1));
end;
dFn1 = max(dFo-1,0);
Fn2 = -Fo(1)*A_z;  dFn2 = dA-1;
[Fn,dFn] = poladd(Fn1,dFn1,Fn2,dFn2);
Fj = Fn; dFj = dFn;
Ej = Eo; dEj = dEo;
endfunction;
```

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

```
// Recursive computation of Ej and Fj for the system
    presented in Example 11.2 on page 408.
// 11.2

exec('poladd.sci',-1);
exec('polsize.sci',-1);
exec('recursion.sci',-1);

C = [1 0.5]; dC = 1;
A = [1 -0.6 -0.16]; dA = 2;
j = 2;
[Fj,dFj,Ej,dEj] = recursion(A,dA,C,dC,j)
```

Scilab code Exa 11.3 Solution of Aryabhatta identity

```
// Solution of Aryabhatta's identity Eq. 11.8, as
   discussed in Example 11.3 on page 409.
// 11.3

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);

C = [1 0.5]; dC = 1; j=2;
A = [1 -0.6 -0.16]; dA = 2;
zj = zeros(1,j+1); zj(j+1) = 1;
[Fj,dFj,Ej,dEj] = xdync(zj,j,A,dA,C,dC)
```

Scilab code Exa 11.4 1st control problem by MacGregor

```
// MacGregor's first control problem, discussed in
   Example 11.4 on page 213.
// 11.4

exec('mv.sci',-1);
exec('cl.sci',-1);
exec('cosfil_ip.sci',-1);
exec('zpowk.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('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
```

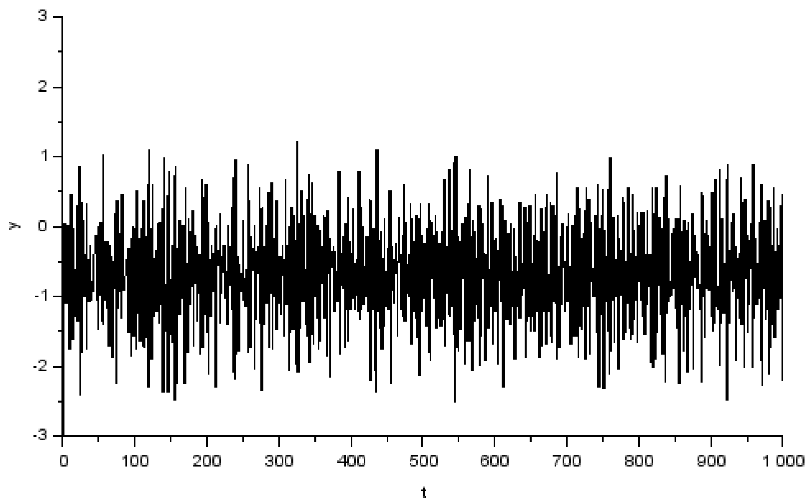

Figure 11.1: 1st control problem by MacGregor

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

Scilab code Exa 11.5 Minimum variance control law design

```
// Minimum variance control law design, given by Eq.
    11.40 on page 413.
// 11.5

// function [S,dS,R,dR] = mv(A,dA,B,dB,C,dC,k,int)
// implements the minimum variance controller
// if int>=1, integrated noise is assumed; otherwise
   ,
```

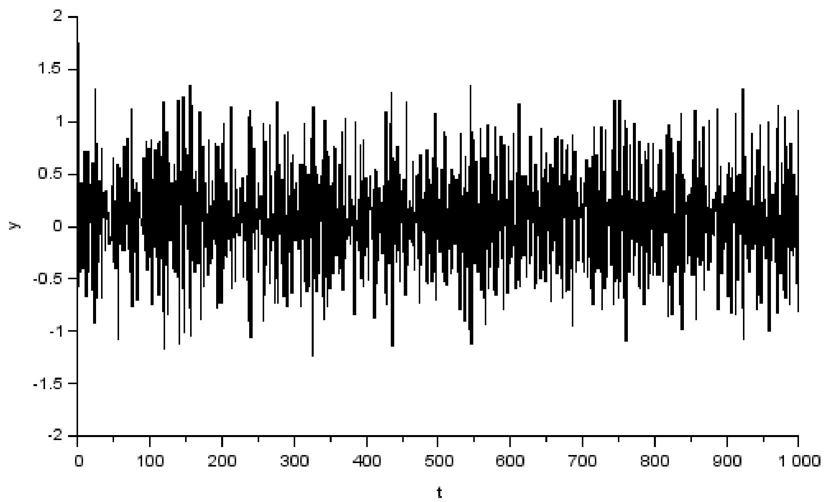

Figure 11.2: 1st control problem by MacGregor

```
// it is not integrated noise

function [S,dS,R,dR] = mv(A,dA,B,dB,C,dC,k,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);
S = Fk; dS = dFk; R = Gk; dR = dGk;
endfunction;
```

Scilab code Exa 11.6 Calculation of closed loop transfer functions

```
// Calculation of closed loop transfer functions
// 11.6

// function [Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] = ...
```

```
//      cl(A,dA,B,dB,C,dC,k,S,dS,R,dR,int)
// int>=1 means integrated noise and control law:
// delta u = - (S/R)y
// Evaluates the closed loop transfer function and
// variances of input and output

function [Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] =
    ...
    cl(A,dA,B,dB,C,dC,k,S,dS,R,dR,int1)
[zk,dzk] = zpowk(k);

[BS,dBS] = polmul(B,dB,S,dS);
[zBS,dzBS] = polmul(zk,dzk,BS,dBS);
[RA,dRA] = polmul(R,dR,A,dA);
if int1>=1, [RA,dRA] = polmul(RA,dRA,[1 -1],1); end

[D,dD] = poladd(RA,dRA,zBS,dzBS);

[Ny,dNy] = polmul(C,dC,R,dR);
[Nu,dNu] = polmul(C,dC,S,dS);

[Nu,dNu,Du,dDu,uvar] = tfvar(Nu,dNu,D,dD);
[Ny,dNy,Dy,dDy,yvar] = tfvar(Ny,dNy,D,dD);

endfunction;
```

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

```
// Cancellation of common factors and determination
    of covariance
// 11.7

// function [N,dN,D,dD,yvar] = tfvar(N,dN,D,dD)
// N and D polynomials in z^{-1} form; discrete case
```

```

function [N,dN,D,dD,yvar] = tfvar(N,dN,D,dD)

[N,dN,D,dD] = l2r(N,dN,D,dD);
N = N/D(1); D = D/D(1);
LN = length(N); LD = length(D);
D1 = D;
if LD<LN, D1 = [D zeros(1,LN-LD)]; dD1 = dD+LN-LD;
    end
H = tf(N,D1,1);//TS=1 (sampling time) has been taken
     constant in tfvar
yvar = covar_m(H,1);
endfunction;
```

Scilab code Exa 11.8 Computing sum of squares

```
// Computing sum of squares, as presented in Example
     11.5 on page 415.
// 11.8

exec('tf.sci',-1);
exec('covar_m.sci',-1);

Y = tf([1 0],[1 -0.9],-1);
covar_m(Y,1)
```

Scilab code Exa 11.9 Minimum variance control for nonminimum phase systems

```
// Minimum variance control for nonminimum phase
    systems
// 11.9

```

```
// function [Sc,dSc,Rc,dRc] = mv_mv(A,dA,B,dB,C,dC,k,int)
// implements the minimum variance controller
// if int>=1, integrated noise is assumed; otherwise,
// it is not integrated noise

function [Sc,dSc,Rc,dRc] = mv_nm(A,dA,B,dB,C,dC,k,int1)
if int1>=1, [A,dA] = polmul([1 -1],1,A,dA); end
[zk,dzk] = zpowk(k);
[Bzk,dBzk] = polmul(B,dB,zk,dzk);
[Bg,Bb] = polsplit3(B); Bbr = flip(Bb);
RHS = convol(C,convol(Bg,Bbr)); dRHS = length(RHS)-1;
[Sc,dSc,Rc,dRc] = xdync(Bzk,dBzk,A,dA,RHS,dRHS);
endfunction;
```

Scilab code Exa 11.10 Minimum variance control for nonminimum phase example

```
// Minimum variance control for nonminimum phase example of Example 11.6 on page 416.
// 11.10

exec('mv_nm.sci',-1);
exec('cl.sci',-1);
exec('zpowk.sci',-1);
exec('polmul.sci',-1);
exec('polsize.sci',-1);
exec('polsplit3.sci',-1);
exec('flip.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('poladd.sci',-1);
exec('tfvar.sci',-1);
exec('l2r.sci',-1);
exec('transp.sci',-1);
exec('tf.sci',-1);
exec('covar_m.sci',-1);

A = convol([1 -1],[1 -0.7]); dA = 2;
B = [0.9 1]; dB = 1; k = 1;
C = [1 -0.7]; dC = 1; int1 = 0;
[Sc,dSc,Rc,dRc] = mv_nm(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);
```

Scilab code Exa 11.11 Minimum variance control of viscosity control problem

```
// Minimum variance control of viscosity control problem
// 11.11

// Viscosity control problem of MacGregor

exec('mv_nm.sci',-1);
exec('polmul.sci',-1);
exec('polsize.sci',-1);
exec('zpowk.sci',-1);
```

```
exec('polsplit3.sci',-1);
exec('flip.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('cl.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);

A = [1 -0.44]; dA = 1; B = [0.51 1.21]; dB = 1;
C = [1 -0.44]; dC = 1; k = 1; int1 = 1;
[Sc,dSc,Rc,dRc] = mv_nm(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);
```

Scilab code Exa 11.12 General minimum variance controller design

```
// General minimum variance controller design, as given by Eq. 11.66 on page 421 and Eq. 11.70 on page 422.
// 11.12

// 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);
```

```
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('zpowk.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);

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;
rho = 1;
[Sc,dSc,Rc,dRc] = gmv(A,dA,B,dB,C,dC,k,rho,int1);
[Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] = ...
        cl(A,dA,B,dB,C,dC,k,Sc,dSc,Rc,dRc,int1);
```

Scilab code Exa 11.14 GMVC design of viscosity problem

```
// GMVC design of viscosity problem, as described in
    Example 11.10 on page 423.
// 11.14

// MacGregor's Viscosity control problem by gmv

exec('gmv.sci',-1);
exec('cl.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('poladd.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 -0.44]; dA = 1; B = [0.51 1.21]; dB = 1;
C = [1 -0.44]; dC = 1; k = 1; int1 = 1;
rho = 1;
[Sc,dSc,R1,dR1] = gmv(A,dA,B,dB,C,dC,k,rho,int1);
[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 11.15 PID tuning through GMVC law

```
// PID tuning through GMVC law, as discussed in
   Example 11.11.
// 11.15

exec('gmvc_pid.sci',-1);
exec('zpowk.sci',-1);
exec('ch_pol.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('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)
```

```

function Y = filtval(P,z)
N = length(P)-1;
Q = polyno(P,'x');
Y = horner(Q,z)/z^N;
endfunction;
```

Scilab code Exa 11.17 PID tuning through GMVC law

```
// PID tuning through GMVC law
// 11.17

// function [Kc,tau_i,tau_d,L] = gmvc_pid(A,B,k,T,Ts
   )
// Determines p,i,d tuning parameters using GMVC
// Plant model: Integrated white noise
// A, B in discrete time form

function [Kc,tau_i,tau_d,L] = gmvc_pid(A,B,k,T,Ts)

dA = length(A)-1; dB = length(B)-1;
dT = length(T)-1;
if dA > 2,
   disp('degree of A cannot be more than 2')
   exit
elseif dB > 1,
   disp('degree of B cannot be more than 1')
   exit
elseif dT > 2,
   disp('degree of T cannot be more than 2')
   exit
end
delta = [1 -1]; ddelta = 1;

[Adelta,dAdelta] = polmul(A,dA,delta,ddelta);
```

```

[Q,dQ,P,dP] = ...
xdync(Adelta,dAdelta,B,dB,T,dT);
PAdelta = P(1)*Adelta;

[zk,dzk] = zpowk(k);
[E,degE,F,degF] = ...
xdync(PAdelta,dAdelta,zk,dzk,P,dP);
nu = P(1)*E(1)*B(1);
Kc = -1/nu*(F(2)+2*F(3));
tau_i = -(F(2)+2*F(3))/(F(1)+F(2)+F(3))*Ts;
tau_d = -F(3)/(F(2)+2*F(3))*Ts;
L(1) = 1+Ts/tau_i+tau_d/Ts;
L(2) = -(1+2*tau_d/Ts);
L(3) = tau_d/Ts;
L = Kc * L';
endfunction;
```

Chapter 12

Model Predictive Control

Scilab code Exa 12.1 Model derivation for GPC design

```
// Model derivation for GPC design in Example 12.1
   on page 439.
// 12.1

exec('xdync.sci',-1);
exec('polmul.sci',-1);
exec('flip.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);

// Camacho and Bordon's GPC example; model formation

```

```
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
```

```
K = inv(G'*G+rho*eye(Nu,Nu))*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 12.3 GPC design for the problem discussed on page 441

```
// GPC design for the problem discussed in Example
   12.2 on page 441.
// 12.3

exec('gpc_bas.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('flip.sci',-1);
exec('filtval.sci',-1);

// Camacho and Bordon's GPC example; Control law
A=[1 -0.8]; dA=1; B=[0.4 0.6]; dB=1; N=3; k=1; rho
   =0.8;
[K,KH1,KH2,Tc,dTc,Sc,dSc,R1,dR1] = ...
```

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

Scilab code Exa 12.4 GPC design

```
// GPC design for the problem discussed in Example
   12.3.
// 12.4

exec('gpc_N.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('flip.sci',-1);

A=[1 -0.8]; dA=1; B=[0.4 0.6]; dB=1;
rho = 0.8; k = 1;
N1 = 0; N2 = 3; Nu = 2;

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

Scilab code Exa 12.5 Calculates the GPC law

```
// Calculates the GPC law given by Eq. 12.36 on page 446.
// 12.5

function [K,KH1,KH2,Tc,dTc,Sc,dSc,R1,dR1] = ...
gpc_N(A,dA,B,dB,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
G = zeros(N2-N1+1,Nu+1);
H1 = zeros(N2-N1+1,k-1+dB); H2 = zeros(N2-N1+1,dA+1);
for j = k+N1:k+N2
    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);
    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:k-1+dB) = Gj(j-k+2:j+dB);
    H2(j-(k+N1-1),1:dA+1) = Fj;
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 12.6 Calculates the GPC law

```
// Calculates the GPC law given by Eq. 12.36 on page 446.
// 12.6

function [K,KH1,KH2,Tc,dTc,Sc,dSc,R1,dR1] = ...
gpc_col(A,dA,B,dB,C,dC,N,k,rho)
D=[1 -1]; dD = 0; AD=convol(A,D); dAD=dA+1; zj=1; dzj=0;
Nu = N+1; G=zeros(Nu,Nu); H1=zeros(Nu,2*k+N-2+dB);
H2 = zeros(Nu,k+N+dA);
for j = 1:Nu,
    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);
    j,Fj,Ej,Mj,Nj,Gj,Pj
    G(j,1:j) = flip(Gj(1:j));
    H1(j,1:dGj-j+1) = Gj(j+1:dGj+1);
    H2(j,1:dPj+1) = Pj;
end
K = inv(G'*G+rho*eye(Nu,Nu))*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 12.7 GPC design for viscosity control

```
// GPC design for viscosity control in Example 12.4
   on page 446.
// 12.7

exec('gpc_col.sci',-1);
exec('poladd.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('flip.sci',-1);

// GPC control of viscosity problem
A=[1 -0.44]; dA=1; B=[0.51 1.21]; dB=1; N=2; k=1;
C = [1 -0.44]; dC = 1; rho = 1;

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

```
// PID controller , tuned with GPC, as discussed in
   Example 12.5 on page 452.
// 12.10

exec('gpc_pid.sci',-1);
exec('zpowk.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);

A = [1 -1.95 0.935];
B=-0.015;
C=1;
degA=2;
degB=0;
degC=0;
N1=1;
N2=5;
Nu=2;
gamm=0.05;
gamma_y=1;
lambda=0.02;

[Kp,Ki,Kd] = ...
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

```
// Predictive PID, tuned with GPC, as explained in Sec. 12.2.3.
// 12.11

function [Kp,Ki,Kd] = ...
gpc_pid(A,dA,B,dB,C,dC,N1,N2,Nu,lambda,gamm,gamma_y)
Adelta=convol(A,[1 -1]); G=[];
for i=N1:N2
    zi=zpowk(i);
    [E,dE,F,dF]=xdync(Adelta,dA+1,zi,i,C,dC);
    [Gtilda,dGtilda,Gbar,dGbar] = ...
      xdync(C,dC,zi,i,E*B,dE+dB);
    for j = 1:i, Gtilda1(j)=Gtilda(i+1-j); end
    Gtilda2 = Gtilda1.'; // Added because Scilab forms a column vecor
    // while Matlab forms a row vector, by default
        if i<=Nu-1
       G=[G;[Gtilda2,zeros(1,Nu-i)]];
    else
       G=[G;Gtilda2(1:Nu)];
    end
end
es=sum(C)/sum(A); gs=sum(B)/sum(A); F_s=es*A; G_s=[];
for i=1:Nu
    if ((Nu - i) == 0)
    row=gs*ones(1,i);
    else
    row=gs*ones(1,i); row=[row,zeros(Nu-i,Nu-i)];
    end;
    G_s=[G_s;row];
end
```

```
lambda_mat=lambda*(diag(ones(1,Nu)));
gamma_mat=gamm*(diag(ones(1,Nu)));
gamma_y_mat=gamma_y*(diag(ones(1,N2-N1+1)));
mat1=inv(G'*gamma_y_mat*G+lambda_mat+G_s'*gamma_mat*
    G_s);
mat2=mat1*(G'*gamma_y_mat);
mat2_s=mat1*(G_s'*gamma_mat);
h_s=sum(mat2_s(1,:)); h=mat2(1,:);
T=C; R=C*(sum(h(:))+h_s); S=0;
for i=N1:N2
    zi=zpowk(i);
    [E,dE,F,dF]=xdync(Adelta,dA+1,zi,i,C,dC);
    [Gtilda,dGtilda,Gbar,dGbar]=...
      xdync(C,dC,zi,i,E*B,dE+dB);
    S=S+F*h(i);
end
S=S+F_s*h_s;
if length(A)==3
   Kp=S(1)-R-S(3); Ki=R; Kd=S(3);
else
   Kp=S(1)-R; Ki=R; Kd=0;
end

endfunction;
```

Chapter 13

Linear Quadratic Gaussian Control

Scilab code Exa 13.1 Spectral factorization

```
// Spectral factorization , as discussed in Example
   13.3 on page 467.
// 13.1

exec('spec1.sci',-1);
exec('flip.sci',-1);
exec('polmul.sci',-1);
exec('polsize.sci',-1);
exec('poladd.sci',-1);

A = convol([-0.5 1],[-0.9 1]); dA = 2;
B = 0.5*[-0.9 1]; dB = 1; rho = 1;
[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
   )
```

```
// Implements the spectral factorization for use
   with LQG control
// design method of Ahlen and Sternard

function [r,b,dAFW] = ...
   specfac(A,degA,B,degB,rho,V,degV,W,degW,F,degF)
AFW = convol(A,convol(W,F));
dAFW = degA + degF + degW;
AFWWFA = rho * convol(AFW,flip(AFW));
BV = convol(B,V);
dBV = degB + degV;
BVVB = convol(BV,flip(BV));
diff1 = dAFW - dBV;
dBVVB = 2*dBV;
for i = 1:diff1
    [BVVB,dBVVB] = polmul(BVVB,dBVVB,[0 1],1);
end
[rbb,drbb] = poladd(AFWWFA,2*dAFW,BVVB,dBVVB);
Rbb = polyno(rbb,'z');
rts = roots(Rbb);
rtsin = rts(dAFW+1:2*dAFW);
b = 1;
for i = 1:dAFW,
    b = convol(b,[1 -rtsin(i)]);
end
b = real(b);
br = flip(b);
bbr = convol(b,br);
r = rbb(1) / bbr(1);
endfunction;
```

Scilab code Exa 13.4 LQG control design by polynomial method

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

```
// 13.4

// LQG controller design by method of Ahlen and
   Sternad
// function [R,degR,S,degS] = ...
// lqg(A,degA,B,degB,C,degC,k,rho,V,degV,W,degW,F,
   degF)

function [R,degR,S,degS] = ...
lqg1(A,degA,B,degB,C,degC,k,rho,V,degV,W,degW,F,degF
   )

[r,b,degb] = ...
specfac(A,degA,B,degB,rho,V,degV,W,degW,F,degF);

WFA = flip(convol(A,convol(F,W)));
dWFA = degW + degF + degA;

[rhs1,drhs1] = polmul(W,degW,WFA,dWFA);
[rhs1,drhs1] = polmul(rhs1,drhs1,C,degC);
rhs1 = rho * rhs1;
rhs2 = convol(C,convol(V,flip(convol(B,V))));
drhs2 = degC + 2*degV + degB;
for i = 1:degb-degB-degV,
  rhs2 = convol(rhs2,[0,1]);
end
drhs2 = drhs2 + degb-degB-degV;
C1 = zeros(1,2);

[C1,degC1] = putin(C1,0,rhs1,drhs1,1,1);
[C1,degC1] = putin(C1,degC1,rhs2,drhs2,1,2);
rbf = r * flip(b);
D1 = zeros(2,2);
[D1,degD1] = putin(D1,0,rbf,degb,1,1);
for i = 1:k,
    rbf = convol(rbf,[0 1]);
end
[D1,degD1] = putin(D1,degD1,rbf,degb+k,2,2);
```

```
N = zeros(1,2);
[N,degN] = putin(N,0,-B,degB,1,1);
[AF,dAF] = polmul(A,degA,F,degF);
[N,degN] = putin(N,degN,AF,dAF,1,2);

[Y,degY,X,degX] = xdync(N,degN,D1,degD1,C1,degC1);

[R,degR] = ext(X,degX,1,1);
[S,degS] = ext(X,degX,1,2);
X = flip(Y);

endfunction;
```

Scilab code Exa 13.5 LQG design

```
// LQG design for the problem discussed in Example
    13.4 on page 472.
// 13.5

// MacGregor's first control problem

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);

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

```
// Simplified LQG control design, obtained by the
   solution of Eq. 13.53 on page 476.
// 13.7

// LQG controller simple design by method of Ahlen
   and Sternad
// function [R1,dR1,S,dS] = ...
// lqg_simple(A,dA,B,dB,C,dC,k,rho,V,dV,W,dW,F,dF)

function [R1,dR1,S,dS] = ...
lqg_simple(A,dA,B,dB,C,dC,k,rho,V,dV,W,dW,F,dF)
[r,b,db] = specfac(A,dA,B,dB,rho,V,dV,W,dW,F,dF);
[D,dD] = polmul(A,dA,F,dF);
[zk,dzk] = zpowk(k);
[N,dN] = polmul(zk,dzk,B,dB);
[RHS,dRHS] = polmul(C,dC,b,db);
[S,dS,R1,dR1] = xdync(N,dN,D,dD,RHS,dRHS);
endfunction;
```

Scilab code Exa 13.8 LQG control design

```
// LQG control design for the problem discussed in
   Example 13.6 on page 474.
// 13.8

// Solves Example 3.1 of Ahlen and Sternad in Hunt's
   book
exec('lqg1.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('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);

A = [1 -0.9]; dA = 1; B = [0.1 0.08]; dB = 1;
k = 2; rho = 0.1; C = 1; dC = 0;
V = 1; dV = 0; F = 1; dF = 0; W = [1 -1]; dW = 1;
[R1,dR1,Sc,dSc] = ...
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

```
// Performance curve for LQG control design of
   viscosity problem
// 13.9

exec('lqg1.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('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

```
// Performance curve for GMVC design of MacGregor's
   first control problem
// 13.10

exec('gmv.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('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 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;
u_gmv = []; y_gmv = []; uy_gmv = [];
```

```

for rho = 0:0.1:10,
    [S,dS,R,dR] = gmv(A,dA,B,dB,C,dC,k,rho,int1);
    [Nu,dNu,Du,dDu,Ny,dNy,Dy,dDy,yvar,uvar] = ...
        cl(A,dA,B,dB,C,dC,k,S,dS,R,dR,int1);
    u_gmv = [u_gmv uvar]; y_gmv = [y_gmv yvar];
    uy_gmv = [uy_gmv; [rho uvar yvar]];
end
plot(u_gmv,y_gmv,'b')
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

```
// Pole placement controller for inverted pendulum,
   discussed in Example 14.1 on page 490. 2.1 should
    be executed before starting this code
// 14.1

exec('pol2cart.sci',-1);

C = eye(4,4);
D = zeros(4,1);
Ts = 0.01;
G = syslin('c',A,B,C,D);
H = dscr(G,Ts);
[a,b,c,d] = H(2:5);
rise = 5; epsilon = 0.1;
N = rise/Ts;
omega = %pi/2/N;
r = epsilon^(omega/%pi);
r1 = r; r2 = 0.9*r;
[x1,y1] = pol2cart(omega,r1);
```

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

Scilab code Exa 14.2 Compensator calculation

```
// Compensator calculation for Example 14.6 on page
    507.
// 14.2

exec('polyno.sci',-1);
exec('polmul.sci',-1);
exec('polsize.sci',-1);

A = [1 2; 0 3]; c = [1 0];
p = roots(polyno([1 -0.5 0.5],'z'));
b = [0; 1];
K = ppol(A,b,p);

p1=0.1+0.1*%i; p2=0.1-0.1*%i;
phi = real(convol([1 -p1],[1 -p2]));
Obs = [c; c*A];
alphae = A^2-0.2*A+0.02*eye(2,2);
Lp = alphae*inv(Obs)*[0; 1];
Lp = ppol([1 0;2 3], ...
[1; 0],[0.1+0.1*%i 0.1-0.1*%i]);
Lp = Lp';

C = [1 0 0.5 2;0 1 -4.71 2.8];
dC = 1;

```

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

Scilab code Exa 14.3 Kalman filter example of estimating a constant

```
// Kalman filter example of estimating a constant, discussed in Example 14.7.
// 14.3

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

x = 5; xhat = 2; P = 1; xvec = x;
xhat_vec = xhat; Pvec = P; yvec = x;
for i = 1:200,
    xline = xhat; M = P;
    [xhat,P,y] = kal_ex(x,xline,M);
    xvec = [xvec;x];
    xhat_vec = [xhat_vec;xhat];
    Pvec = [Pvec;P]; yvec = [yvec;y];
end
n = 1:201;
plot(Pvec);
xtitle('','n');
halt();
clf();
plot(n,xhat_vec',n,yvec',n,xvec');
xtitle('','n');
```

Scilab code Exa 14.4 Kalman filter example of estimating a constant

```
// Kalman filter example of estimating a constant
// 14.4

```

```
function [xhat,P,y] = kal_ex(x,xline,M)
y = x + rand();
Q = 0; R = 1;
xhat_ = xline;
P_ = M + Q;
K = P_/(P_+R);
P = (1-K)*P_;
xhat = xhat_ + K*(y-xhat_);
endfunction;
```