

Scilab Textbook Companion for  
Process Dynamics And Controls  
by D. E. Seborg, T. F. Edgar, D. A.  
Mellichamp And F. J. Doyle III<sup>1</sup>

Created by  
Dhruv Gupta  
Process Control CL 417 IITB  
Chemical Engineering  
IIT-Bombay  
College Teacher  
Na  
Cross-Checked by  
Mukul R. Kulkarni

July 31, 2019

<sup>1</sup>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:** Process Dynamics And Controls

**Author:** D. E. Seborg, T. F. Edgar, D. A. Mellichamp And F. J. Doyle III

**Publisher:** John Wiley And Sons (Asia) Pvt. Ltd.

**Edition:** 3

**Year:** 2013

**ISBN:** 978-81-265-4126-3

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	4
2 Theoretical Models of Chemical Processes	5
6 Development of Empirical Models from Process Data	15
8 Control System Instrumentation	29
9 Process Safety and Process Control	35
10 Dynamic behavior	38
11 PID Controller Design Tuning and Troubleshooting	51
12 Control Strategies at the Process Unit Level	75
13 Frequency response analysis and control system design	76
14 Feedforward and Ratio Control	93
15 Enhanced Single Loop Control Strategies	100
16 Multiloop and Multivariable Control	104
17 Digital Sampling Filtering and Control	117

<b>19 Real Time Optimization</b>	<b>143</b>
<b>20 Model Predictive Control</b>	<b>148</b>
<b>21 Process Monitoring</b>	<b>172</b>
<b>22 Biosystems Control Design</b>	<b>187</b>

# List of Scilab Codes

Exa 2.1	Stirred tank blending process . . . . .	5
Exa 2.2	Degrees of freedom 1 . . . . .	8
Exa 2.3	Degrees of freedom 2 . . . . .	10
Exa 2.4	Electrically heated stirred tank process . . . . .	10
Exa 2.5	Nonlinear dynamic behavior of CSTR . . . . .	13
Exa 6.1	Gas turbine generator . . . . .	15
Exa 6.2	Continuous stirred tank reactor . . . . .	17
Exa 6.3	Off gas CO <sub>2</sub> concentration response . . . . .	18
Exa 6.5	Estimation of model parameters . . . . .	22
Exa 6.6	Step test of distillation column . . . . .	25
Exa 8.2	Pump and heat exchanger system . . . . .	29
Exa 9.1	Liquid surge system . . . . .	35
Exa 9.2	Abnormal event in distillation column . . . . .	35
Exa 9.3	Reliability of flow control loop . . . . .	37
Exa 10.2	Set point response of level control system . . . . .	38
Exa 10.3	Disturbance response of level control system . . . . .	40
Exa 10.4	Stability of feedback control system . . . . .	43
Exa 10.10	Routh Array 1 . . . . .	45
Exa 10.11	Routh Array 2 . . . . .	46
Exa 10.12	Direct substitution to find stability . . . . .	47
Exa 10.13	Root Locus . . . . .	47
Exa 10.14	Transient response from root locus . . . . .	49
Exa 11.1	Direct synthesis for PID . . . . .	51
Exa 11.3	PI and PID control of liquid storage tank . . . . .	54
Exa 11.4	IMC for lag dominant model . . . . .	59
Exa 11.5	PI controller IMC ITAE . . . . .	66
Exa 11.6	Controller with two degrees of freedom . . . . .	66
Exa 11.7	Continuous cycling method . . . . .	69

Exa 11.8	Reaction curve method . . . . .	72
Exa 12.1	Degrees of freedom . . . . .	75
Exa 13.3	Bode Plot . . . . .	76
Exa 13.4	Bode . . . . .	79
Exa 13.5	PI control of overdamped second order process	81
Exa 13.6	Bode plot . . . . .	83
Exa 13.7	Bode Plot . . . . .	85
Exa 13.8	Maximum permissible time delay for stability	88
Exa 14.1	Ratio control . . . . .	93
Exa 14.5	Feedforward control in blending process . .	93
Exa 15.1	Stability limits for proportional cascade controller . . . . .	100
Exa 15.2	Set point response for second order transfer function . . . . .	101
Exa 16.1	Pilot scale distillation column . . . . .	104
Exa 16.6	Sensitivity of steady state gain matrix . . .	113
Exa 16.7	Preferred multiloop control strategy . . . . .	114
Exa 17.1	Performance of alternative filters . . . . .	117
Exa 17.2	Response of first order difference equation .	122
Exa 17.3	Recursive relation with inputs . . . . .	125
Exa 17.5	Digital control of pressure in a tank . . . . .	127
Exa 17.6	Dahlin controller . . . . .	129
Exa 17.7	Non ringing Dahlin controller . . . . .	134
Exa 19.2	Nitration of Decane . . . . .	143
Exa 19.3	Refinery blending and production . . . . .	144
Exa 19.4	Fuel cost in boiler house . . . . .	145
Exa 20.1	Step response coefficients . . . . .	148
Exa 20.3	J step ahead prediction . . . . .	150
Exa 20.4	Output feedback and bias correction . . . . .	153
Exa 20.5	Comparison of MPCs and PID . . . . .	156
Exa 21.2	Semiconductor processing control charts . .	172
Exa 21.3	Shewart CUSUM EWMA comparison . . . . .	176
Exa 21.4	Process Capability Indices . . . . .	179
Exa 21.5	Effluent Stream from wastewater treatment	179
Exa 21.6	Hotellings T square statistic . . . . .	185
Exa 22.1	Fermentor . . . . .	187
Exa 22.2	Granulation process control . . . . .	190
Exa 22.3	Type 1 Diabetes . . . . .	198

Exa 22.4 Influence of drugs . . . . . 200



## Chapter 2

# Theoretical Models of Chemical Processes

Scilab code Exa 2.1 Stirred tank blending process

```
1 clear
2 clc
3
4 //Example 2.1
5 disp('Example 2.1')
6
7 w1bar=500;
8 w2bar=200;
9 x1bar=0.4;
10 x2bar=0.75;
11 wbar=w1bar+w2bar;
12 t=0:0.1:25; //Time scale for plotting of graphs
13
14 //(a)
15 xbar=(w1bar*x1bar+w2bar*x2bar)/wbar;
16 printf('\n (a) The steady state concentration is %f
        \n',xbar)
```

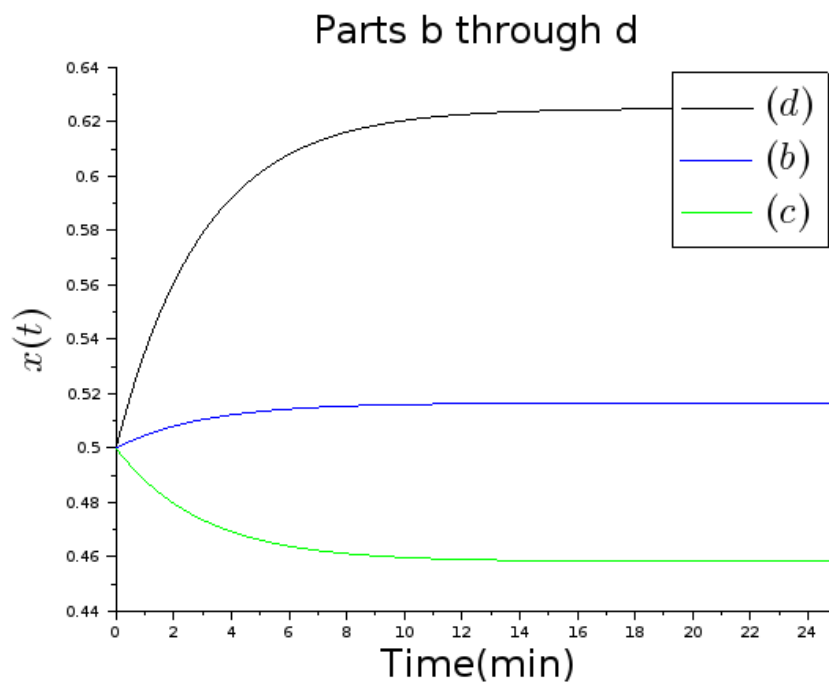


Figure 2.1: Stirred tank blending process

```

17
18 //(b)
19 w1bar=400; //flow rate changes, rest remains same
20 wbar=w1bar+w2bar;
21 tau=3;
22 x0=0.5;
23 Cstarb=(w1bar*x1bar+w2bar*x2bar)/wbar; //C*
    variable
24 printf('\n (b) The value of C* is %f',Cstarb)
25 printf('\n x(t)=0.5exp(-t/3)+%f(1-exp(-t/3)) \n',
    Cstarb);
26 xtd=0.5*exp(-t/3)+Cstarb*(1-exp(-t/3));
27
28 xtb=0.5*exp(-t/3)+Cstarb*(1-exp(-t/3)); //x(t) for
    part (b)
29
30 //(c)
31 w1bar=500;w2bar=100; //flow rate changes, rest
    remains same
32 wbar=w1bar+w2bar;
33 tau=3;
34 x0=0.5;
35 Cstarc=(w1bar*x1bar+w2bar*x2bar)/wbar; //C*
    variable
36 printf('\n (c) The value of C* is %f',Cstarc)
37 printf('\n x(t)=0.5exp(-t/3)+%f(1-exp(-t/3)) \n',
    Cstarc);
38 xtc=0.5*exp(-t/3)+Cstarc*(1-exp(-t/3));
39
40 //(d)
41 w1bar=500;w2bar=100;x1bar=0.6;x2bar=0.75; //flow
    rate changes, rest remains same
42 wbar=w1bar+w2bar;
43 tau=3;
44 x0=0.5;
45 Cstard=(w1bar*x1bar+w2bar*x2bar)/wbar; //C*
    variable
46 printf('\n (d) The value of C* is %f',Cstard)

```

```

47 printf( '\n x(t)=0.5exp(-t/3)+%f(1-exp(-t/3)) \n ',
          Cstard);
48 xtd=0.5*exp(-t/3)+Cstard*(1-exp(-t/3));
49
50 plot2d(t,[xtd',xtb',xtdc'])
51 xtitle('Parts b through d','Time(min)','$x(t)$');
52 a=legend("$ (d)$","$(b)$","$(c)$",position=1);
53 a.font_size=5;
54 a=get("current_axes");b=a.title;b.font_size=5;c=a.
          x_label;c.font_size=5;
55 c=a.y_label;c.font_size=5;
56
57 //(e)
58 xNb=(xtb-x0)/(Cstarb-x0); //Normalized response for
          part b
59 xNc=(xtd-x0)/(Cstarc-x0); //Normalized response for
          part c
60 xNd=(xtd-x0)/(Cstard-x0); //Normalized response for
          part d
61
62 scf() //Creates new window for plotting
63 plot2d(t,[xNd',xNb',xNc'],style=[1 1 1])
64 //Style sets the color, -ve values means discrete
          plotting, +ve means color
65 xtitle('Part e','Time(min)','Normalized response');
66 a=legend("$ (e)$",position=1);
67 a.font_size=5;
68 a=get("current_axes");b=a.title;b.font_size=5;c=a.
          x_label;c.font_size=5;
69 c=a.y_label;c.font_size=5;

```

---

Scilab code Exa 2.2 Degrees of freedom 1

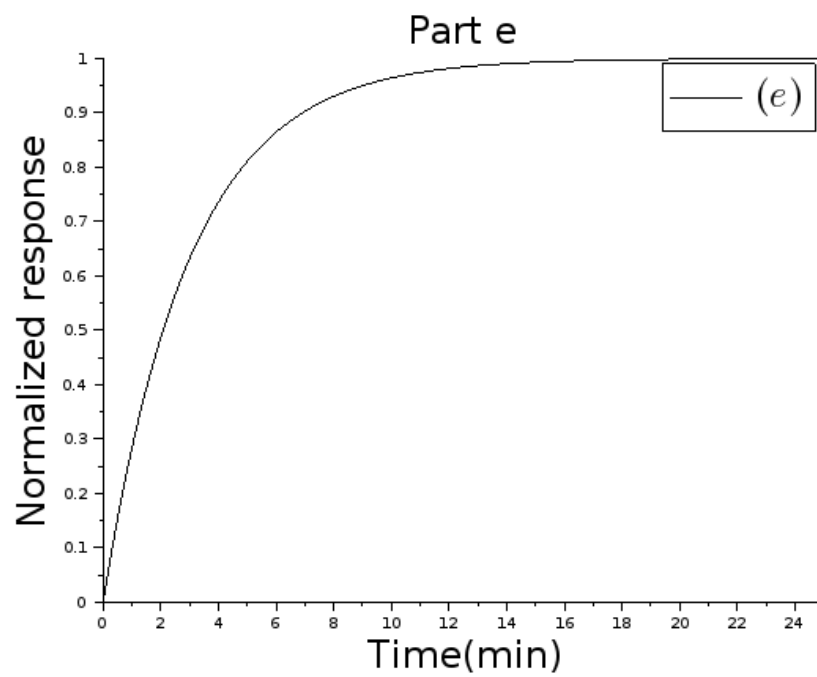


Figure 2.2: Stirred tank blending process

```
1 clear
2 clc
3
4 //Example 2.2
5 disp('Example 2.2 ')
6
7 N_V=4;
8 N_E=1;
9 N_F=N_V-N_E;
10 printf('\n Degrees of freedom N_F= %i \n',N_F)
```

---

#### Scilab code Exa 2.3 Degrees of freedom 2

```
1 clear
2 clc
3
4 //Example 2.3
5 disp('Example 2.3 ')
6
7
8 N_V=7;
9 N_E=2;
10 N_F=N_V-N_E;
11 printf('\n Degrees of freedom N_F= %i \n',N_F)
```

---

#### Scilab code Exa 2.4 Electrically heated stirred tank process

```
1 clear
2 clc
3
4 //Example 2.4
```

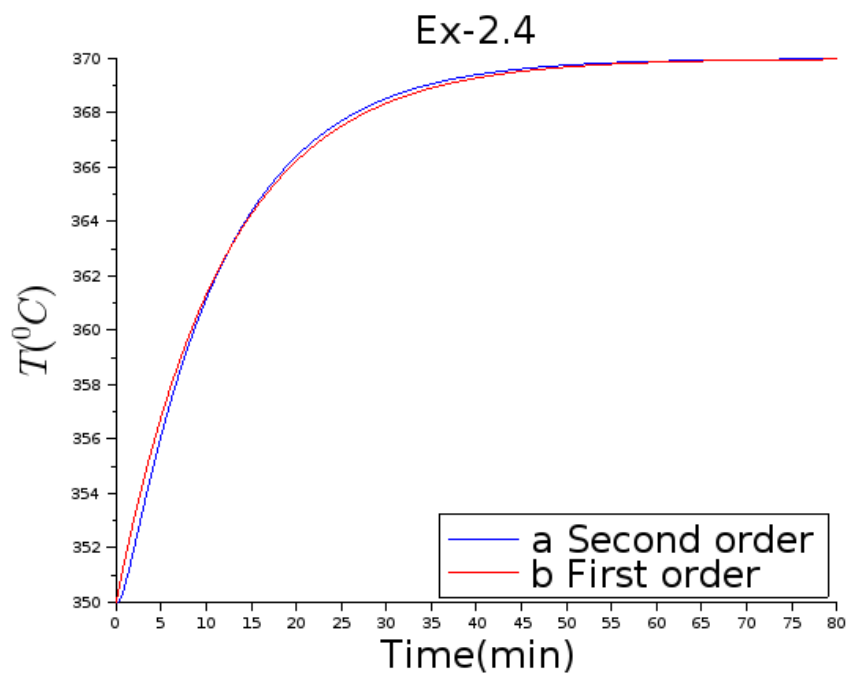


Figure 2.3: Electrically heated stirred tank process

```

5 disp('Example 2.4')
6
7 mprintf('\n Important Note: Errata for book: Values
      of the parameters \n...
8 meCe/heAe and meCe/wC should be 1 min each and not
      0.5 min %s \n', '')
9
10 Tibar=100; //deg C
11 Qbar=5000; //kcal/min
12 wc_inv=0.05; // 1/wc degC min/kcal
13
14 //(a)
15 Tbar=Tibar+wc_inv*Qbar;
16 mprintf('\n (a) Nominal steady state temperature= %i
      ', Tbar)
17 mprintf(' degree celsius %s \n', '')
18
19 //(b)
20 mprintf('\n Eqn 2-29 becomes 10 d2T/dt2 + 12 dT/dt +
      T = 370 with T(0)=350 %s \n', '')
21 t=0:0.1:80; //Time values
22 Tt_2=350+20*(1-1.089*exp(-t/11.099)+0.084*exp(-t
      /0.901)); //T(t) from order 2 equation
23
24 //(c)
25 mprintf('\n Eqn 2-29 becomes 12 dT/dt + T = 370 with
      T(0)=350 %s \n', '')
26 Tt_1=350+20*(1-exp(-t/12)); //T(t) from order 1
      equation
27
28
29 plot2d(t, [Tt_2', Tt_1'], [2 5], rect=[0 350 80 370])
30 xtitle('Ex-2.4', 'Time (min)', '$T(^{\circ}C)$');
31 a=legend("a Second order", "b First order", position
      =4);
32 a.font_size=5;
33 a=get("current_axes"); b=a.title; b.font_size=5; c=a.
      x_label; c.font_size=5;

```



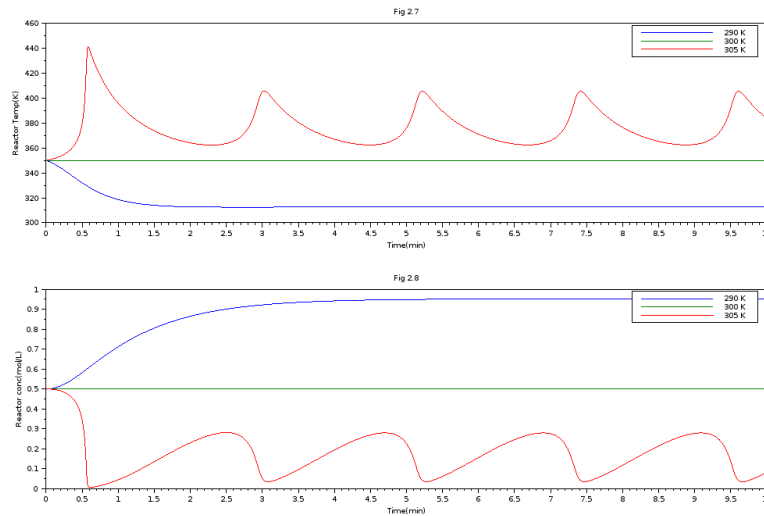


Figure 2.4: Nonlinear dynamic behavior of CSTR

34 `c=a.y_label;c.font_size=5;`

---

Scilab code Exa 2.5 Nonlinear dynamic behavior of CSTR

```

1
2 clear
3 clc
4
5 //Example 2.5
6 disp('Example 2.5')
7
8 function ydot=CSTR(t,y,Tc) //y is [Conc Temp]' Tc is
   coolant temp
9     q=100;ci=1;V=100;rho=1000;C=0.239;deltaHR=5E4;k0
       =7.2E10;UA=5E4;Er=8750;
10    Ti=350;
```

```

11     c=y(1);T=y(2);
12     k=k0*exp(-Er/T); //Er=E/R
13     ydot(1)=1/V*(q*(ci-c)-V*k*c); //ydot(1) is
        dc_dt
14     ydot(2)=1/(V*rho*C)*(q*rho*C*(Ti-T)+deltaHR*V*k*
        c+UA*(Tc-T)) //ydot(2) is dT_dt
15 endfunction
16
17 c0=0.5;T0=350;
18 y0=[c0 T0]';
19 t0=0;
20 t=0:0.01:10;
21 Tc=[290 305];
22 y1 = ode(y0,t0,t,list(CSTR,Tc(1)));
23 y2 = ode(y0,t0,t,list(CSTR,Tc(2)));
24 y3=[0.5 0;0 350]*ones(2,length(t))
25 //Temp plot
26 subplot(2,1,1);
27 plot(t,[y1(2,:) y3(2,:) y2(2,:)']);
28 xtitle(" Fig 2.7 ", "Time(min)", "Reactor Temp(K)");
29 legend("290 K", "300 K", "305 K")
30 //conc plot
31 subplot(2,1,2);
32 plot(t,[y1(1,:) y3(1,:) y2(1,:)']);
33 xtitle(" Fig 2.8 ", "Time(min)", "Reactor conc(mol/L)");
34 legend("290 K", "300 K", "305 K");

```

---

# Chapter 6

## Development of Empirical Models from Process Data

Scilab code Exa 6.1 Gas turbine generator

```
1 clear
2 clc
3
4 //Example 6.1
5 disp('Example 6.1 ')
6
7 //Fuel flow rate appended with ones for intercept in
  regression
8 fuel=[1 2.3 2.9 4 4.9 5.8 6.5 7.7 8.4 9];
9 X=[ones(1,10);fuel]';
10 Y=[2 4.4 5.4 7.5 9.1 10.8 12.3 14.3 15.8 16.8]'; //
  Power generated
11
12 Bhat=inv(X'*X)*X'*Y;
13
14 mprintf('\n Linear model \n B1_hat=%f \n B2_hat=%f',
  Bhat ')
15
16
```

```

17 //For better accuracy we can fit higher order model
18 X_new=[ones(1,10);fuel;fuel.^2]';
19 Bhat_new=inv(X_new'*X_new)*X_new'*Y;
20 mprintf('\n \n Quadratic model \n B1_hat=%f \n
    B2_hat=%f \n B3_hat=%f',Bhat_new')
21 Output_table=[fuel' Y X*Bhat X_new*Bhat_new];
22
23 //mprintf('\n Fuel          Power Generated          Linear
    Model          Quadratic Model %f %f',Output_table(:,1)
    ,Output_table(:,2))
24 //disp(Output_table)
25
26 //Table 6.1
27 mprintf('\n \n Table 6.1  %s','')
28 mprintf('\n          ui          yi          Linear Model
    Quadratic Model  %s','')
29
30 mprintf('\n          %f          %f          %f          %15f',Output_table)
31
32
33 //Error calculations ----(This is not given in book-
    requires understanding of statistics)
34 Yhat=X*Bhat; //Predicted Y from regression variables
35 S_lin=(Y-Yhat)'*(Y-Yhat); //Sum of errors in Y for
    linear model ---eqn 6.9
36 S_quad=(Y-X_new*Bhat_new)'*(Y-X_new*Bhat_new); //
    Errors in Y for quadratic model
37 mprintf('\n          %25s%f          %10s%f', 'S=',S_lin,'S=',
    S_quad)
38
39 n=length(fuel);
40 sigma=S_lin/(n-1)*(inv(X'*X));
41 bounds=(sigma.^0.5)/sqrt(n)*2.262;
42
43 mprintf('\n The errors in Bhats are not calculated
    because the procedure is not...
44 \n given in the solution of the example')

```

---

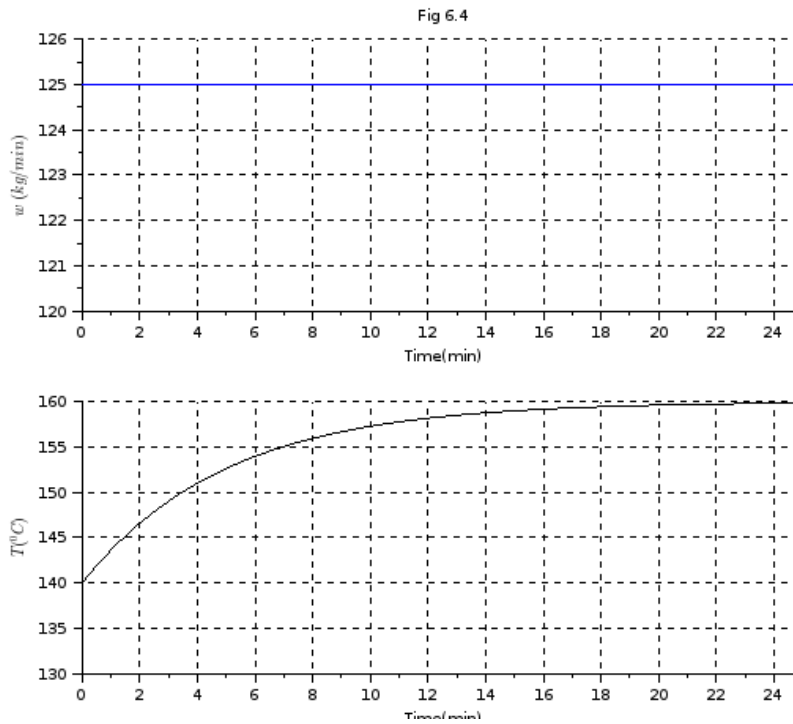


Figure 6.1: Continuous stirred tank reactor

Scilab code Exa 6.2 Continuous stirred tank reactor

```

1 clear
2 clc
3
4 //Example 6.2
5 disp('Example 6.2 ')
6
7 deltaw=5; //kg/min
8 deltaT=20; //deg C

```

```

9 K=deltaT/deltaw
10 tau=5//min
11 T=140+0.632*20;//152.6 deg C
12
13 s=%s;
14 G=4/(5*s+1); //G=T'(s)/W'(s)
15
16 mprintf( '\n T(s)/W(s)=%s', '' )
17 disp(G)
18
19 t=0:0.01:25;
20 n=length(t);
21 w=5*ones(1,n);
22 yt=csim(w,t,G)+140;
23 wt=w+120;
24 subplot(2,1,2);
25 plot2d(t,yt,rect=[0,130,25,160]);
26 xtitle("","Time(min)","$T(^{\circ}C)$")
27 xgrid();
28 subplot(2,1,1);
29 plot2d(t,wt,rect=[0,120,25,126],style=2)
30 xtitle("Fig 6.4","Time(min)","$w\ (kg/min)$")
31 xgrid();

```

---

Scilab code Exa 6.3 Off gas CO2 concentration response

```

1 clear
2 clc
3
4 //Example 6.3
5 disp('Example 6.3')
6
7

```

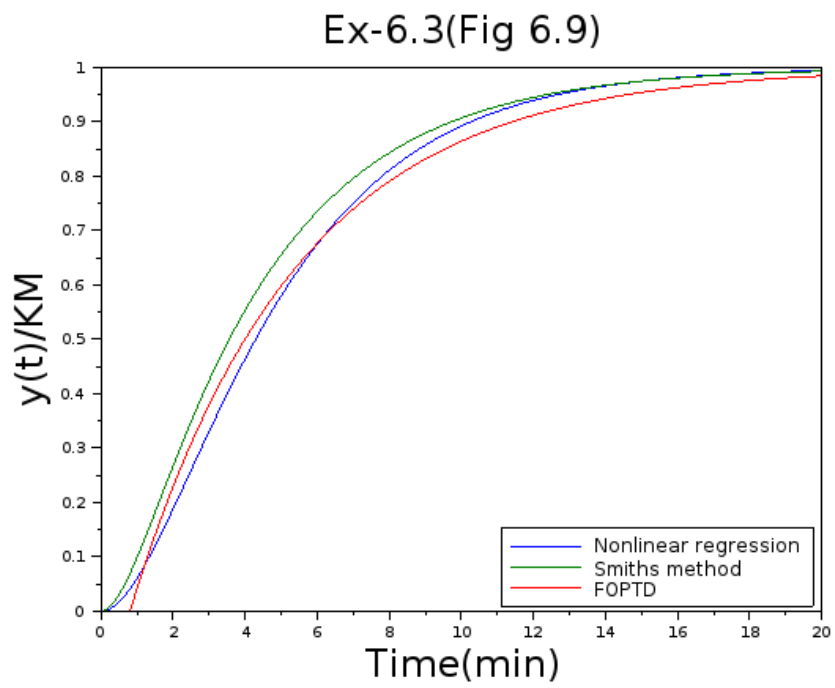


Figure 6.2: Off gas CO2 concentration response

```

8 //Smith's method
9 t20=1.85;//min
10 t60=5;//min
11 ratio=t20/t60;
12 zeta=1.3;//Zeta obtained from Fig 6.7 page 109
13 tau=t60/2.8//Value 2.8 obtained from Fig 6.7
14
15 tau1=tau*zeta+tau*sqrt(zeta^2-1);
16 tau2=tau*zeta-tau*sqrt(zeta^2-1);
17
18 mprintf('From Smiths method \n tau1=%f min\n tau2=
%f min \n',[tau1 tau2])
19
20 //Nonlinear regression
21 //This method cannot be directly used here because
we do not have the data with us
22 //Had the data been given in tabular form we could
have performed a regression
23 //Converting graphical data(Fig 7.8) printed in
textbook to tabular form is not practical
24 //Towards the end of the program however a
roundabout way has been implemented so
25 //that the user can learn the technique of non-
linear optimization
26
27
28
29 s=%s;
30 G2=1/((tau1*s+1)*(tau2*s+1)) //Smith's method
31 G3=1/(4.60*s+1)//First order with time delay: Note
that we cannot have exp(-0.7s) without symbolic
toolbox so we use a roundaround trick for this
32 //Also note that we could have used Pade's
approximation but that works well only for very
small time delays
33 G1=1/((3.34*s+1)*(1.86*s+1)); //Nonlinear regression
34
35 Glist=syslin('c',[G1 G2 G3]') //Simply collating

```



```

        them together for further simulation
36
37 G=syslin('c',Glist);
38 t=0:0.1:20;
39 y=csim('step',t,G);
40 y(3,:)=[zeros(1,8) y(3,1:$-8)] //This is the
        roundabout trick to introduce time lag by
        manually
41 //shifting the response by 0.7 min
42 plot(t,y)
43 xtitle('Ex-6.3(Fig 6.9)', 'Time(min)', 'y(t)/KM');
44 a=legend("Nonlinear regression", "Smiths method", "
        FOPTD", position=4);
45 a.font_size=2;
46 a=get("current_axes");b=a.title;b.font_size=5;c=a.
        x_label;c.font_size=5;
47 c=a.y_label;c.font_size=5;
48
49
50 //====NON-LINEAR REGRESSION====//
51 //Now that we have the response data and also taking
        the word from the book that
52 //simulation from Excel/Matlab is identical to
        experimental data, we can actually
53 //take this simulation and use it for showing
        regression
54 //So our experimental data is y(1)
55 //For nonlinear regression we define a cost function
        which we have to minimize
56 function err=experiment(tau)
57     s=%s;
58     G=syslin('c',1/((tau(1)*s+1)*(tau(2)*s+1)));
59     t=0:0.1:20;
60     response=csim('step',t,G);
61     err=sum((response-y(1,:)).^2);
62 endfunction
63
64 //f is value of cost function, g is gradient of cost

```

```

        function ,
65 //ind is just a dummy variable required by optim
        function
66 function [f,g,ind]=cost(tau,ind)
67     f=experiment(tau)
68     g=numdiff(experiment,tau)
69 endfunction
70
71 x0=[3 1]'; //Initial guess for optimization routine
72 [cost_opt, tau_opt]=optim(cost,x0)
73 mprintf('\n Optimization using optim function done
        successfully \n')
74 mprintf('\n From nonlinear regression \n  tau1=%f
        min\n  tau2=%f min \n',[tau_opt]')
75
76
77
78 //Formatted output...
79 mprintf('\n
        tau1(min) tau2(min)
        Sum of squares')
80 mprintf('\n      Smith      %f %f %f'
        ,3.81,0.84,0.0769)
81 mprintf('\n First Order\n(delay=0.7min)      %f
        %s      %f',4.60,'—',0.0323)
82 mprintf('\n Excel and Matlab      %f %f %f \n'
        ,3.34,1.86,0.0057)

```

---

#### Scilab code Exa 6.5 Estimation of model parameters

```

1 clear
2 clc
3
4 //Example 6.5
5 disp('Example 6.5 ')
6

```

```

7 k=0:10';
8 yk=[0 0.058 0.217 0.360 0.488 0.600 0.692 0.772
      0.833 0.888 0.925]';
9
10 Y=yk(2:$);
11 n=length(Y);
12
13 yk_1=[yk(1:$-1)];
14 yk_2=[0;yk(1:$-2)];
15 uk_1=ones(n,1);
16 uk_2=[0;uk_1(1:$-1)];
17
18 X=[yk_1 yk_2 uk_1 uk_2];
19
20 Bhat=inv(X'*X)*X'*Y;//Equation 6-10
21 //a1, a2, b1, b2 are components of Bhat for linear
    regression
22 K_lin=(Bhat(3)+Bhat(4))/(1-Bhat(1)-Bhat(2)); //
    Equation 6-42
23
24 //====NON-LINEAR REGRESSION====//
25 //Now that we have the response data we can do non-
    linear regression through
26 //the transfer function approach. Total no. of
    variables to be regressed are
27 //three: K, tau1, tau2.
28 //For nonlinear regression we define a cost function
    which we have to minimize
29
30
31 function err=experiment(tau)
32     K=tau(3);tau1=tau(1);tau2=tau(2);
33     t=k';
34     response=tau(3)*(1-(tau1*exp(-t/tau1)-tau2*exp(-
        t/tau2))/(tau1-tau2))
35     err=sum((response-[yk]).^2);
36 endfunction
37

```

```

38 //f is value of cost function , g is gradient of cost
    function ,
39 //ind is just a dummy variable required by optim
    function
40 function [f,g,ind]=cost(tau,ind)
41     f=experiment(tau)
42     g=numdiff(experiment,tau)
43 endfunction
44
45 x0=[1 3 1]'; //Initial guess for optimization
    routine
46 [cost_opt, tau_opt]=optim(cost,x0)
47 mprintf('\n Optimization using optim function done
    successfully \n')
48 mprintf('\n From nonlinear regression \n tau1=%f \n
    tau2=%f min \n K=%f min \n',[tau_opt]')
49
50 //Now we have to get discrete ARX model parameters
    from transfer function parameters
51 //For this we use Equation nos.: 6-32 to 6-36
52
53 deltat=1;taua=0;tau1=tau_opt(1);tau2=tau_opt(2);K=
    tau_opt(3);
54 a1=exp(-deltat/tau1)+exp(-deltat/tau2);
55 a2=-exp(-deltat/tau1)*exp(-deltat/tau2);
56 b1=K*(1+(taua-tau1)/(tau1-tau2)*exp(-deltat/tau1)-(
    taua-tau2)/(tau1-tau2)*exp(-deltat/tau2));
57 b2=K*(exp(-deltat*(1/tau1+1/tau2))+(taua-tau1)/(tau1
    -tau2)*exp(-deltat/tau2)-(taua-tau2)/(tau1-tau2)*
    exp(-deltat/tau1));
58
59 mprintf("\n          Linear Regression          Non-
    Linear Regression")
60 mprintf("\n          %s          %f          %20f",["a1";"a2"
    ;"b1";"b2";"K"],[[Bhat;K_lin] [a1;a2;b1;b2;K]])
61
62 yL_hat=X*Bhat;
63 yN_hat=X*[a1;a2;b1;b2];

```

```

64
65 mprintf(" \n \n      n          y          yL_hat
          yN_hat")
66 mprintf(" \n      %f      %f      %f      %f" , [1:10] ', yk
          (2:$), yL_hat , yN_hat)
67
68 mprintf(" \n \n Note that values of coefficients for
          non-linear regression \n are different...
69 from that of linear regression , but the \n output is
          the same \n...
70 hence this shows that the coefficients need not be
          unique....
71 \n the coefficients for nonlinear regression given
          in book and from this scilab code \n...
72 both are correct")

```

---

### Scilab code Exa 6.6 Step test of distillation column

```

1 clear
2 clc
3
4 //Example 6.6
5 disp('Example 6.6 ')
6
7 mprintf(" \n It is not possible to fit Model 1 or \n
          plot it because experimental data...
8 has not been given in the book. \n Hence we
          simply plot Model 2,3,4 \n")
9
10
11 //Model 2
12 a=[3.317 -4.033 2.108 0.392 ]'
13 b=[-0.00922 0.0322 -0.0370 0.0141]';

```

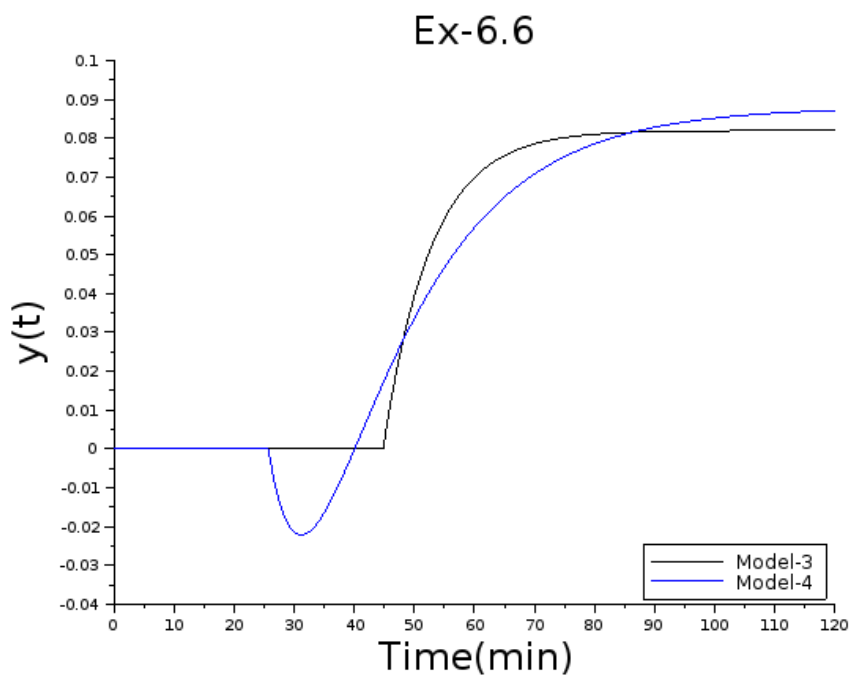


Figure 6.3: Step test of distillation column

```

14 uk=[ones(120,1)]; //Inputs-step at t=1 min
15 yk=zeros(120,1); //Outputs initialization
16
17 for k=5:120
18     yk(k)=a(1)*yk(k-1)+a(2)*yk(k-2)+a(3)*yk(k-3)+a
        (4)*yk(k-4)...
19         +b(1)*uk(k-1)+b(2)*uk(k-2)+b(3)*uk(k-3)+
        b(4)*uk(k-4);
20 end
21 //Model 2 trial with transfer function
22 //a=flipdim([-1 3.317 -4.033 2.108 0.392] ',1);
23 //b=flipdim([-0.00922 0.0322 -0.0370 0.0141] ',1);
24 //
25 //Gz=poly(b,"z","coeff")/poly(a,"z","coeff");
26 //u=ones(120,1);
27 //Gz=syslin('d',Gz);
28 //yk=flts(u',Gz)
29
30 //Although the code is correct, the values given in
    the book for the coeffs
31 //of the ARX model are wrong and hence the system
    diverges (blows up)
32
33 mprintf('Although the code is correct, the values \n
        given in the book for the coeffs of model 2...
34 \n of the ARX model are wrong and hence the system
        diverges (blows up)')
35
36 //Model 3
37 s=%s;
38 G3=0.082/(7.95*s+1); //We have to add delay of 44.8
    min
39 //Model 4
40 G4=0.088*(1-12.2*s)/(109.2*s^2+23.1*s+1); //We have
    to add delay of 25.7 min
41
42 G=syslin('c',[G3;G4]);
43 t=[0:0.1:120]';

```

```
44 y=csim('step',t,G);
45
46 y(1,:)=[zeros(1,448) y(1,1:$-448)]
47 y(2,:)=[zeros(1,257) y(2,1:$-257)]
48 plot2d(t,y')
49
50 xtitle('Ex-6.6', 'Time(min)', 'y(t)');
51 a=legend("Model-3", "Model-4", position=4);
52 a.font_size=2;
53 a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
54 c=a.y_label;c.font_size=5;
```

---



# Chapter 8

## Control System Instrumentation

Scilab code Exa 8.2 Pump and heat exchanger system

```
1 clear
2 clc
3
4 //Example 8.2
5 disp('Example 8.2 ')
6
7 //Eqn 8-6
8
9 //Pump characteristics
10 q=0:0.1:240;
11 Phe=30*(q/200).^2;
12 plot2d(q,Phe,rect=[0,0,240,40]);
13 xgrid()
14 xtitle("Fig 8.13 Pump characteristics", "q, gal/min", "
    P, psi")
15 scf();
16
17 q=200; //Flow rate in gal/min
18 Phe=30*(q/200).^2;
```

```

19 Pv=40-Phe; //Eqn 8-8
20
21
22 //(a)
23 l=0.5; Pv=10;
24 Cv=q/l/sqrt(Pv);
25
26 mprintf("(a) The value of coefficient Cv is %f",Cv)
27
28 //plotting valve characteristic curve
29
30 l=[0:0.01:0.8]';
31 n=length(l);
32 Cv=125;
33
34 function y=valve_1(q)
35     Pv=40-30*(q/200).^2;
36 y=Cv*l.*sqrt(Pv)-q;
37 endfunction
38
39 [q_valve1,f1]=fsolve(200*ones(n,1),valve_1); //200*
    ones(n,1) is the initial guess for q
40
41 plot2d(l,q_valve1);
42
43 //(b)
44 q=200*110/100; //110% flow rate
45 Phe=30*(q/200).^2;
46 Pv=40-Phe; //Eqn 8-8
47 l=1;
48 Cv=q/sqrt(Pv)/l;
49 mprintf("\n(b) The value of coefficient Cv is %f",Cv
    )
50
51 //We use Cv=115;
52 Cv=115;
53 l=[0.2:0.01:0.9]';
54 n=length(l);

```

```

55 R=50;
56
57 function y=valve_2(q)
58     Pv=40-30*(q/200).^2;
59     y=[R^(1-1)]*Cv.*sqrt(Pv)-q;
60 endfunction
61 [q_valve2,f2]=fsolve(150*ones(n,1),valve_2);
62 plot2d(1,q_valve2,style=2)
63
64 //(c)
65 Cv=1.2*115;
66 mprintf("\n(c) The value of coefficient Cv is %f",Cv
    )
67
68 l=[0.2:0.01:0.9]';
69 n=length(l);
70 R=50;
71
72 function y=valve_3(q)
73     Pv=40-30*(q/200).^2;
74     y=[R^(1-1)]*Cv.*sqrt(Pv)-q;
75 endfunction
76 [q_valve3,f3]=fsolve(linspace(60,200,n)',valve_3);
77 //Initial guess has to be smart for each valve,
78 //since we want near linear profile we can give a
79 //linear initial guess
80 plot2d(1,q_valve3,style=3)
81
82 //(d)
83 Cv=0.8*115;
84 mprintf("\n(d) The value of coefficient Cv is %f",Cv
    )
85
86 l=[0.2:0.01:0.9]';
87 n=length(l);
88 R=50;
89
90 function y=valve_4(q)

```

```

89     Pv=40-30*(q/200).^2;
90     y=[R^(1-1)]*Cv.*sqrt(Pv)-q;
91 endfunction
92 [q_valve4,f4]=fsolve(linspace(60,200,n)',valve_4);
93 //Initial guess has to be smart for each valve,
94 //since we want near linear profile we can give a
95 //linear initial guess
96 plot2d(1,q_valve4,style=4,rect=[0,0,1,240])
97
98 xtitle('Ex-8.2 Installed valve characteristics','l$
99       ', 'q gal/min');
100 a=legend(" Valve 1, linear Cv=125", " Valve 2, Equal%
101       Cv=115", " Valve 3, Equal% Cv=138", " Valve 4, Equal%
102       Cv=92", position=4);
103 a.font_size=2;
104 a=get("current_axes");b=a.title;b.font_size=3;c=a.
105     x_label;c.font_size=5;
106 c=a.y_label;c.font_size=5;

```

---

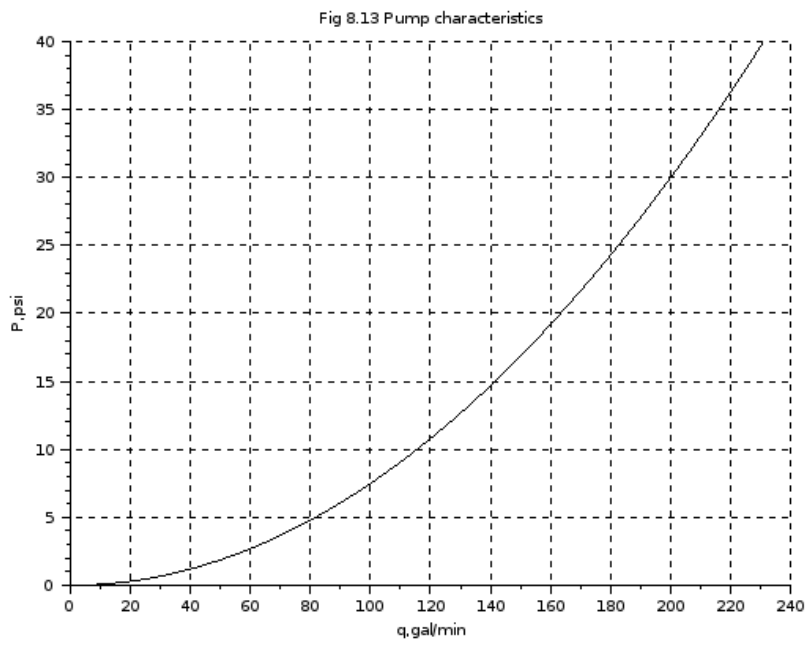


Figure 8.1: Pump and heat exchanger system

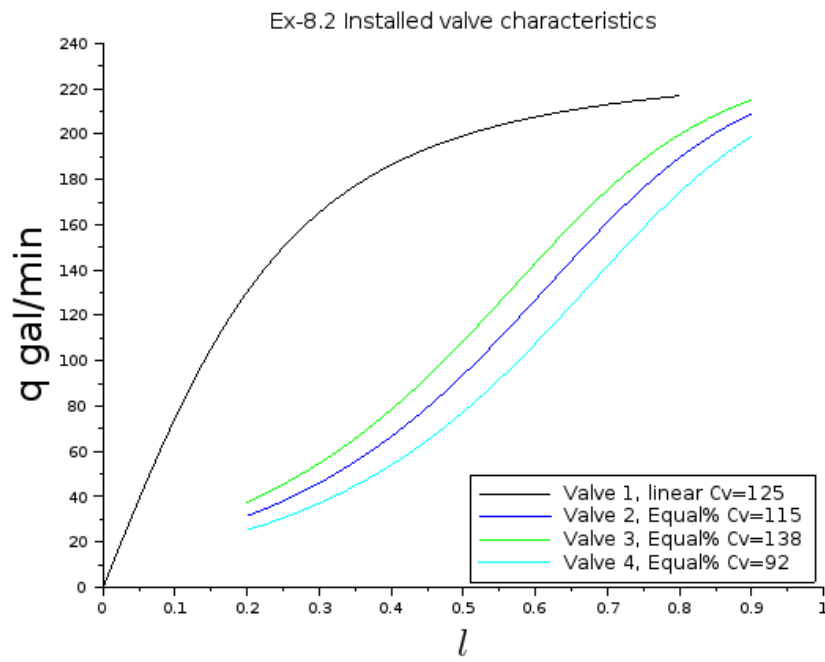


Figure 8.2: Pump and heat exchanger system

# Chapter 9

## Process Safety and Process Control

Scilab code Exa 9.1 Liquid surge system

```
1 clear
2 clc
3
4 //Example 9.1
5 disp('Example 9.1 ')
6
7 q1=12; //cub ft/min
8 q2=6;
9 q3=13;
10 A=%pi*3^2; //ft^2
11 delta_t=10; //min
12 delta_h_max=delta_t*(q1+q2-q3)/A;
13
14 mprintf('Alarm should be at least %f ft below top of
        tank ',delta_h_max)
```

---

Scilab code Exa 9.2 Abnormal event in distillation column

```
1 clear
2 clc
3
4 //Example 9.2
5 disp('Example 9.2')
6
7 mu=[0.5 0.8 0.2]; //population means of z y x
8 S=[0.01 0.020 0.005]; //population std dev of z y x
9
10 z=[0.485]; //steady state values
11 y=[0.825];
12 x=0.205;
13
14 F=4;D=2;B=2; //flow rates
15
16 Ec=F*z-D*y-B*x;
17
18 disp(Ec,"Ec=")
19
20 sigma_Ec=sqrt(F^2*S(1)^2+D^2*S(2)^2+B^2*S(3)^2)
21
22 disp(sigma_Ec,"sigma_Ec")
23
24
25
26 Z=(Ec-0)/sigma_Ec;
27
28 disp(Z,"Z=");
29
30 [P,Q]=cdfnor("PQ",0.120,0,sigma_Ec);
31
32 //Since P is close to 1, we use Q
33
34 Probability=1-2*Q;
35
36 disp(Probability,"Probability of abnormal event=")
```



---

Scilab code Exa 9.3 Reliability of flow control loop

```
1
2 clear
3 clc
4
5 //Example 9.3
6 disp('Example 9.3')
7
8 mu=[1.73 0.05 0.49 0.60 0.44]'; //failures/yr
9 R=exp(-mu);
10 P=1-R;
11
12 R_overall=prod(R);
13 P_overall=1-R_overall;
14 mu_overall=-log(R_overall);
15 MTBF=1/mu_overall;
16
17 mprintf("MTBF= %f yr",MTBF)
```

---

# Chapter 10

## Dynamic behavior

Scilab code Exa 10.2 Set point response of level control system

```
1 clear
2 clc
3
4 //Example 10.2
5 disp('Example 10.2')
6
7 A=%pi*0.5^2; //Square meters
8 R=6.37;
9 Kp=R//min/sq.m=R
10 tau=R*A;
11
12 Km=100/2; //% per meter
13
14 l=0.5;
15 q=0.2*30^(1-1);
16 dq_d1=0.2*log(30)*30^(1-1); //cu.meter/min Eqn 10-48
17
18 Kip=(15-3)/100; //psi/%
19 d1_dpt=(1-0)/(15-3); //psi^-1
```

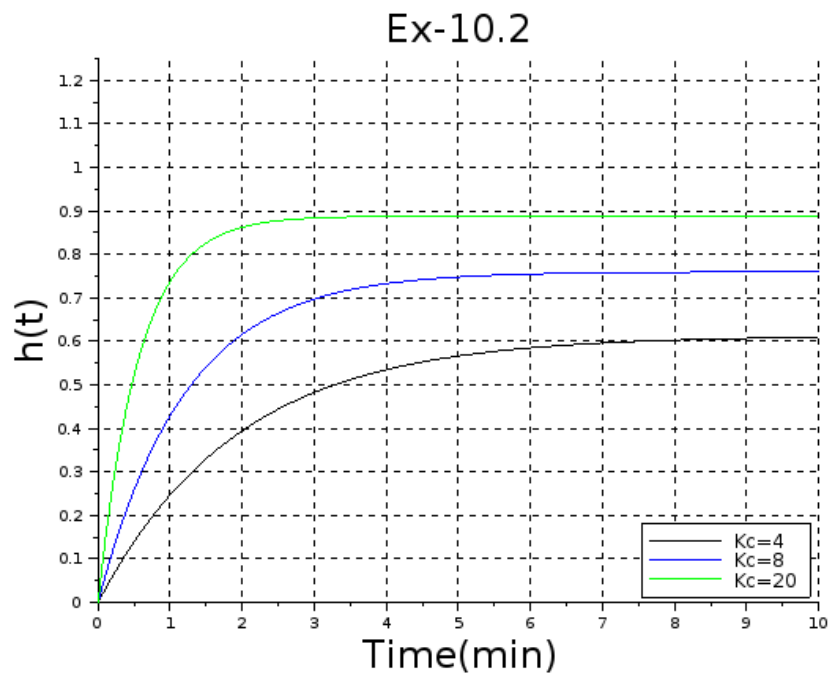


Figure 10.1: Set point response of level control system

```

20
21 Kv=dq_dl*dl_dpt //Eqn 10-50
22
23 Kc=[4 8 20]'; //different values of Kc that we have
    to try
24 K_OL=Kc*Kv*Kp*Km*Kip;//Open loop gain Eqn 10-40
25
26 K1=K_OL./(1+K_OL);//Eqn 10-38
27 tau1=tau./(1+K_OL); //Eqn 10-39
28
29 //Now we simulate the close loop process for these
    different values of K1 and tau1
30 s=%s;
31 G=K1./(tau1*s+1);
32 G=syslin('c',G);
33 t=[0:0.1:10]'; //time in minutes
34 hdash=csim('step',t,G)';
35
36 plot2d(t,hdash,rect=[0,0,10,1.25])
37 xgrid()
38 xtitle('Ex-10.2','Time(min)','h(t)');
39 a=legend("Kc=4","Kc=8","Kc=20",position=4);
40 a.font_size=2;
41 a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
42 c=a.y_label;c.font_size=5;

```

---

Scilab code Exa 10.3 Disturbance response of level control system

```

1 clear
2 clc
3
4 //Example 10.3

```

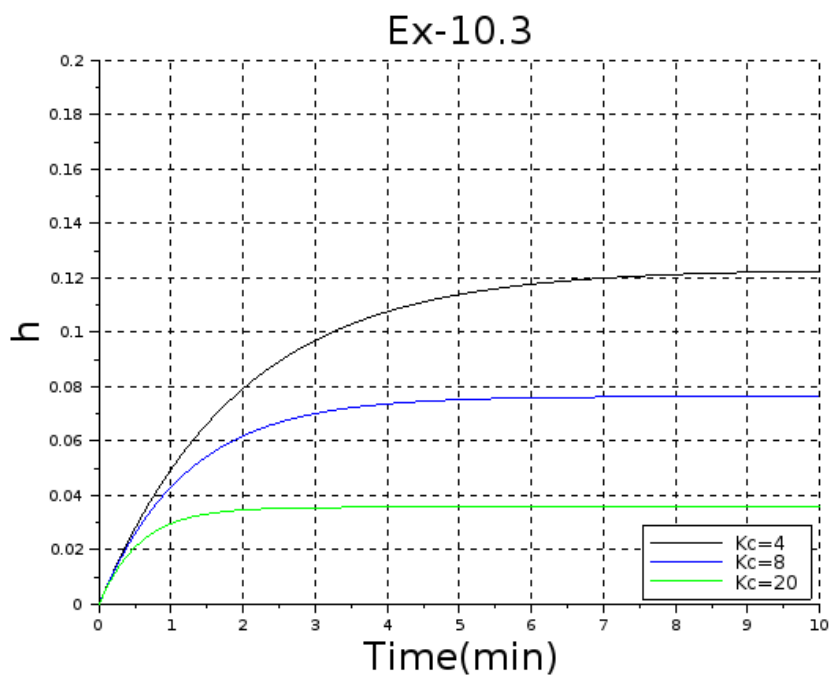


Figure 10.2: Disturbance response of level control system

```

5 disp('Example 10.3')
6
7 //This example draws upon the calculations of Ex
  10.2 and hence it has been
8 //reproduced
9 A=%pi*0.5^2; //Square meters
10 R=6.37;
11 Kp=R//min/sq.m=R
12 tau=R*A;
13 Km=100/2 //% per meter
14 l=0.5;
15 q=0.2*30^(1-1);
16 dq_dl=0.2*log(30)*30^(1-1); //cu.meter/min Eqn 10-48
17 Kip=(15-3)/100; //psi/%
18 dl_dpt=(1-0)/(15-3); //psi^-1
19 Kv=dq_dl*dl_dpt //Eqn 10-50
20 Kc=[4 8 20]'; //different values of Kc that we have
  to try
21 K_OL=Kc*Kip*Kv*Kp*Km;//Open loop gain Eqn 10-40
22 K1=K_OL./(1+K_OL);//Eqn 10-38
23 tau1=tau./(1+K_OL); //Eqn 10-39
24
25 //=====Example 11.3 now starts here=====//
26 //Now we simulate the close loop process for these
  different values of K2 and tau1
27 M=0.05;//Magnitude of step
28 K2=Kp./(1+K_OL);
29 s=%s;
30 G=K2./(tau1*s+1);
31 G=syslin('c',G);
32 t=[0:0.1:10]'; //time in minutes
33 hdash=M*csim('step',t,G)';
34
35 plot2d(t,hdash,rect=[0,0,10,0.2])
36 xgrid()
37 xtitle('Ex-10.3','Time(min)','h');
38 a=legend("Kc=4","Kc=8","Kc=20",position=4);
39 a.font_size=2;

```

```

40 a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
41 c=a.y_label;c.font_size=5;
42
43 offset=-Kp*M./(1+K_OL);
44
45 mprintf("    Kc    Offset")
46 mprintf("\n%f    %f",[Kc offset])
47
48 mprintf("\n\nNote that the book has a mistake in the
    question and legend of fig 10.19\n...
49 the values of Kc should be 4,8,20 and not 1,2,5\n...
50 this mistake is there because in the second edition
    of the book\n...
51 Kc has values 1 2 5 but then level transmitter span
    is 0.5 instead of 2")

```

---

#### Scilab code Exa 10.4 Stability of feedback control system

```

1 clear
2 clc
3
4 //Example 10.4
5 disp('Example 10.4')
6
7 Km=1; //We take set point gain as 1 for illustrative
    purposes
8 Kc=[15 6 2]'; //different values of Kc for which we
    will simulate
9 Gc=Kc;
10 s=%s;
11 Gv=1/(2*s+1);
12 Gd=1/(5*s+1);

```

### Ex-10.4

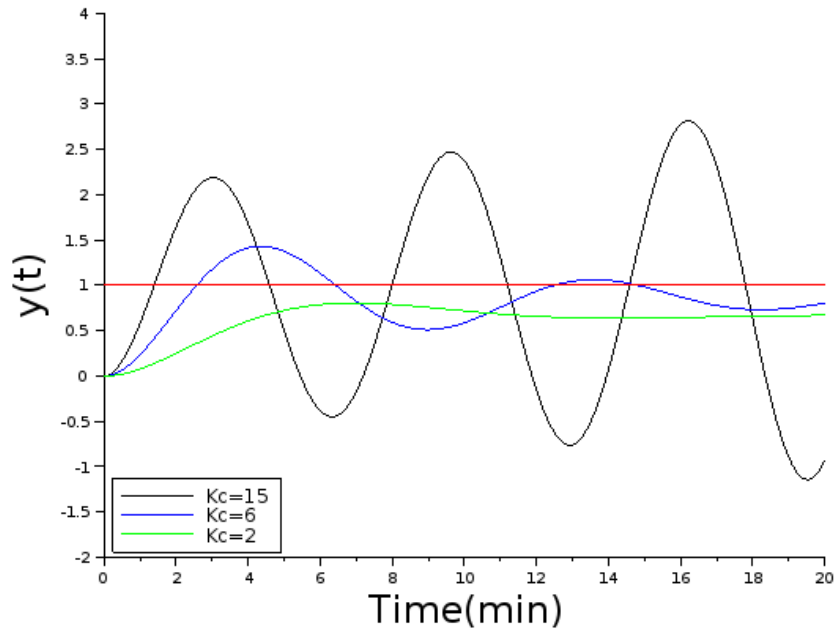


Figure 10.3: Stability of feedback control system



```

13 Gp=Gd;
14 Gm=1/(s+1);
15
16 G=Km*Gc*Gv*Gp./(1+Km*Gc*Gv*Gp*Gm); //Eqn 10-75 G=Y/
    Ysp
17
18 //Now we simulate the close loop process for these
    different values of Kc
19
20 G=syslin('c',G);
21 t=[0:0.1:20]'; //time in minutes
22 Y=csim('step',t,G)';
23
24 plot2d(t,Y,rect=[0,-2,20,4])
25 plot2d(t,ones(length(t),1),style=5)
26 xtitle('Ex-10.4','Time(min)','y(t)');
27 a=legend("Kc=15","Kc=6","Kc=2",position=3);
28 a.font_size=2;
29 a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
30 c=a.y_label;c.font_size=5;

```

---

#### Scilab code Exa 10.10 Routh Array 1

```

1 clear
2 clc
3
4 //Example 10.10
5 disp('Example 10.10')
6
7
8 s=%s;
9 Gp=1/(5*s+1);
10 Gm=1/(s+1);
11 Gv=1/(2*s+1);

```

```

12 Ys=Gv*Gp*Gm
13
14 Routh=routh_t(Ys,poly(0,"Kc")); // produces routh
    table for polynomial 1+Kc*Ys
15 disp(Routh)
16 K1=roots( numer(Routh(3,1)));
17 K2=roots( numer(Routh(4,1)));
18
19 mprintf('K lies between %f and %f for system to be
    stable ', K2,K1)

```

---

#### Scilab code Exa 10.11 Routh Array 2

```

1 clear
2 clc
3
4 //Example 10.11
5 disp('Example 10.11 ')
6
7 Kc=poly(0,"Kc");//defining a polynomial variable
8 a2=2.5;a1=5.5-Kc;a0=1+2*Kc;//a# are coefficients
9 b1=(a1*a0-a2*0)/a1;
10 mprintf("Routh Array is")
11 A=[a2 a0;a1 0;b1 0]
12 disp(A)
13
14 mprintf("\n All entries in first column should be
    positive")
15
16 Kc_up=roots(a1);//upper limit for Kc by solving
    third row column 1 of array
17 b1=numer(b1);//This is done to extract the numerator
    from rational cl
18 //without extracting numerator we cannot find roots
    using "roots" function

```

```

19 Kc_ul=roots(b1); //lower limit for Kc
20
21 mprintf("\n\nThe values of Kc for system to be
    stable are \n    %f<Kc<%f",Kc_ul,Kc_up);

```

---

#### Scilab code Exa 10.12 Direct substitution to find stability

```

1 clear
2 clc
3
4 //Example 10.12
5 disp('Example 10.12 ')
6
7 w=poly(0,"w")
8 s=%i*w; //j or iota is i
9 G=10*s^3+17*s^2+8*s+1; //Kc has been removed here
    because in a single expression
10 //two polynomials are not allowed
11 w=roots(imag(G));
12 p=-real(G) //Real part of G
13 Kc=horner(p,w)
14
15 mprintf("The values outside which system is unstable
    \nare %f and %f",Kc(1),Kc(3))

```

---

#### Scilab code Exa 10.13 Root Locus

```

1 clear
2 clc
3
4 //Example 10.13

```

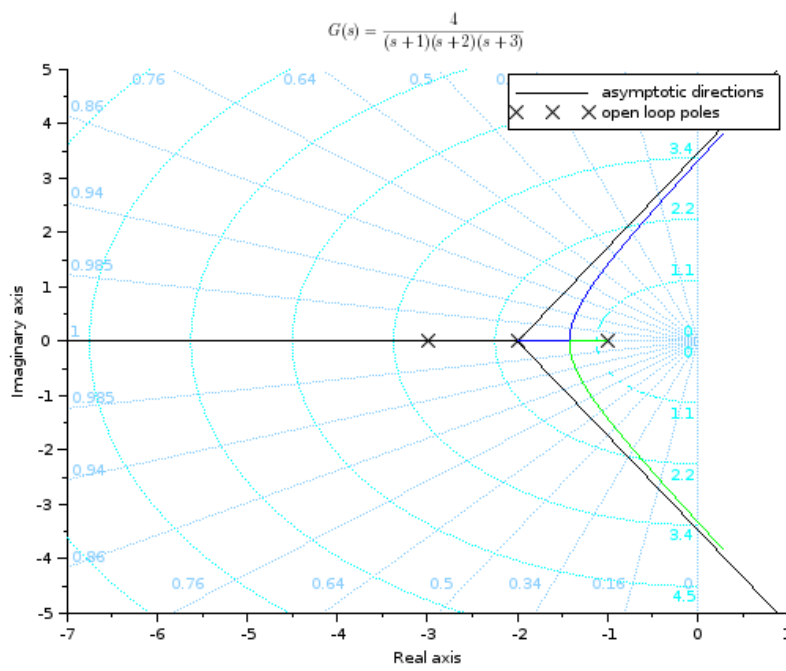


Figure 10.4: Root Locus

```

5 disp('Example 10.13 ')
6
7 s=%s;
8 G=4/((s+1)*(s+2)*(s+3));
9 G=syslin('c',G);
10 [ki,s_i]=kpure(G);
11 evans(G,ki*1.5); // plots for until K = 1.5*ki
12 //disp(G,"For G=");disp(ki,"K=")
13 disp(ki,"Max value of k for which we have closed
    loop stability",G,"For G=")
14 xtitle("$G(s)=\frac{4}{(s+1)(s+2)(s+3)}$")
15 sgrid();

```

---

#### Scilab code Exa 10.14 Transient response from root locus

```

1 clear
2 clc
3
4 //Example 10.14
5 disp('Example 10.14 ')
6
7 s=%s;
8 G=4/((s+1)*(s+2)*(s+3));
9 K=10; //given in question
10 p=1+K*G;//characteristic equation
11 q=roots( numer(p));
12
13 q_abs=abs(q);
14 q_real=real(q);
15 q_imag=imag(q);
16 d=q_abs(2);
17 psi=%pi-acos(q_real./q_abs);//angle in radians
18 tau=1/d;
19 eta=cos(psi)
20

```

```
21 mprintf(" \nd=%f\npsi=%f degrees \ntau=%f time units \neta=%f", d, psi(2)*180/%pi, tau, eta(2))
22
23 mprintf(" \n\nPlease note that the answers given in
    book are wrong")
```

---

# Chapter 11

## PID Controller Design Tuning and Troubleshooting

Scilab code Exa 11.1 Direct synthesis for PID

```
1 clear
2 clc
3
4 //Example 11.1
5 disp('Example 11.1 ')
6
7 //(a) Desired closed loop gain=1 and tau=[1 3 10]
8 s=%s;
9 tauc=[1 3 10]';
10 tau1=10;tau2=5;K=2;theta=1; //Time delay
11 Y_Ysp=(1)./(tauc*s+1); //Y/Ysp=delay/(tau*s+1) Eqn
    11-6
12
13 //delay=(1-theta/2*s+theta^2/10*s^2-theta^3/120*s^3)
    /(1+theta/2*s+theta^2/10*s^2+theta^3/120*s^3);//
    Third order pade approx
14 delay=(1-theta/2*s)/(1+theta*s/2); //first order Pade
    approx
15
```

```

16 G=(K)./((tau1*s+1)*(tau2*s+1))*delay;
17 G_tilda=G//Model transfer function
18
19 //Eqn-11-14
20 Kc=1/K*(tau1+tau2)./(tauc+theta);tauI=tau1+tau2;tauD
    =tau1*tau2/(tau1+tau2);
21 Gc=Kc*(1+1/tauI/s+tauD*s); //PID without derivative
    filtering
22 G_CL=syslin('c',Gc/delay*G./(1+Gc*G));//closed loop
    transfer function
23 t=0:160;
24 y=csim('step',t,G_CL);
25 //plot(t,y)
26
27 t_d=81:160;
28 G_CL_dist=syslin('c',G/delay./(1+Gc*G));//closed
    loop wrt disturbance
29 u_d=[0 ones(1,length(t_d)-1)]
30 y_d=csim('step',t_d,G_CL_dist);
31 y(:,81:160)=y(:,81:160)+y_d
32 plot(t,y)
33
34 xgrid()
35 xtitle('Ex-11.1 Correct Model','Time(min)','y(t)');
36 a=legend("$\tau_c=1$","$\tau_c=3$","$\tau_c=10$",
    position=4);
37 a.font_size=2;
38 a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
39 c=a.y_label;c.font_size=5;
40
41 mprintf("\n                tauc=1        tauc=3
    tauc=10")
42 mprintf("\n Kc(K_tilda=2) %10f      %f      %f",Kc');
43
44
45 //Simulation for model with incorrect gain
46 scf()

```



```

47 K_tilda=0.9
48
49 //Eqn-11-14
50 Kc=1/K_tilda*(tau1+tau2)./(tauc+theta);tauI=tau1+
    tau2;tauD=tau1*tau2/(tau1+tau2);
51 Gc=Kc*(1+1/tauI/s+tauD*s)
52 mprintf("\n Kc(K_tilda=0.9) %10f      %f      %f",Kc')
    ;
53 mprintf("\n tauI %20f      %f      %f",tauI*ones(1,3))
    ;
54 mprintf("\n tauD %20f      %f      %f",tauD*ones(1,3))
    ;
55
56 G_CL=syslin('c',Gc*G./(1+Gc*G));//closed loop
    transfer function
57 t=0:160;
58 y=csim('step',t,G_CL);
59
60 t_d=81:160;
61 G_CL_dist=syslin('c',G./(1+Gc*G));//closed loop wrt
    disturbance
62 y_d=csim('step',t_d,G_CL_dist);
63 y(:,81:160)=y(:,81:160)+y_d
64 plot(t,y)
65
66 xgrid()
67 xtitle('Ex-11.1 Model with incorrect gain','Time(min
    )','y(t)');
68 a=legend("$\tau_c=1$","$\tau_c=3$","$\tau_c=10$",
    position=4);
69 a.font_size=2;
70 a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
71 c=a.y_label;c.font_size=5;
72
73 mprintf('\n \nThere is a slight mis-match between
    graphs from scilab code\n...
74 and those given in the book because of Pade approx

```

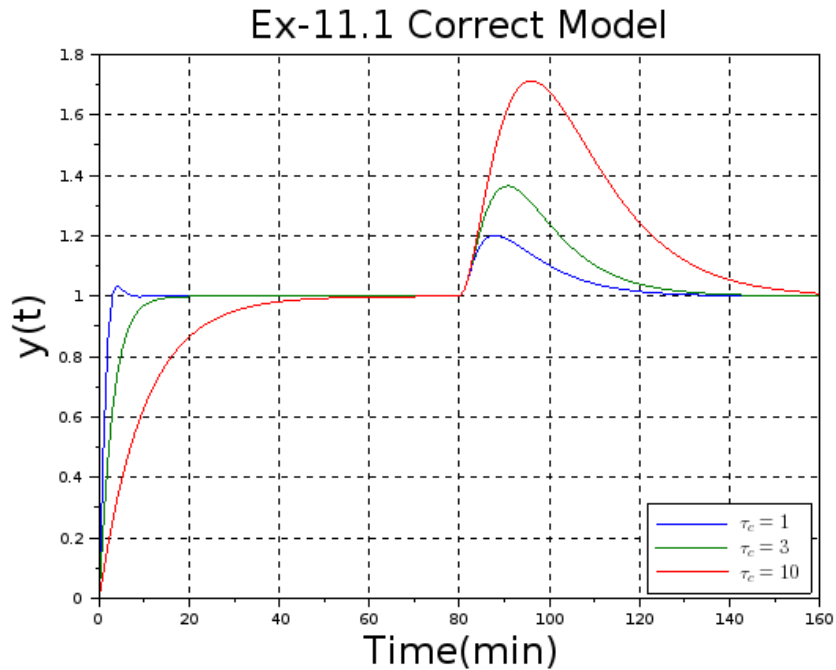


Figure 11.1: Direct synthesis for PID

which is very bad\n...  
 75 for delay being 1. It works only for small delays.  
 Scilab does \n...  
 76 not handle continuous delays and hence this problem  
 cannot \n...  
 77 be circumvented' )

---

Scilab code Exa 11.3 PI and PID control of liquid storage tank

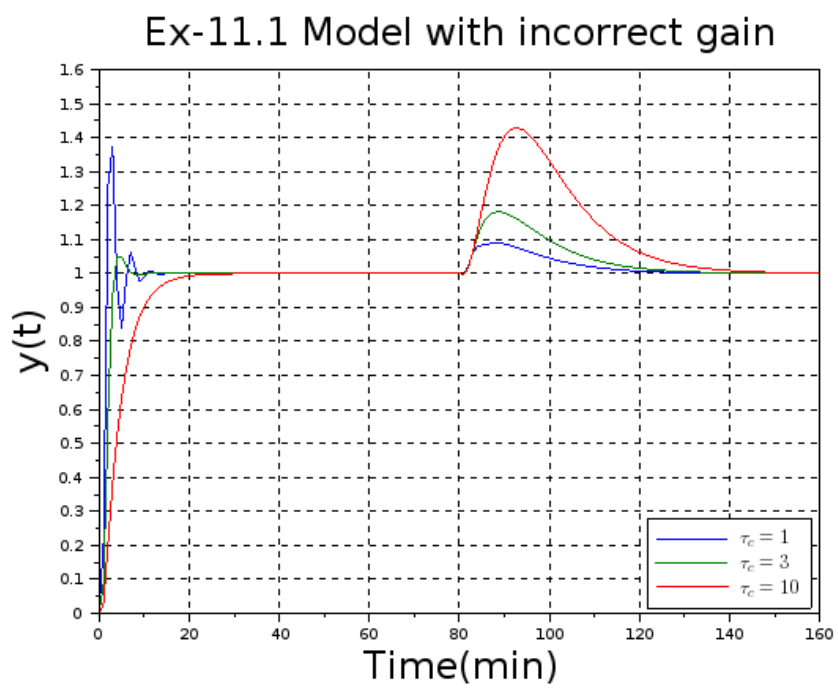


Figure 11.2: Direct synthesis for PID

```

1 clear
2 clc
3
4 //Example 11.3
5 disp('Example 11.3')
6
7 //(a)
8 K=0.2;theta=7.4;tauc=[8 15]';
9
10 Kc1=1/K*(2*tauc+theta)./(tauc+theta).^2; //Row M
11 Kc2=1/K*(2*tauc+theta)./(tauc+theta/2).^2; //Row N
12 tauI=2*tauc+theta;
13 tauD=(tauc*theta+theta^2/4)./(2*tauc+theta);
14
15 mprintf('          Kc          tauI
          tauD')
16 mprintf('\nPI(tauC=8)   %f   %f   %f',Kc1(1),tauI
          (1),0)
17 mprintf('\nPI(tauC=15)  %f   %f   %f',Kc1(2),
          tauI(2),0)
18 mprintf('\nPID(tauC=8)  %f   %f   %f',Kc2(1),
          tauI(1),tauD(1))
19 mprintf('\nPID(tauC=15) %f   %f   %f',Kc2(2),
          tauI(2),tauD(2))
20
21 s=%s;
22
23 //delay=(1-theta/2*s+theta^2/10*s^2-theta^3/120*s^3)
          /(1+theta/2*s+theta^2/10*s^2+theta^3/120*s^3);//
          Third order pade approx
24 delay=(1-theta/2*s+theta^2/10*s^2)/(1+theta/2*s+
          theta^2/10*s^2);//second order pade approx
25 //delay=(1-theta/2*s)/(1+theta/2*s);// first order
          pade approx
26 G=K*delay/s;
27 Gc1=Kc1.*(1+(1)./tauI/s)
28 Gc2=Kc2.*(1+(1)./tauI/s+tauD*s./(0.1*tauD*s+1));//
          PID with derivative filtering

```

```

29 G_CL1=syslin('c',Gc1*G./(1+Gc1*G));
30 G_CL2=syslin('c',Gc2*G./(1+Gc2*G));
31 t=0:300;
32 y1=csim('step',t,G_CL1);
33 y2=csim('step',t,G_CL2);
34 y1(:,1:theta)=0;//accounting for time delay—this is
    required otherwise
35 //an unrealistic inverse response is seen due to the
    pade approx
36 y2(:,1:theta)=0;
37
38 t_d=151:300;
39 G_CL_dist1=syslin('c',G./(1+Gc1*G));//closed loop
    wrt disturbance
40 G_CL_dist2=syslin('c',G./(1+Gc2*G));//closed loop
    wrt disturbance
41 y_d1=csim('step',t_d,G_CL_dist1);
42 y_d1(:,1:theta)=0;//accounting for time delay
43 y_d2=csim('step',t_d,G_CL_dist2);
44 y_d2(:,1:theta)=0;//accounting for time delay
45 y1(:,t_d)=y1(:,t_d)+y_d1;
46 y2(:,t_d)=y2(:,t_d)+y_d2;
47
48 //plot(t,y1)
49 //xgrid()
50 //xtitle('Ex-11.3 PI control','Time(min)','y(t)');
51 //a=legend("$\tau_c=8$","$\tau_c=15$",position=1);
52 //a.font_size=2;
53 //a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
54 //c=a.y_label;c.font_size=5;
55 //scf()
56 //
57 //plot(t,y2)
58 //xgrid()
59 //xtitle('Ex-11.3 PID control','Time(min)','y(t)');
60 //a=legend("$\tau_c=8$","$\tau_c=15$",position=1);
61 //a.font_size=2;

```

```

62 //a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
63 //c=a.y_label;c.font_size=5;
64
65
66 mprintf('\n\nThere is uncertainty as to whether PID
    with derivative filtering\n...
67 to be used or not. Since one gets results by using
    PID with filtering\n...
68 it has been used here. Note that pade approx for
    delay=7.4\n...
69 is totally wrong because it is too gross an approx
    but we have no\n...
70 other way of making delay approx so we have to live
    with it.\n\n...')
71
72
73 //Part (b) Routh Array testing
74 //For frequency response refer to ch-13 for Bode
    Plots
75 G=(1-theta*s)/s;
76 poly_PI=Gc1*G;//denom(G_CL1);//G*Gc for PI
    controller
77 poly_PID=Gc2*G;//G*Gc for PID controller
78
79 Routh1=routh_t(poly_PI(1,1)/1,poly(0,"K")); //
    produces routh table for polynomial 1+Kc*poly
80 disp(Routh1,"Routh1=")
81 Kmax1=roots(numer(Routh1(1,1)));
82
83 Routh2=routh_t(poly_PI(2,1)/1,poly(0,"K")); //
    produces routh table for polynomial 1+Kc*poly
84 disp(Routh2,"Routh2=")
85 Kmax2=roots(numer(Routh2(1,1)));
86
87 Routh3=routh_t(poly_PID(1,1)/1,poly(0,"K")); //
    produces routh table for polynomial 1+Kc*poly
88 disp(Routh3,"Routh3=")

```

```

89 //Kmax3=roots (numer (Routh3(1,1)));
90
91 Routh4=routh_t(poly_PID(2,1)/1,poly(0,"K")); //
    produces routh table for polynomial 1+Kc*poly
92 disp(Routh4,"Routh4=")
93 //Kmax4=roots (numer (Routh4(1,1)));
94
95 mprintf('\n Kmax should be less than %f and %f \n
    for tauc=8 and 15 respectively for PI system to
    be stable ',Kmax1,Kmax2)
96 mprintf('\n\nAnswers to Kmax for PID controller
    using \n...
97 Routh Array in the book are wrong. This can be
    easily \n...
98 checked from Routh3 and Routh4 which are displayed\n
    ')
99 mprintf('\n\nFor frequency response refer to ch-13
    for Bode Plots\n')

```

---

#### Scilab code Exa 11.4 IMC for lag dominant model

```

1
2 clear
3 clc
4
5 //Example 11.4
6 disp('Example 11.4')
7
8 s=%s;
9 theta=1;tau=100;K=100;
10 delay=(1-theta/2*s+theta^2/10*s^2-theta^3/120*s^3)

```

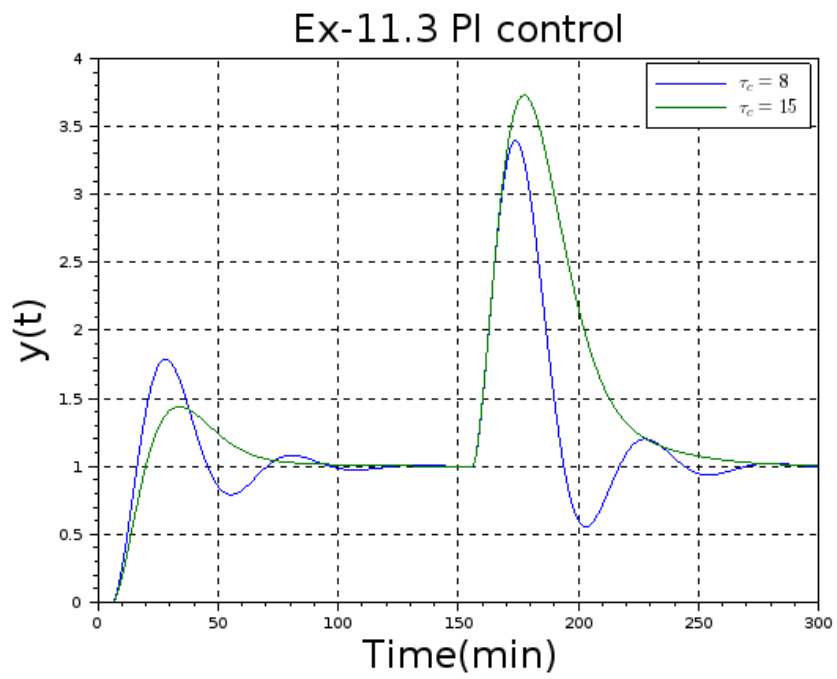


Figure 11.3: PI and PID control of liquid storage tank



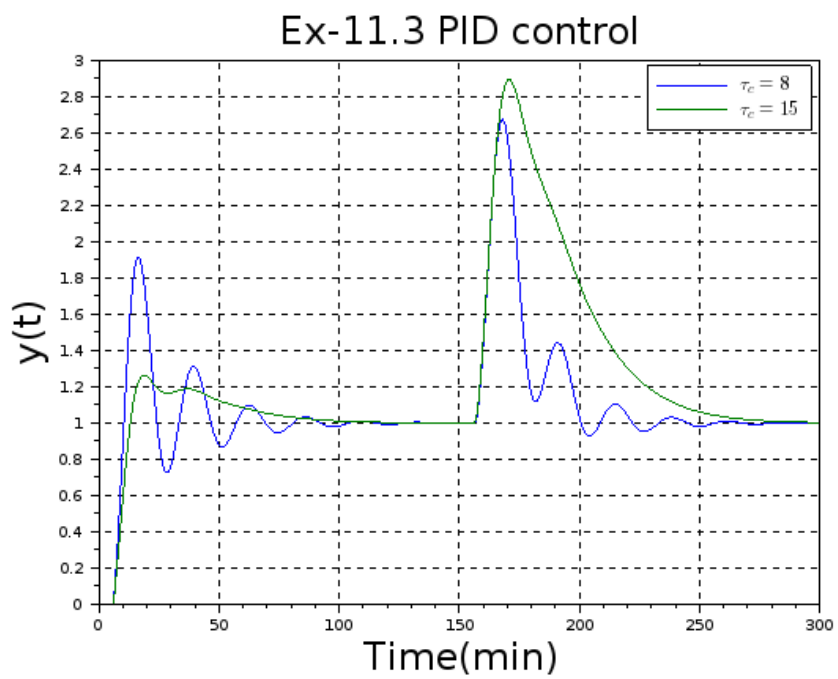


Figure 11.4: PI and PID control of liquid storage tank

```

        /(1+theta/2*s+theta^2/10*s^2+theta^3/120*s^3); //
        Third order pade approx
11 G=K*delay/(tau*s+1);
12
13 //(a)
14 tauca=1;
15 Kc1=1/K*tau/(tauca+theta);tau1=tau;
16 //(b)
17 taucb=2;Kstar=K/tau;
18 Kc2=1/Kstar*(2*taucb+theta)./(taucb+theta).^2; //Row
        M
19 tau12=2*taucb+theta;
20 //(c)
21 taucc=1;
22 Kc3=Kc1;tau13=min(tau1,4*(taucc+theta))
23 //(d)
24 //Kc4=0.551;tau14=4.91;
25 //Chen and Seborg settings given in Second Edition
        of book
26
27 mprintf('                Kc                tau1')
28 mprintf('\nIMC    %20f    %f',Kc1,tau1)
29 mprintf('\nIntegrator approx    %-5f    %f',Kc2,tau12
        )
30 mprintf('\nSkogestad    %15f    %f',Kc3,tau13)
31 //mprintf('\nDS-d    %20f    %f',Kc4,tau14)
32
33
34
35 Gc=[Kc1 Kc2 Kc3]'.*(1+(1)./([tau1 tau12 tau13]'*s))
36
37 G_CL=syslin('c',Gc*G./(1+Gc*G));
38 t=0:0.1:20;
39 y=csim('step',t,G_CL);
40 y(:,1:theta/0.1)=0; //accounting for time delay—this
        is required otherwise
41 //an unrealistic inverse response is seen due to the
        pade/taylor approx

```

```

42 plot(t,y);
43 xgrid()
44 xtitle('Ex-11.4 Tracking problem','Time(min)','y(t)')
    );
45 a=legend("IMC","Integrator approx","Skogestad",
    position=4);
46 a.font_size=2;
47 a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
48 c=a.y_label;c.font_size=5;
49
50 scf()
51 t=0:0.1:60;
52 G_CL_dist=syslin('c',G./(1+Gc*G));//closed loop wrt
    disturbance
53 yd=csim('step',t,G_CL_dist);
54 yd(:,1:theta/0.1)=0;//accounting for time delay—
    this is required otherwise
55 //an unrealistic inverse response is seen due to the
    pade/taylor approx
56 plot(t,yd);
57
58 xgrid()
59 xtitle('Ex-11.4 Disturbance rejection','Time(min)','y(t)');
60 a=legend("IMC","Integrator approx","Skogestad",
    position=4);
61 a.font_size=2;
62 a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
63 c=a.y_label;c.font_size=5;

```

---

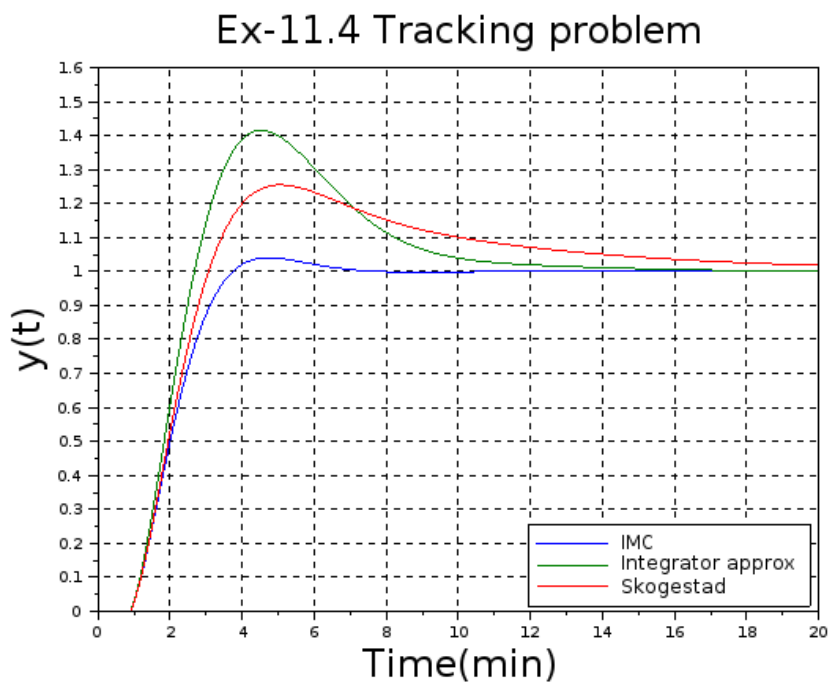


Figure 11.5: IMC for lag dominant model

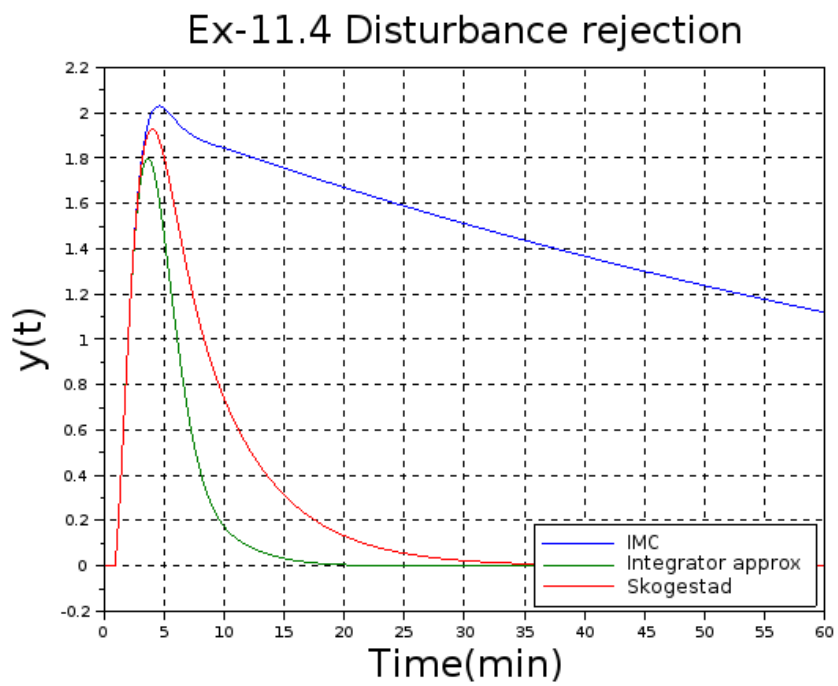


Figure 11.6: IMC for lag dominant model

### Scilab code Exa 11.5 PI controller IMC ITAE

```
1 clear
2 clc
3
4 //Example 11.5
5 disp('Example 11.5')
6
7 K=1.54;theta=1.07;tau=5.93;
8
9
10 //(a)
11 tauca=tau/3;
12 Kc1=1/K*tau/(tauca+theta);taui1=tau; //Table 11.1
13 //(b)
14 taucb=theta;
15 Kc2=1/K*tau/(taucb+theta);taui2=tau; //Table 11.1
16 //(c)
17 //Table 11.3
18 Y=0.859*(theta/tau)^(-0.977);Kc3=Y/K;
19 taui3=tau*inv(0.674*(theta/tau)^-0.680);
20 //(d)
21 //Table 11.3
22 Kc4=1/K*0.586*(theta/tau)^-0.916;taui4=tau*inv
    (1.03-0.165*(theta/tau));
23
24 mprintf('          Kc          tauI')
25 mprintf('\nIMC(tauC=tau/3)    %f    %f',Kc1,taui1)
26 mprintf('\nIMC(tauC=theta)    %-5f    %f',Kc2,taui2)
27 mprintf('\nITAE(disturbance)  %f    %f',Kc3,taui3)
28 mprintf('\nITAE(set point)   %10f    %f',Kc4,taui4)
```

---

### Scilab code Exa 11.6 Controller with two degrees of freedom

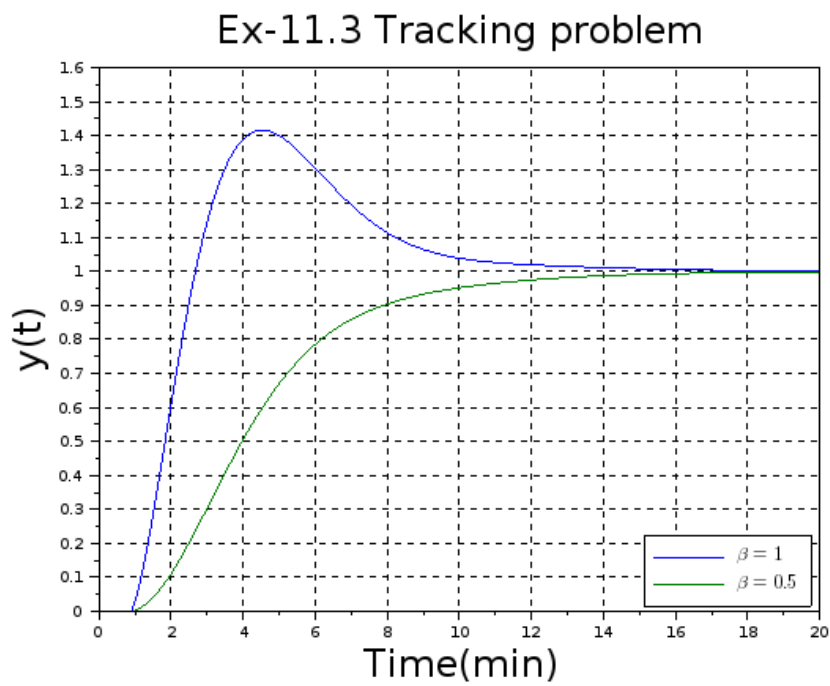


Figure 11.7: Controller with two degrees of freedom

```

1 clear
2 clc
3
4 //Example 11.6
5 disp('Example 11.6')
6
7 //Drawing on example 11.4
8 s=%s;
9 theta=1;tau=100;K=100;
10 delay=(1-theta/2*s+theta^2/10*s^2-theta^3/120*s^3)
    /(1+theta/2*s+theta^2/10*s^2+theta^3/120*s^3);//
    Third order pade approx
11 G=K*delay/(tau*s+1);
12
13 Kc=0.556;taui=5;
14
15 Gc=Kc.*(1+(1)./([taui]*s))
16 G_CL=syslin('c',Gc*G./(1+Gc*G));
17 t=0:0.1:20;
18 y1=csim('step',t,G_CL);
19 y1(:,1:theta/0.1)=0;//accounting for time delay—
    this is required otherwise
20 //an unrealistic inverse response is seen due to the
    pade/taylor approx
21
22 beta=0.5;
23 G_CL2=syslin('c',(Gc+beta-1)*G./(1+Gc*G));//This can
    be obtained on taking
24 //laplace transform of eqn 11-39 and making a block
    diagram
25 //In Eqn 11-39 p refers to input to the process
26 t=0:0.1:20;
27 y2=csim('step',t,G_CL2);
28 y2(:,1:theta/0.1)=0;//accounting for time delay—
    this is required otherwise
29 //an unrealistic inverse response is seen due to the
    pade/taylor approx
30

```



```

31 plot(t,[y1; y2]);
32 xgrid()
33 xtitle('Ex-11.3 Tracking problem', 'Time(min)', 'y(t)')
    );
34 a=legend("$\beta=1$", "$\beta=0.5$", position=4);
35 a.font_size=2;
36 a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
37 c=a.y_label;c.font_size=5;
38
39 //Note that there is a slight mis-match between the
    plots obtained from scilab code
40 //and that of the book because of third order pade
    approximation
41 //The plots in the book have been produced using
    advanced proprietary software
42 //which supports using exact delays while scilab
    does not have that functionality

```

---

#### Scilab code Exa 11.7 Continuous cycling method

```

1 clear
2 clc
3
4 //Example 11.7
5 disp('Example 11.7')
6
7 K=2; theta=1; tau1=10; tau2=5; //Model parameters
8
9 s=%s;
10 delay=(1-theta/2*s+theta^2/10*s^2-theta^3/120*s^3)
    /(1+theta/2*s+theta^2/10*s^2+theta^3/120*s^3); //
    Third order pade approx
11 G=K*delay/((tau1*s+1)*(tau2*s+1));
12 Ku=[8.01]'; //Trials for various values of Ku can be

```

```

done by changing Ku
13 G_CL_trial=syslin('c',G*Ku./(1+G*Ku))
14 t=0:0.1:100;
15 y=csim('step',t,G_CL_trial);
16 plot(t,y)
17 xtitle('Ex-11.7 Finding ultimate gain Ku','Time(min)
    ', 'y(t)');
18 a=legend("Closed loop test",position=4);
19 a.font_size=2;
20 a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
21 c=a.y_label;c.font_size=5;
22 //There isnot a sustained oscillation for Ku=7.88,
    in our simulation because
23 //we are using a third order Pade Approx for delay.
    But still we go ahead with it
24 //so that it matches with the example values. Our
    simulations give Ku=8
25 Ku=7.88;Pu=11.66;
26
27
28 //(a) Table 11.4 ZN
29 Kc1=0.6*Ku;taui1=Pu/2;tauD1=Pu/8;
30 //(b) Table 11.4 TL
31 Kc2=0.45*Ku;taui2=Pu*2.2;tauD2=Pu/6.3;
32 //(c) DS method
33 tauc=3;
34 Kc3=1/K*(tau1+tau2)/(tauc+theta);taui3=tau1+tau2;
    tauD3=tau1*tau2/(tau1+tau2);
35
36 mprintf('          Kc          tauI          tauD')
37 mprintf('\nZN    %f    %f    %f',Kc1,taui1,tauD1)
38 mprintf('\nTL    %f    %f    %f',Kc2,taui2,tauD2)
39 mprintf('\nDS    %f    %f    %f',Kc3,taui3,tauD3)
40
41
42 // Now we compare the performance of each controller
    setting

```

```

43 Gc1=Kc1.*(1+(1)./taui1/s+tauD1*s)
44 Gc2=Kc2.*(1+(1)./taui2/s+tauD2*s)
45 Gc3=Kc3.*(1+(1)./taui3/s+tauD3*s)
46 Gc=[Gc1 Gc2 Gc3]';
47 G_CL=syslin('c',Gc*G./(1+Gc*G));
48 t=0:160;
49 y=csim('step',t,G_CL);
50 y(:,1:theta)=0;//accounting for time delay—this is
    required otherwise
51 //an unrealistic inverse response is seen due to the
    pade/taylor approx
52
53
54 t_d=81:160;
55 G_CL_dist=syslin('c',G./(1+Gc*G));//closed loop wrt
    disturbance
56 yd=csim('step',t_d,G_CL_dist);
57 yd(:,1:theta)=0;//accounting for time delay
58 y(:,t_d)=y(:,t_d)+yd;
59
60 scf();
61 plot(t,y)
62 xgrid()
63 xtitle('Ex-11.7 Comparison of 3 controllers','Time(
    min)','y(t)');
64 a=legend("Ziegler-Nichols","Tyerus-Luyben","Direct
    Synthesis",position=4);
65 a.font_size=2;
66 a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
67 c=a.y_label;c.font_size=5;
68
69 mprintf('\n\nThere is slight mismatch between scilab
    simulation\n...
70 and book simulation due to Pade approx\n')

```

---

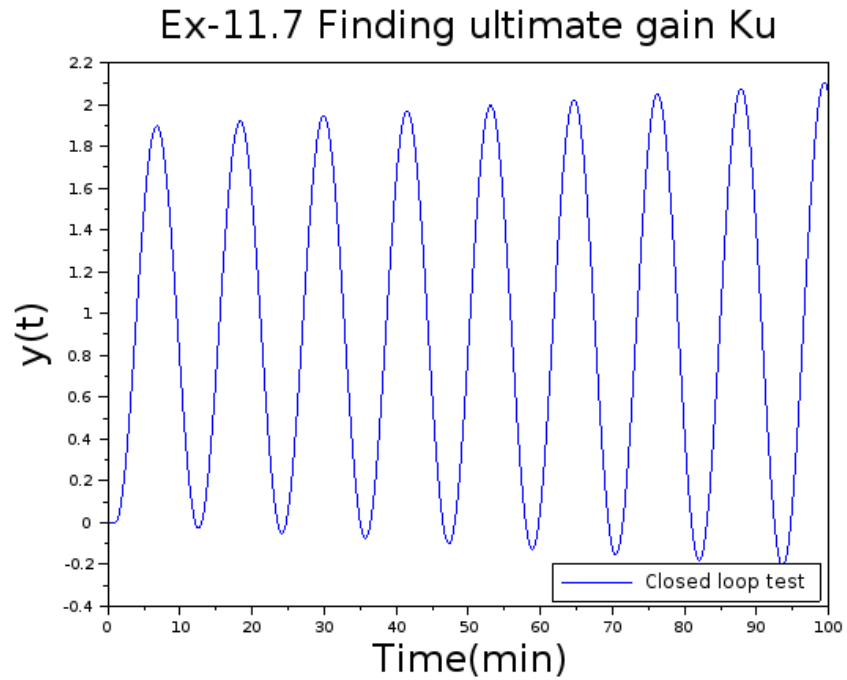


Figure 11.8: Continuous cycling method

Scilab code Exa 11.8 Reaction curve method

```

1 clear
2 clc
3
4 //Example 11.8
5 disp('Example 11.8 ')
6

```

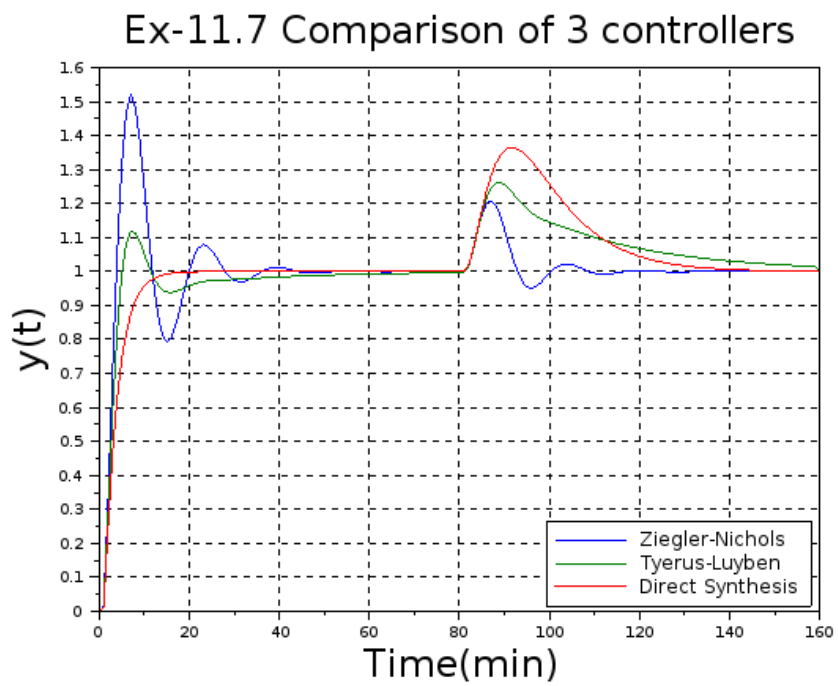


Figure 11.9: Continuous cycling method

```
7 S=(55-35)/(7-1.07); //%/min
8 delta_p=43-30; //%
9 R=S/delta_p; //min^-1
10
11 delta_x=55-35; //%
12 K=delta_x/delta_p;
13 theta=1.07; //min
14 tau=7-theta; //min
15
16 mprintf("\nThe resulting process model is with delay
          of 1.07 min\n")
17 s=%s;
18 G=K/(tau*s+1);
19 disp(G, 'G=')
```

---

# Chapter 12

## Control Strategies at the Process Unit Level

Scilab code Exa 12.1 Degrees of freedom

```
1 clear
2 clc
3
4 //Example 12.1
5 disp('Example 12.1 ')
6
7 NE=3;
8 NV=6;
9 NF=NV-NE;
10 ND=2;
11 NFC=NF-ND;
12 mprintf("    NF=%i\n    NFC=%i",NF,NFC)
```

---

# Chapter 13

## Frequency response analysis and control system design

Scilab code Exa 13.3 Bode Plot

```
1 clear
2 clc
3
4 //Example 13.3
5 disp('Example 13.3 ')
6
7 function bodegen(num,den,w,lf,delay)
8 //Bode plot
9 //Numerator and denominator are passed as input
   arguments
10 //Both are polynomials in powers of s(say)
11
12 //This function has been modified from the original
   one
13 //written by Prof Kannan Moudgalya, Dept of ChemE,
   IIT-B
14 G = num/den;
```



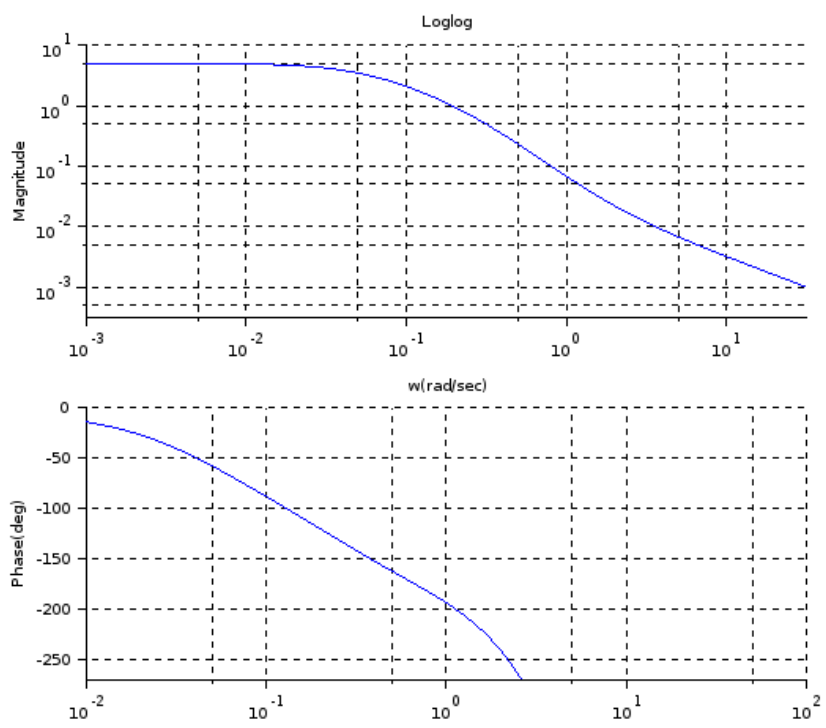


Figure 13.1: Bode Plot

```

15 G1 = horner(G,%i*w);
16 G1p = phasemag(G1)-delay*w*180/%pi;
17
18 if LF == "normal" then
19     xset('window',0); clf();
20     subplot(2,1,1)
21     plot2d(w,abs(G1),logflag="nn",style = 2);
22     xtitle('Normal scale',' ','Magnitude'); xgrid();
23     subplot(2,1,2)
24     plot2d1(w,G1p,logflag="nn",style = 2);
25     xgrid();
26     xtitle('w(rad/sec)',' ','Phase(deg)');
27 elseif LF == "semilog" then
28     xset('window',1); clf();
29     subplot(2,1,1)
30     plot2d(w,20*log10(abs(G1)),logflag="ln",style =
31         2);
32     xgrid();
33     xtitle('Semilog',' ','Magnitude (dB)');
34     subplot(2,1,2)
35     plot2d1(w,G1p,logflag="ln",style = 2);
36     xgrid();
37     xtitle('w(rad/sec)',' ','Phase(deg)');
38 elseif LF == "loglog" then
39     xset('window',2); clf();
40     subplot(2,1,1)
41     plot2d(w,abs(G1),logflag="ll",style = 2);
42     xgrid();
43     xtitle('Loglog',' ','Magnitude');
44     subplot(2,1,2)
45     plot2d1(w,G1p,logflag="ln",style = 2,rect
46         =[0.01,-270,100,0]); //note the usage of rect
47         for this particular example
48     xgrid();
49     xtitle('w(rad/sec)',' ','Phase(deg)');
50 end
51 endfunction;
52

```

```

50
51 s = %s;
52 num = 5*(0.5*s+1);
53 den = (20*s+1)*(4*s+1);
54 theta=1;
55
56 w = 0.001:0.002:10*%pi;
57 LF = "loglog" // Warning: Change this as necessary
58
59 bodegen(num,den,w,LF,theta);
60
61 //Checking using iodelay toolbox in scilab
62 //G=syslin('c',num/den);
63 //G=iodelay(G,1)
64 //bode(G,0.01,0.1)

```

---

#### Scilab code Exa 13.4 Bode

```

1 clear
2 clc
3
4 //Example 13.4
5 disp('Example 13.4')
6
7
8 s = %s;
9 num = 2;
10 den = (0.5*s+1)^3;
11 delay=0;
12 w = 0.001:0.002:100*%pi;
13 LF = "loglog" // Warning: Change this as necessary
14
15

```

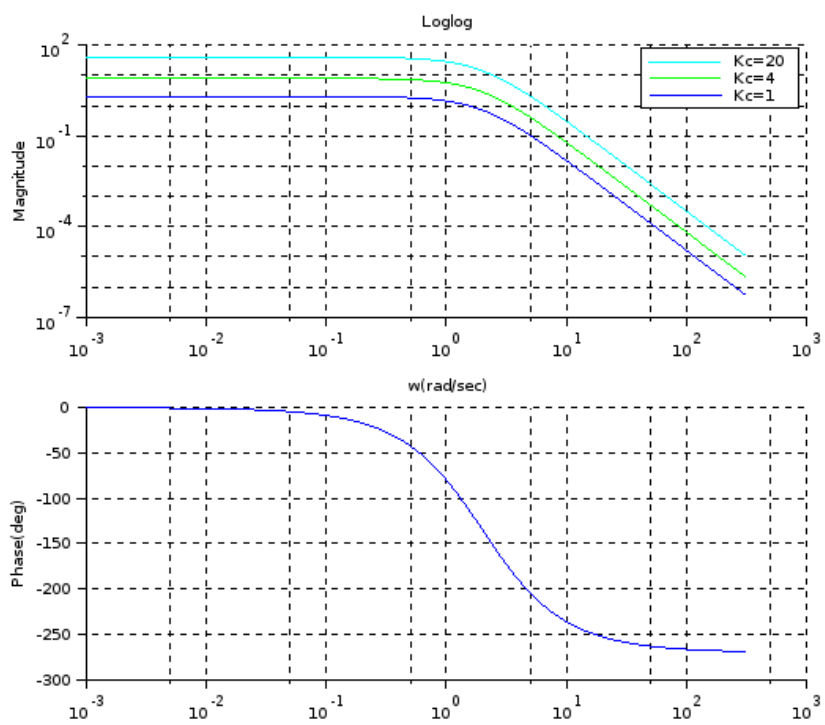


Figure 13.2: Bode

```

16
17 //Kc=1
18 G1 = num/den;
19 G1m = horner(G1,%i*w); //G1m denotes magnitude
20 G1p = phasemag(G1m)-delay*w*180/%pi; //G1p denotes
    phase
21
22 //Kc=4
23 G2 = 4*num/den;
24 G2m = horner(G2,%i*w);
25 G2p = phasemag(G2m)-delay*w*180/%pi;
26
27 //Kc=20
28 G3 = 20*num/den;
29 G3m = horner(G3,%i*w);
30 G3p = phasemag(G3m)-delay*w*180/%pi;
31
32 xset('window',0);
33 subplot(2,1,1)
34 plot2d(w,abs(G3m),logflag="ll",style = 4);
35 plot2d(w,abs(G2m),logflag="ll",style = 3);
36 plot2d(w,abs(G1m),logflag="ll",style = 2);
37
38 xgrid();
39 xtitle('Loglog',' ',' Magnitude');
40 legend("Kc=20","Kc=4","Kc=1")
41 subplot(2,1,2)
42 plot2d1(w,G1p,logflag="ln",style = 2);
43 xgrid();
44 xtitle('w(rad/sec)',' ',' Phase(deg)');

```

---

Scilab code Exa 13.5 PI control of overdamped second order process

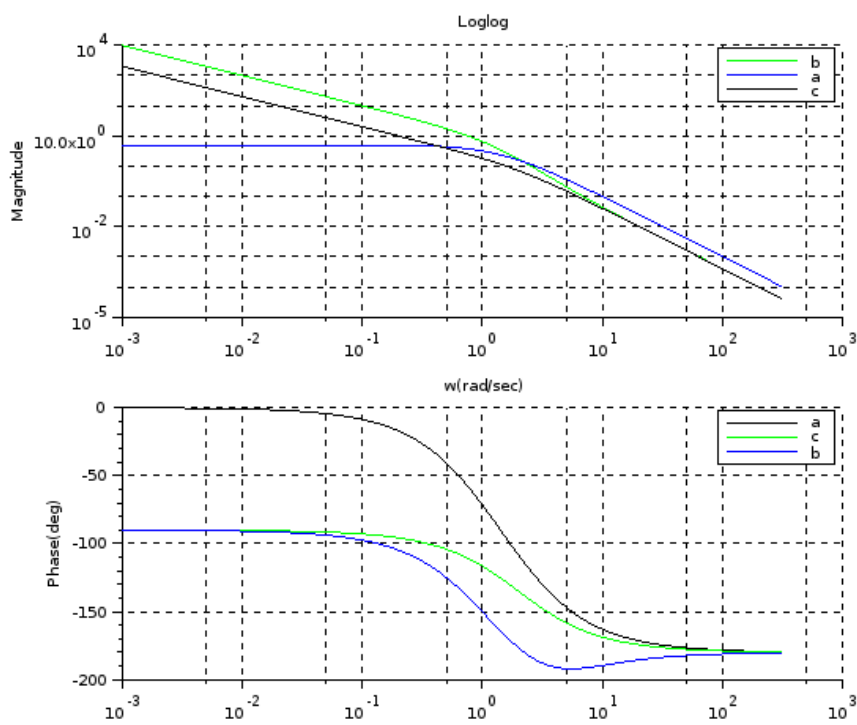


Figure 13.3: PI control of overdamped second order process

```

1 clear
2 clc
3
4 //Example 14.5
5 disp('Example 14.5')
6
7
8 s = %s;
9 num = 1;
10 den = (9*s+1)*(11*s+1);
11 delay=0.3;
12 w = 0.001:0.002:5*%pi;
13 LF = "loglog" // Warning: Change this as necessary
14
15 Gc=20*(1+1/2.5/s+s);
16 G1 = num/den*Gc;
17 G1m = horner(G1,%i*w); //G1m denotes magnitude
18 G1p = phasemag(G1m)-delay*w*180/%pi; //G1p denotes
    phase
19
20 xset('window',0);
21 subplot(2,1,1)
22 plot2d(w,abs(G1m),logflag="ll",style = 3);
23 xgrid();
24 xtitle('Loglog', '', 'Magnitude');
25 subplot(2,1,2)
26 plot2d1(w,G1p,logflag="ln",style = 1);
27 xgrid();
28 xtitle('w(rad/sec)', '', 'Phase(deg)');

```

---

### Scilab code Exa 13.6 Bode plot

```

1 clear

```

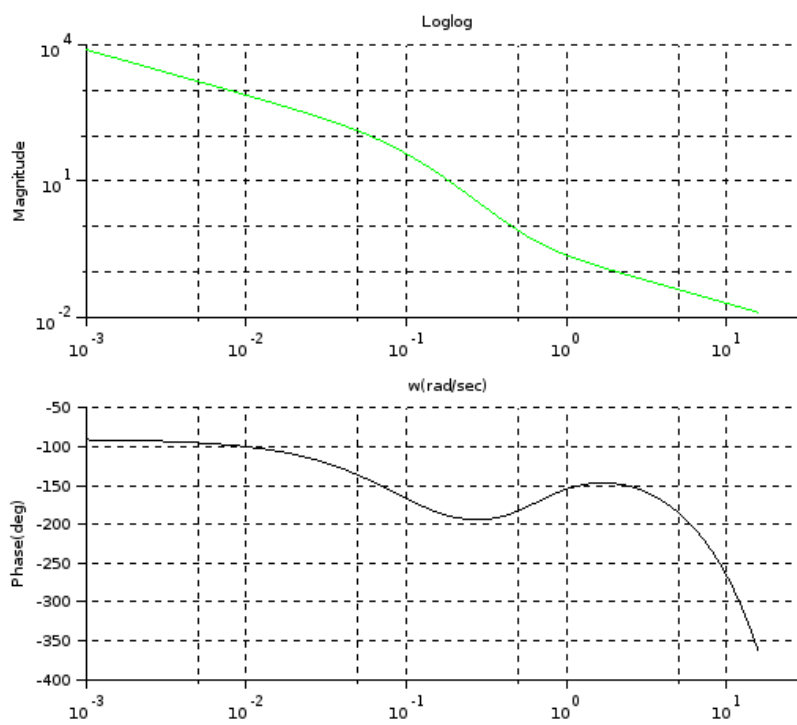


Figure 13.4: Bode plot



```

2  clc
3
4  //Example 13.6
5  disp('Example 13.6')
6
7
8  s = %s;
9  num = 1;
10 den = (9*s+1)*(11*s+1);
11 delay=0.3;
12 w = 0.001:0.002:5*%pi;
13 LF = "loglog" // Warning: Change this as necessary
14
15 Gc=20*(1+1/2.5/s+s);
16 G1 = num/den*Gc;
17 G1m = horner(G1,%i*w); //G1m denotes magnitude
18 G1p = phasemag(G1m)-delay*w*180/%pi; //G1p denotes
    phase
19
20  xset('window',0);
21  subplot(2,1,1)
22  plot2d(w,abs(G1m),logflag="ll",style = 3);
23  xgrid();
24  xtitle('Loglog',' ',' Magnitude');
25  subplot(2,1,2)
26  plot2d1(w,G1p,logflag="ln",style = 1);
27  xgrid();
28  xtitle('w(rad/sec)', ' ',' Phase(deg)');

```

---

### Scilab code Exa 13.7 Bode Plot

```

1  clear
2  clc

```

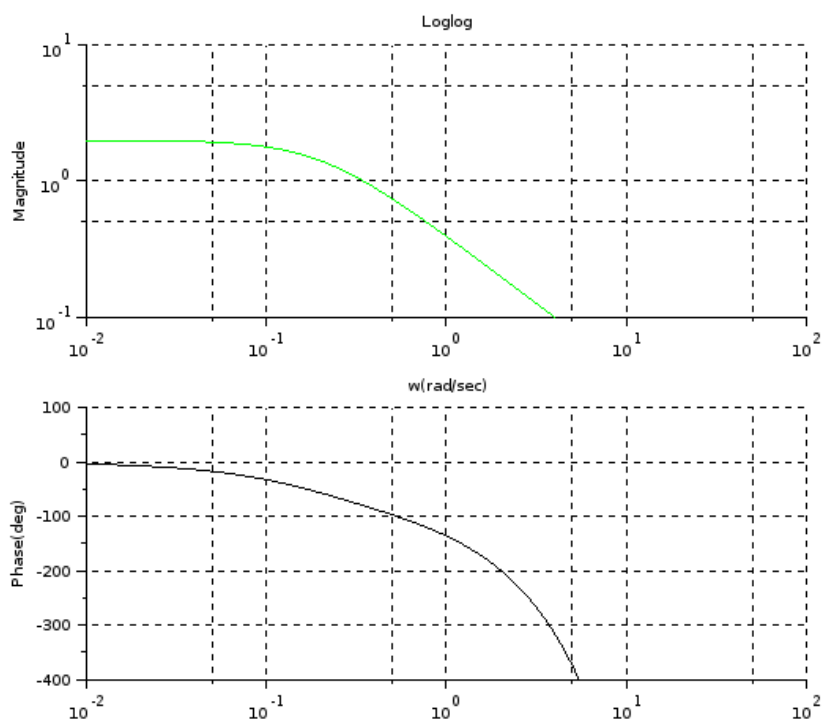


Figure 13.5: Bode Plot

```

3
4 //Example 13.7
5 disp('Example 13.7')
6
7
8 s = %s;
9 num = 4;
10 den = (5*s+1);
11 delay=1;
12 w = 0.001:0.002:10*%pi;
13 LF = "loglog" // Warning: Change this as necessary
14
15 Gv=2; Gm=0.25; Gc=1;
16 G1 = num/den*Gc*Gm*Gv;
17 G1m = horner(G1,%i*w); //G1m denotes magnitude
18 G1p = phasemag(G1m)-delay*w*180/%pi; //G1p denotes
    phase
19
20 xset('window',0);
21 subplot(2,1,1)
22 plot2d(w,abs(G1m),logflag="ll",style = 3,rect
    =[0.01,0.1,100,10]);
23 xgrid();
24 xtitle('Loglog',' ','Magnitude');
25 subplot(2,1,2)
26 plot2d1(w,G1p,logflag="ln",style = 1,rect
    =[0.01,-400,100,100]);
27 xgrid();
28 xtitle('w(rad/sec)',' ','Phase(deg)');
29
30 //Example ends
31
32 //
    *****

33 //In the SECOND EDITION of the book, this example
    also asks for drawing Nyquist plot
34 //In case you want to learn how to do it, Uncomment

```

```

    the code below
35
36  ////Please install IODELAY toolbox from Modeling and
    Control tools in ATOMS
37  ////http://atoms.scilab.org/toolboxes/iodelay/0.4.5
38  ////There is no inbuilt toolbox in scilab for
    introducing time delays other than
39  ////above mentioned. The output of iodelay toolbox
    however does not work
40  ////with csim and syslin commands
41  ////The output of iodelay however can be used for
    frequency related analyses
42  ////like bode and nyquist
43
44  //xset('window',1);
45  //G=syslin('c',G1);
46  //G=iodelay(G,delay);
47  //nyquist(G,%f); // %f => asymmetric, see help
    nyquist
48  //
    //*****

```

---

**Scilab code Exa 13.8** Maximum permissible time delay for stability

```

1  clear
2  clc
3
4  //Example 13.8
5  disp('Example 13.8')
6
7
8  s = %s;

```

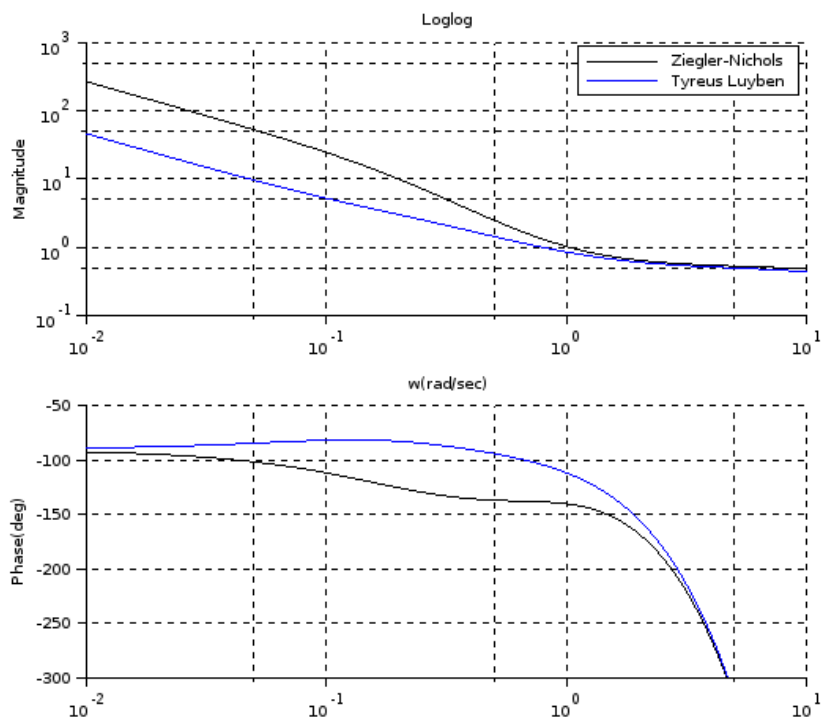


Figure 13.6: Maximum permissible time delay for stability

```

9 num = 4;
10 den = (5*s+1);
11 delay=1;
12 w = 0.001:0.002:10*%pi;
13 LF = "loglog" // Warning: Change this as necessary
14
15 Gv=2; Gm=0.25;
16
17 Ku=4.25; Pu=2*%pi/1.69;
18
19 //Ziegler Nichols
20 Kc1=0.6*Ku; tauI1=Pu/2; tauD1=Pu/8;
21 //Tyreus Luyben
22 Kc2=0.45*Ku; tauI2=Pu*2.2; tauD2=Pu/6.3;
23
24 mprintf('          Kc          tauI          tauD')
25 mprintf('\nZN   %f   %f   %f',Kc1,tauI1,tauD1)
26 mprintf('\nTL   %f   %f   %f',Kc2,tauI2,tauD2)
27
28 Gc_ZN=Kc1*(1+1/tauI1/s+s*tauD1/(0.1*s*tauD1+1));
29 Gc_TL=Kc2*(1+1/tauI2/s+s*tauD2/(0.1*s*tauD2+1)); //
    Filtered Controllers with filter constant as 0.1
30
31 G1 = num/den*Gc_ZN*Gm*Gv;
32 G1m = horner(G1,%i*w); //G1m denotes magnitude
33 Abs_G1m=abs(G1m)
34 G1p = phasemag(G1m)-delay*w*180/%pi; //G1p denotes
    phase
35
36
37 G2 = num/den*Gc_TL*Gm*Gv;
38 G2m = horner(G2,%i*w); //G2m denotes magnitude
39 Abs_G2m=abs(G2m);
40 G2p = phasemag(G2m)-delay*w*180/%pi; //G2p denotes
    phase
41
42 xset('window',0);
43 subplot(2,1,1)

```

```

44     plot2d(w,Abs_G1m,logflag="ll",style = 1,rect
        =[0.01,0.1,10,1000]);
45     plot2d(w,Abs_G2m,logflag="ll",style = 2,rect
        =[0.01,0.1,10,1000]);
46     legend("Ziegler-Nichols","Tyreus Luyben")
47     xgrid();
48     xtitle('Loglog',' ','Magnitude');
49     subplot(2,1,2)
50     plot2d1(w,G1p,logflag="ln",style = 1,rect
        =[0.01,-300,10,-50]);
51     plot2d1(w,G2p,logflag="ln",style = 2,rect
        =[0.01,-300,10,-50]);
52 //     legend("Ziegler-Nichols","Tyreus Luyben")
53     xgrid();
54     xtitle('w(rad/sec)',' ','Phase(deg)');
55
56 //G_ZN=syslin('c',G1);
57 //G_ZN=iodelay(G_ZN,delay);
58 //G_TS=syslin('c',G2);
59 //G_TS=iodelay(G_TS,delay);
60 //scf();nyquist(G_TS,%f)
61 // [gm_ZN,fr_ZN]=g_margin(G_ZN);[gm_TS,fr_TS]=
    g_margin(G_TS);
62 // [pm_ZN,fr_ZN_p]=p_margin(G_ZN);[pm_TS,fr_TS_p]=
    p_margin(G_TS);
63 //g_maring and p_margin do not support iodelay
    toolbox, hence we
64 //cannot use these so we try a workaround
65
66 //We can find w for which magnitude(AR) is 1 and
67 //and calculate phase corresponding to it which
    gives us phase margin
68 //Also we can find crossover frequency and thus find
    Gain Margin
69
70     indices1=find(abs(Abs_G1m-1)<0.01) //We find
        those values of indices of Abs_G1m for which
        it is almost 1

```

```

71     indices2=find(abs(Abs_G2m-1)<0.01)
72     //size(indices)
73     PM1=mean(G1p(indices1))+180
74     PM2=mean(G2p(indices2))+180
75
76     indices1_p=find(abs(G1p+180)<0.05) //We find
       those values of indices of G1p for which it
       is almost -180
77     indices2_p=find(abs(G2p+180)<0.05)
78     //size(indices)
79     GM1=1/mean(Abs_G1m(indices1_p))
80     GM2=1/mean(Abs_G2m(indices2_p))
81
82     wc1=mean(w(indices1_p));
83     wc2=mean(w(indices2_p));
84
85
86     mprintf( '\n\n\n          GM          PM          wc ')
87     mprintf( '\nZN    %f    %f    %f', GM1, PM1, wc1)
88     mprintf( '\nTL    %f    %f    %f\n', GM2, PM2, wc2)
89
90     theta=PM2*%pi/0.79/180;
91     disp(theta, 'deltatheta (min)=')

```

---



# Chapter 14

## Feedforward and Ratio Control

Scilab code Exa 14.1 Ratio control

```
1 clear
2 clc
3
4 //Example 14.1
5 disp('Example 14.1 ')
6
7 Sd=30;
8 Su=15;
9 Rd=1/3;
10 K_R=Rd*Sd/Su; //Eqn 14-3
11 mprintf("    K_R=%f" ,K_R)
```

---

Scilab code Exa 14.5 Feedforward control in blending process

```
1 clear
2 clc
3
```

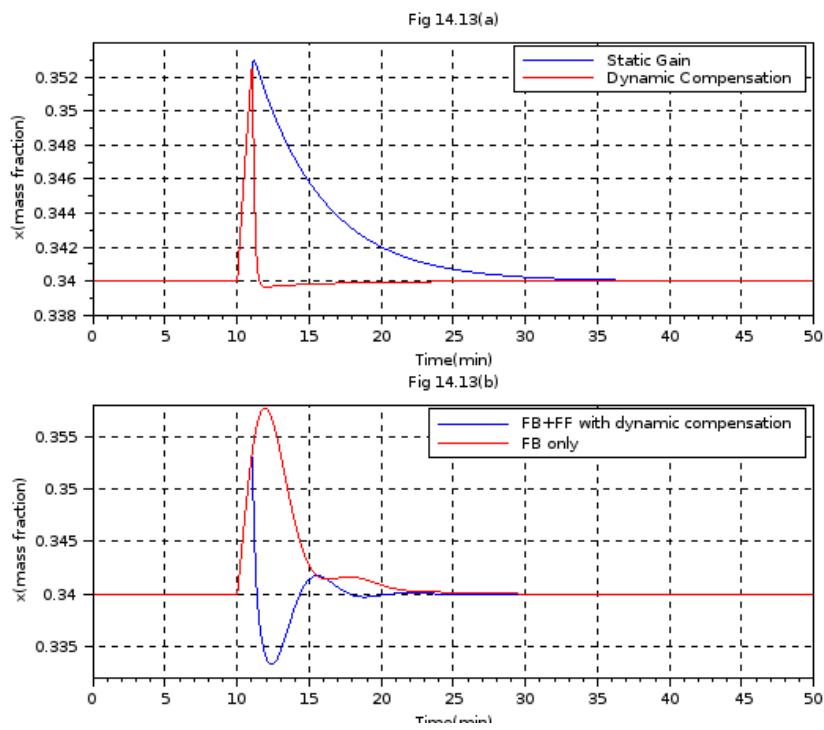


Figure 14.1: Feedforward control in blending process

```

4 //Example 14.5
5 disp('Example 14.5')
6
7 mprintf('\n\nThere are many errors in this example\n
...
8 1.In Eqn 14-17 the value of w2_o is not zero. It is
50kg/min.\n...
9 This is so because otherwise current signal from p(t
) ie ...
10 \n eqn 14-30 is more than 20mA which is not possible
\n\n....
11 2.The step change in x1 is from 0.2 to 0.3 and not
0.2 to 0.4\n...
12 If there is a step change to x1=0.4, then with x2
=0.6\n...
13 one cannot achieve output xsp=0.34 because it is
less\n...
14 both x1 and x2.\n\n...
15 3.The gain of Gd is 0.65 which is correct because V\
n...
16 has to be calculated using height=1.5meter ie\n...
17 how much the CSTR is filled and not h=3m, ie\n...
18 the capacity of CSTR. This is important because\n...
19 the person who has made solutions for the book has
taken h=3m\n...
20 for generating graphs and hence the gain is twice. \
n...
21 the graphs generated from this code are correct\n\n'
)
22
23 //part(a) //=====Static feedforward controller
=====//
24 K_IP=(15-3)/(20-4);
25 Kv=300/12;tauV=0.0833;
26 Kt=(20-4)/0.5;
27 w2_o=50;x1_o=0;//Zero of the instrument
28 w1bar=650;w2bar=350;//kg/min
29 C1=4-w2_o/Kv/K_IP; //Eqn 14-16 to 14-19

```

```

30 C2=w1bar/(Kv*K_IP*Kt);
31 C3=4+Kt*x1_o;
32 x1bar=0.2;x2bar=0.6;xbar=0.34;
33
34 mprintf('\nThe values of C1, C2, C3 in Eqns 14-16 to
          14-19 are\n %f,          %f,          %f',C1,C2,C3)
35
36 //part(b) //=====Dynamic feedforward controller
          =====//
37 s=%s;
38 theta=1;
39 V=%pi*1^2*1.5; //pi*r^2*h finding volume
40 rho=1000; //kg/m3
41 wbar=w1bar+w2bar;
42 tauD=V*rho/w2bar;tauP=V*rho/wbar;
43 Kp=(x2bar-xbar)/wbar;
44 Kd=w1bar/wbar;
45
46 Gv=Kv/(tauV*s+1);
47 Gd=Kd/(tauP*s+1);
48 Gt=Kt;delay=1;
49 Gp=Kp/(tauP*s+1);
50 Gf=-Gd/Gv/Gt/Gp/K_IP; //Equation 14-33 without exp(+
          s)
51 //Gt=32*(1-theta/2*s+theta^2/12*s^2)/(1+theta/2*s+
          theta^2/12*s^2);//second order Pade approx.
52 Gt=32*(1-theta/2*s)/(1+theta/2*s);//first order Pade
          approx.
53 alpha=0.1;
54 Gf=horner(Gf,0)*(1.0833*s+1)/(alpha*1.0833*s+1);//
          Eqn 14-34
55 disp(Gf,"Gf=")
56
57
58 //=====Static feedforward controller simulation
          =====//
59 Ts=0.01; //sampling time in minutes
60 t=Ts:Ts:50;

```

```

61 xsp=0.34; //set point for conc. output of blender
62 x1step=0.2+[zeros(1,length(t)/5) 0.1*ones(1,length(t)
    )*4/5)]; //disturbance
63 //There is a one second lag in the measurement of
    the disturbance by Gt
64
65 delay=1;
66 d=[0.2*ones(1,delay/Ts) x1step(1,1:$-delay/Ts)]; //
    measurement of disturbance with lag
67 x1m=4+Kt*d; //Eqn 14-12 where d=x1(t)-x1_o
68
69 //plot(t,[x1step' x1m'])
70 pt=C1+C2*(Kt*xsp-x1m+C3)/(x2bar-xsp);
71 //Now the values calculated by the above controller
    needs to be passed to the process
72 G_static=syslin('c',[Gd K_IP*Gv*Gp]);
73 //we take disturbance and control action in
    deviation variables
74 yt=0.34+csim([x1step-x1step(1,1);pt-pt(1,1)],t,
    G_static);
75 subplot(2,1,1)
76 plot(t,yt);
77 xtitle(" Fig 14.13(a)", "Time(min)", "x(mass fraction)"
    )
78 xgrid();
79
80 //=====Dynamic feedforward controller simulation
    =====//
81
82 Ys_Ds=[Gd K_IP*32*Gf*Gv*Gp]; //Gt=32 without delay
83 Ys_Ds=syslin('c',Ys_Ds);
84 t=0.01:0.01:50;
85 d=[zeros(1,length(t)/5) 0.1*ones(1,length(t)*4/5)];
    //disturbance
86 d_shift=[zeros(1,1.1*length(t)/5) 0.1*ones(1,length(
    t)*3.9/5)];
87 //we delay the disturbance by one minute for the
    feedforward controller

```

```

88 //We do this because Pade approx is not good for
    delay of 1 minute
89 yt=0.34+csim([d;d_shift],t,Ys_Ds)
90 plot(t,yt,color='red')
91 legend("Static Gain","Dynamic Compensation")
92
93 //part(c) //=====PI controller for Feedback
    =====//
94 Kcu=48.7;Pu=4;//min
95 Kc=0.45*Kcu;tauI=Pu/1.2;tauD=0;
96 Gc=Kc*(1+1/(tauI*s)+tauD*s/(1+tauD*s*0.1));
97 Gm=Gt;//sensor/transmitter
98
99
100 //=====Feedforward and feedback control with
    dynamic compensation=====//
101 Ys_Ds=[Gd K_IP*32*Gf*Gv*Gp]/(1+K_IP*Gc*Gv*Gp*Gm);//
    32 is magnitude of Gt
102 Ys_Ds=syslin('c',Ys_Ds);
103 t=0.01:0.01:50;
104 d=[zeros(1,length(t)/5) 0.1*ones(1,length(t)*4/5)];
    //disturbance
105 yt=0.34+csim([d;d_shift],t,Ys_Ds)
106 //This shifting is better because Pade approx is not
    accurate. Note that there is
107 //pade approx in the denominator also(Gm) which we
    cant help.
108 subplot(2,1,2)
109 plot(t,yt)
110 xgrid();
111 xtitle("Fig 14.13(b)","Time(min)","x(mass fraction)"
    )
112
113 //=====Feedback control only with dynamic
    compensation=====//
114 Ys_Ds=(Gd)/(1+K_IP*Gc*Gv*Gp*Gm);
115 Ys_Ds=syslin('c',Ys_Ds);
116 d=[zeros(1,length(t)/5) 0.1*ones(1,length(t)*4/5)];

```

```
    //disturbance
117 yt=0.34+csim(d,t,Ys_Ds)
118 plot(t,yt,color='red')
119 legend("FB+FF with dynamic compensation","FB only")
120
121 mprintf("\n\nNote the slight mismatch between
    response \n...
122 times due to pade approx the gain is half of that in
    the\n...
123 book. Please see the heigh explanation above to
    understand.")
```

---

# Chapter 15

## Enhanced Single Loop Control Strategies

Scilab code Exa 15.1 Stability limits for proportional cascade controller

```
1 clear
2 clc
3
4 //Example 15.1
5 disp('Example 15.1 ')
6
7
8 s=%s;
9 Gp1=4/((4*s+1)*(2*s+1));Gp2=1;Gd2=1;Gd1=1/(3*s+1);
10 Gm1=0.05;Gm2=0.2;
11 Gv=5/(s+1);
12 Kc2=4;
13 Ys=Kc2*Gv*Gp1*Gm1/(1+Kc2*Gv*Gm2);
14
15 Routh=routh_t(Ys,poly(0,"Kc1")); // produces routh
    table for polynomial 1+Kc*Ys
16 disp(Routh)
17 K1=roots( numer(Routh(3,1)) );
18 K2=roots( numer(Routh(4,1)) );
```



```

19
20 mprintf(' \n Kc1 lies between %f and %f \n for
    cascade system to be stable ', K2,K1)
21
22 Ys2=Gv*Gp2*Gp1*Gm1;
23 Routh2=routh_t(Ys2,poly(0,"Kc1")); // produces routh
    table for polynomial 1+Kc*Ys
24 disp(Routh2)
25 K1_2=roots(numer(Routh2(3,1)));
26 K2_2=roots(numer(Routh2(4,1)));
27
28 mprintf(' \n Kc1 lies between %f and %f \n for
    conventional system to be stable ', K2_2,K1_2)
29
30
31
32 //We cannot find offset symbolically in Scilab
    because scilab does not support
33 //handling of two variables in single polynomial
34 //To find this limit one can use Sage
35 //However in this case the calculations can be done
    in a very easy way by hand
36 //and hence do not require to be computed from Sage

```

---

**Scilab code Exa 15.2** Set point response for second order transfer function

```

1 clear
2 clc
3
4 //Example 15.2
5 disp('Example 15.2 ')
6
7 s = %s;

```

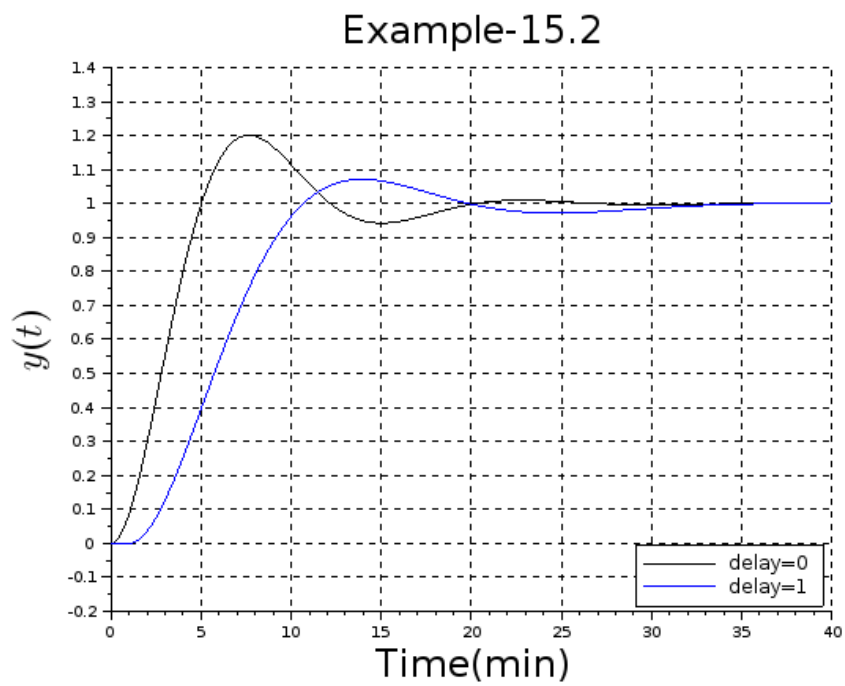


Figure 15.1: Set point response for second order transfer function

```

8 theta=1 //delay
9 delay=(1-theta/2*s+theta^2/12*s^2)/(1+theta/2*s+
    theta^2/12*s^2); //Second order pade approx
10 G=1/((5*s+1)*(3*s+1));
11 Gp=[G;delay*G]; //Both models with and without delay
12 Gc=[3.02*(1+1/(6.5*s));1.23*(1+1/(7*s))];
13 G_CL=syslin('c',(Gp.*Gc)./(1+Gp.*Gc))
14 t=0:0.01:40;
15 yt=csim('step',t,G_CL)
16
17 plot2d(t',yt') //For plotting multiple graphs in one
    command make sure time is n*1 vector
18 //while yt is n*p vector where p are the no. of
    plots
19 xtitle('Example-15.2','Time(min)','$y(t)$');
20 xgrid();
21 a=legend("delay=0","delay=1",position=4);
22 a.font_size=2;
23 a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
24 c=a.y_label;c.font_size=5;

```

---

# Chapter 16

## Multiloop and Multivariable Control

Scilab code Exa 16.1 Pilot scale distillation column

```
1 clear
2 clc
3
4 //Example 16.1
5 disp('Example 16.1 ')
6 K=[12.8 -18.9;6.6 -19.4];
7 tau=[16.7 21;10.9 14.4];
8 s=%s;
9 G=K./(1+tau*s);
10
11 //ITAE settings from Table 11.3
12 K1=12.8;tau1=16.7;theta1=1;K2=-19.4;tau2=14.4;theta2
    =3;
13 Kc1=1/K1*0.586*(theta1/tau1)^-0.916;taui1=tau1*inv
    (1.03-0.165*(theta1/tau1));
14 Kc2=1/K2*0.586*(theta2/tau2)^-0.916;taui2=tau2*inv
    (1.03-0.165*(theta2/tau2));
```

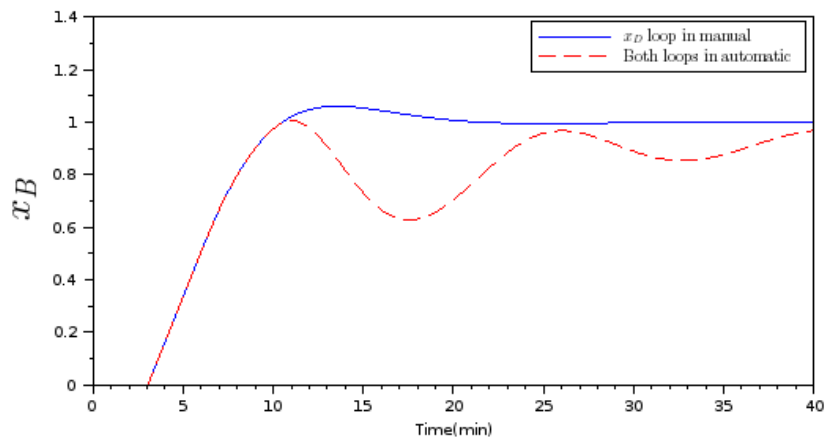
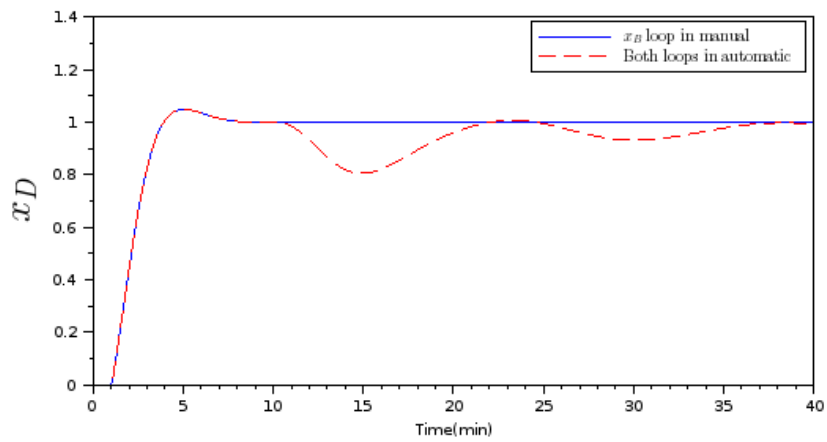


Figure 16.1: Pilot scale distillation column

```

15
16 mprintf( '          Kc          tauI ')
17 mprintf( '\nx-D-R    %f    %f',Kc1,tauI1)
18 mprintf( '\nx-B-R    %f    %f',Kc2,tauI2)
19
20 Kc=[Kc1;Kc2];
21 tauI=[tauI1;tauI2];
22
23 //====Making step response models of the continuos
    transfer functions====//
24 Ts=0.1;//Sampling time ie delta_T
25 delay3=3/Ts;
26 delay1=1/Ts;
27 delay7=7/Ts;
28 N=100/Ts;//Model Order
29 s=%s;
30 G=syslin('c',diag(matrix(G,1,4)));//Transfer
    function
31 t=0:Ts:N*Ts;
32 u_sim=ones(4,length(t));
33 //Modeling Output delays through input delay in
    steps
34 u_sim(1,1:(delay1))=zeros(1,delay1);
35 u_sim(3,1:(delay7))=zeros(1,delay7);
36 u_sim([2 4],1:(delay3))=zeros(2,delay3);
37 S=csim(u_sim,t,G)';//generating step response model
    for real plant
38 //plot(t,S);
39 S(1,:)=[];
40 //Now we have these step response models for each of
    the transfer functions
41 // [S1 S3
42 // S2 S4
43
44
45
46
47 T=120;//Simulation Run Time in minutes

```

```

48 n=T/Ts*2+1; //no. of discrete points in our domain
    of analysis
49
50 //
=====//
51 //
=====//
52 //
=====//

53 //=====Set point as +1 in X-D=X-B loop in manual
=====//
54 //p is the controller output
55 p=zeros(n,2);
56 delta_p=zeros(n,2);
57 e=zeros(n,2); //errors=(ysp-y) on which PI acts
58 ysp=zeros(n,2);
59 ysp((n-1)/2+1:n,1)=ones(n-((n-1)/2+1)+1,1);
60
61 t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
62 y=zeros(n,2);
63
64
65 for k=(n-1)/2+1:n
66
67     //Error e
68     e(k,:)=ysp(k-1,:)-y(k-1,:);
69     delta_e(k,:)=e(k,:)-e(k-1,:);
70
71     //Controller calculation — Digital PID — Eqn
        7-28 Pg 136 (Velocity form)
72     p(k,1)=p(k-1,1)+([delta_e(k,1)+e(k,1)*diag(Ts/
        taui1)]*diag(Kc1));
73     //1-1/2-2 pairing
74
75     delta_p(k,:)=p(k,:)-p(k-1,:);

```

```

76
77 //Output
78 y(k,1)=[S(1:N-1,1);S(1:N-1,3)]'. . .
79     * [flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
80         delta_p(k-N+1:k-1,2),1)]. . .
81     + [S(N,1) S(N,3)]*[p(k-N,1);p(k-N,2)];
82 y(k,2)=[S(1:N-1,2);S(1:N-1,4)]'. . .
83     * [flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
84         delta_p(k-N+1:k-1,2),1);]. . .
85     + [S(N,2) S(N,4)]*[p(k-N,1);p(k-N,2)];
86 end
87 subplot(2,1,1);
88 plot(t',y(:,1),'b-');
89 set(gca(),"data_bounds",[0 40 0 1.4]); //putting
90     bounds on display
91 l=legend("$x_B\text{ loop in manual}$",position=1);
92 xtitle("", "Time(min)", "$x_D$");
93 a=get("current_axes");
94 c=a.y_label;c.font_size=5;
95 //
96 //=====
97 //=====Set point as +1 in X-B=X-D loop in manual
98 //=====
99 //p is the controller output
100 p=zeros(n,2);
101 delta_p=zeros(n,2);
102 e=zeros(n,2); //errors=(ysp-y) on which PI acts
103 ysp=zeros(n,2);
104 ysp((n-1)/2+1:n,2)=ones(n-((n-1)/2+1)+1,1);

```



```

104
105 t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
106 y=zeros(n,2);
107
108
109 for k=(n-1)/2+1:n
110
111     //Error e
112     e(k,:)=ysp(k-1,:)-y(k-1,:);
113     delta_e(k,:)=e(k,:)-e(k-1,:);
114
115     //Controller calculation ——Digital PID——Eqn
116     //7-28 Pg 136 (Velocity form)
117     p(k,2)=p(k-1,2)+([delta_e(k,2)+e(k,2)*diag(Ts/
118     //1-1/2-2 pairing
119     //tau_i2)]*diag(Kc2));
120
121     delta_p(k,:)=p(k,:)-p(k-1,:);
122
123     //Output
124     y(k,1)=[S(1:N-1,1);S(1:N-1,3)]'...
125     * [flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
126     //delta_p(k-N+1:k-1,2),1)]...
127     +[S(N,1) S(N,3)]*[p(k-N,1);p(k-N,2)];
128     y(k,2)=[S(1:N-1,2);S(1:N-1,4)]'...
129     * [flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
130     //delta_p(k-N+1:k-1,2),1);]...
131     +[S(N,2) S(N,4)]*[p(k-N,1);p(k-N,2)];
132 end
133
134 subplot(2,1,2);
135 plot(t',y(:,2),'b-');
136 set(gca(),"data_bounds",[0 40 0 1.4]); //putting
137 //bounds on display
138 l=legend("$x_D\text{ { loop in manual} }$",position=1);
139 xtitle("", "Time(min)", "$x_B$");
140 a=get("current_axes");
141 c=a.y_label;c.font_size=5;

```

```

137
138 //
=====
139 //
=====
140 //
=====

141 //=====Set point as +1 in X-D=====Both loops
      Automatic=====//
142 //p is the controller output
143 p=zeros(n,2);
144 delta_p=zeros(n,2);
145 e=zeros(n,2); //errors=(ysp-y) on which PI acts
146 ysp=zeros(n,2);
147 ysp((n-1)/2+1:n,1)=ones(n-((n-1)/2+1)+1,1);
148
149 t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
150 y=zeros(n,2);
151
152
153 for k=(n-1)/2+1:n
154
155     //Error e
156     e(k,:)=ysp(k-1,:)-y(k-1,:);
157     delta_e(k,:)=e(k,:)-e(k-1,:);
158
159     //Controller calculation ——Digital PID——Eqn
        7-28 Pg 136 (Velocity form)
160 //     p(k,:)=p(k-1,:)+flipdim([delta_e(k,:)+e(k,:)*
diag(Ts./tauI)]*diag(Kc),2);
161     p(k,:)=p(k-1,:)+([delta_e(k,:)+e(k,:)*diag(Ts./
tauI)]*diag(Kc));
162     //1-1/2-2 pairing
163
164     delta_p(k,:)=p(k,:)-p(k-1,:);

```

```

165
166 //Output
167 y(k,1)=[S(1:N-1,1);S(1:N-1,3)]'. . .
168     * [flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
169         delta_p(k-N+1:k-1,2),1)]. . .
170     + [S(N,1) S(N,3)]*[p(k-N,1);p(k-N,2)];
171 y(k,2)=[S(1:N-1,2);S(1:N-1,4)]'. . .
172     * [flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
173         delta_p(k-N+1:k-1,2),1);]. . .
174     + [S(N,2) S(N,4)]*[p(k-N,1);p(k-N,2)];
175 end
176
177 subplot(2,1,1);
178 plot(t',y(:,1),'r—');
179 set(gca(),"data_bounds",[0 40 0 1.4]); //putting
180     bounds on display
181 l=legend("$x_B\text{ loop in manual}$","$\text{Both}$",position=1);
182     loops in automatic}$",position=1);
183 //l.font_size=5;
184 xtitle("", "Time(min)", "$x_D$");
185 a=get("current_axes");
186 c=a.y_label;c.font_size=5;
187
188 //
189 //=====
190 //
191 //=====Set point as +1 in X-B=====Both loops
192     Automatic=====//
193 //p is the controller output
194 p=zeros(n,2);
195 delta_p=zeros(n,2);

```

```

192 e=zeros(n,2); //errors=(ysp-y) on which PI acts
193 ysp=zeros(n,2);
194 ysp((n-1)/2+1:n,2)=ones(n-((n-1)/2+1)+1,1);
195
196 t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
197 y=zeros(n,2);
198
199
200 for k=(n-1)/2+1:n
201
202     //Error e
203     e(k,:)=ysp(k-1,:)-y(k-1,:);
204     delta_e(k,:)=e(k,:)-e(k-1,:);
205
206     //Controller calculation ——Digital PID——Eqn
207     // p(k,:)=p(k-1,:)+flipdim([delta_e(k,:)+e(k,:)*
208     // diag(Ts./tauI)]*diag(Kc),2);
209     //1-1/2-2 pairing
210
211     delta_p(k,:)=p(k,:)-p(k-1,:);
212
213     //Output
214     y(k,1)=[S(1:N-1,1);S(1:N-1,3)]'...
215     * [flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
216     // delta_p(k-N+1:k-1,2),1)]...
217     + [S(N,1) S(N,3)]*[p(k-N,1);p(k-N,2)];
218     y(k,2)=[S(1:N-1,2);S(1:N-1,4)]'...
219     * [flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
220     // delta_p(k-N+1:k-1,2),1);]...
221     + [S(N,2) S(N,4)]*[p(k-N,1);p(k-N,2)];
222 end
223 subplot(2,1,2);
224 plot(t',y(:,2),'r—');
225 set(gca(),"data_bounds",[0 40 0 1.4]); //putting

```

```

    bounds on display
225 l=legend("$x_D\text{ loop in manual}$", "$\text{Both
    loops in automatic}$", position=1);
226 xtitle("", "Time(min)", "$x_B$");
227 a=get("current_axes");
228 c=a.y_label;c.font_size=5;
229
230
231 //Also refer to Example 22.4 for similar application
    of algorithm of multiploop PID

```

---

Scilab code Exa 16.6 Sensitivity of steady state gain matrix

```

1
2 clear
3 clc
4
5 //Example 16.6
6 disp('Example 16.6 ')
7
8
9 K1=[1 0;10 1]; //K with K12=0
10
11 eig1=spec(K1);
12 sigma1=spec(K1'*K1);
13 CN1=sqrt(max(sigma1)/min(sigma1))
14 mprintf('\nEigenvalues of K1 are %f and %f\n and CN
    is %f',eig1',CN1)
15
16
17
18
19 K2=[1 0.1;10 1]; //K with K12=0.1
20
21 eig2=spec(K2);

```

```

22 sigma2=spec(K2'*K2);
23 CN2=sqrt(max(sigma2)/min(sigma2))
24
25 mprintf('\nEigenvalues of K2 are %f and %f\n and CN
    is %f',eig2',CN2)

```

---

### Scilab code Exa 16.7 Preferred multiloop control strategy

```

1 clear
2 clc
3
4 //Example 16.7
5 disp('Example 16.7')
6
7
8 X=[0.48 0.9 -0.006;0.52 0.95 0.008; 0.90 -0.95
    0.020];
9 [U,S,V]=svd(X)
10
11 RGA=X.*([inv(X)]') //Eqn 16-36
12
13 //Condition no. of X
14 CN=max(diag(S))/min(diag(S))
15
16 //Note that condition no. can also be found with
    command cond(X)
17
18 // The RGA given in the book is wrong! Eqn 16-73 is
    wrong.
19 mprintf('\n The RGA given in the book is wrong! Eqn
    16-73 is wrong.\n')
20 disp(RGA,'RGA=')
21
22 X1=X(1:2,1:2);
23 X2=X(1:2,[1 3]);

```

```

24 X3=X(1:2,2:3);
25
26 X4=X([1 3],1:2);
27 X5=X([1 3],[1 3]);
28 X6=X([1 3],2:3);
29
30 X7=X([2 3],1:2);
31 X8=X([2 3],[1 3]);
32 X9=X([2 3],2:3);
33
34 lamda1=max(X1.*inv(X1'));
35 lamda2=max(X2.*inv(X2'));
36 lamda3=max(X3.*inv(X3'));
37 lamda4=max(X4.*inv(X4'));
38 lamda5=max(X5.*inv(X5'));
39 lamda6=max(X6.*inv(X6'));
40 lamda7=max(X7.*inv(X7'));
41 lamda8=max(X8.*inv(X8'));
42 lamda9=max(X9.*inv(X9'));
43
44
45 mprintf('\n Pairing no.          CN          lambda
      \n')
46 mprintf('\n 1          %f          %f',cond(X1),
      lamda1)
47 mprintf('\n 2          %f          %f',cond(X2),
      lamda2)
48 mprintf('\n 3          %f          %f',cond(X3),
      lamda3)
49 mprintf('\n 4          %f          %f',cond(X4),
      lamda4)
50 mprintf('\n 5          %f          %f',cond(X5),
      lamda5)
51 mprintf('\n 6          %f          %f',cond(X6),
      lamda6)
52 mprintf('\n 7          %f          %f',cond(X7),
      lamda7)
53 mprintf('\n 8          %f          %f',cond(X8),

```

```
    lamda8)
54 mprintf('\n 9          %f          %f', cond(X9),
    lamda9)
```

---



# Chapter 17

## Digital Sampling Filtering and Control

Scilab code Exa 17.1 Performance of alternative filters

```
1 clear
2 clc
3
4 //Example 17.1
5 disp('Example 17.1 ')
6
7 //In this solution we assume that a sampled signal
   is given to us at a very fast
8 //sampling rate and then we resample from it for our
   computations
9 //This depicts how data is in practical situations.
10 //Since computers are digital data is always
   discrete
11 //A more kiddish way of writing this code would have
   been to make a function
12 //which takes time as input and gives signal value
   as output ie generate a
13 //continuous signal definition. Then no matter what
   our sampling time is
```

```

14 //we can always get the desired values by calling
    the function say func(Ts*k)
15 //where Ts denotes sampling time and k is the index
    no.(ie deltaT*k)
16 //In principle this will also work fine and will
    reduce the length of the code
17 //but this will not lead to learning for coding in
    practical situations
18
19 Ts=0.001 //sampling time for analog
20 t=0:Ts:5;
21 n=length(t);
22 square_base=0.5*squarewave((t-0.5)*2*%pi/3)+0.5;
23 ym=square_base+0.25*sin(t*2*%pi*9);
24 subplot(2,2,1)
25 plot(t,[square_base' ym'])
26 xtitle('Fig 17.6 (a)', 'Time(min)', 'Output');
27
28
29 //Analog Filter
30 tauf1=0.1;tauf2=0.4;
31 s=%s;
32 F1=syslin('c',1/(tauf1*s+1));
33 F2=syslin('c',1/(tauf2*s+1));
34 yf1=csim(ym,t,F1);
35 yf2=csim(ym,t,F2);
36 subplot(2,2,2);
37 plot(t,[yf1' yf2' square_base'])
38 legend("$\tau_F=0.1\ \text{min}$", "$\tau_F=0.4\ \text{min}$",
    position=3);
39 xtitle('Fig 17.6 (b)', 'Time(min)', 'Output');
40
41 //Note that analog filtering can also be achieved
    by perfect sampling in EWMA digital filter
42 //Since Exponentially weighted digital filter is an
    exact discretization of analog
43 //filter if we take Ts=0.001 ie the perfect
    sampling of data we get identical answers

```

```

44 //from digital or analog filter. You can try this
    by chaning Ts1 or Ts2 to 0.001
45
46 //Digital filtering
47 Ts1=0.05;Ts2=0.1;
48 alpha1=exp(-Ts1/tauf1);
49 alpha2=exp(-Ts2/tauf1);
50 samples1=1:Ts1/Ts:n;
51 samples2=1:Ts2/Ts:n;
52 yf1=zeros(length(samples1),1);
53 yf2=zeros(length(samples2),1);
54
55 for k=1:length(samples1)-1
56     yf1(k+1)=alpha1*yf1(k)+(1-alpha1)*ym(samples1(k)
        );
57 end
58 for k=1:length(samples2)-1
59     yf2(k+1)=alpha2*yf2(k)+(1-alpha2)*ym(samples2(k)
        );
60 end
61
62 subplot(2,2,3);
63 plot(t(samples1)',[yf1],color='blue');
64 plot(t(samples2)',yf2,color='red');
65 plot(t,square_base,color='black');
66 legend("$\Delta t=0.05 \ min$", "$\Delta t=0.1 \ min$"
        ",position=3);
67 xtitle('Fig 17.6 (c)', 'Time(min)', 'Output');
68
69
70
71 //Moving Filter
72 N1=3;
73 N2=7;
74 yf1=zeros(1,length(samples1))
75 yf2=zeros(1,length(samples1))
76 for k=N1+1:length(samples1)
77     yf1(k)=yf1(k-1)+1/N1*(ym(samples1(k))-ym(

```

```

        samples1(k-N1)));
78 end
79 for k=N2+1:length(samples1)
80     yf2(k)=yf2(k-1)+1/N2*(ym(samples1(k))-ym(
        samples1(k-N2)));
81 end
82 //for k=N2+1:n
83 //     yf2(k)=yf2(k-1)+1/N2*(ym(k)-ym(k-N2));
84 //end
85 subplot(2,2,4);
86 plot(t(samples1),[yf1' yf2'])
87 plot(t,square_base,color='black');
88 legend("$N^*=3$","$N^*=7$",position=4);
89 xtitle('Fig 17.6 (d)', 'Time(min)', 'Output');
90
91
92
93 //Now for the gaussian noise
94 scf();
95 Ts=0.001 //sampling time for analog
96 t=0:Ts:5;
97 n=length(t);
98 square_base=0.5*squarewave((t-0.5)*2*%pi/3)+0.5;
99 ym=square_base+grand(1,length(t),'nor', 0, sqrt(0.1)
    );//0.1 is for setting variance=0.1
100 subplot(2,2,1)
101 plot(t,[square_base' ym'])
102 xtitle('Fig 17.6 (a)', 'Time(min)', 'Output');
103
104
105 //Analog Filter
106 tauf1=0.1;tauf2=0.4;
107 s=%s;
108 F1=symlin('c',1/(tauf1*s+1));
109 F2=symlin('c',1/(tauf2*s+1));
110 yf1=csim(ym,t,F1);
111 yf2=csim(ym,t,F2);
112 subplot(2,2,2);

```

```

113 plot(t,[yf1' yf2' square_base'])
114 legend("$\tau_F=0.1\ min$", "$\tau_F=0.4\ min$",
        position=3);
115 xtitle('Fig 17.6 (b)', 'Time(min)', 'Output');
116
117
118 //Digital filtering
119 Ts1=0.05;Ts2=0.1;
120 alpha1=exp(-Ts1/tauf1);
121 alpha2=exp(-Ts2/tauf1);
122 samples1=1:Ts1/Ts:n;
123 samples2=1:Ts2/Ts:n;
124 yf1=zeros(length(samples1),1);
125 yf2=zeros(length(samples2),1);
126
127 for k=1:length(samples1)-1
128     yf1(k+1)=alpha1*yf1(k)+(1-alpha1)*ym(samples1(k)
        );
129 end
130 for k=1:length(samples2)-1
131     yf2(k+1)=alpha2*yf2(k)+(1-alpha2)*ym(samples2(k)
        );
132 end
133
134 subplot(2,2,3);
135 plot(t(samples1)', [yf1], color='blue');
136 plot(t(samples2)', yf2, color='red');
137 plot(t, square_base, color='black');
138 legend("$\Delta t=0.05 \ min$", "$\Delta t=0.1\ min$",
        ", position=3);
139 xtitle('Fig 17.6 (c)', 'Time(min)', 'Output');
140
141
142
143 //Moving Filter
144 N1=3;
145 N2=7;
146 yf1=zeros(1, length(samples1))

```

```

147 yf2=zeros(1,length(samples1))
148 for k=N1+1:length(samples1)
149     yf1(k)=yf1(k-1)+1/N1*(ym(samples1(k))-ym(
        samples1(k-N1)));
150 end
151 for k=N2+1:length(samples1)
152     yf2(k)=yf2(k-1)+1/N2*(ym(samples1(k))-ym(
        samples1(k-N2)));
153 end
154 //for k=N2+1:n
155 //     yf2(k)=yf2(k-1)+1/N2*(ym(k)-ym(k-N2));
156 //end
157 subplot(2,2,4);
158 plot(t(samples1),[yf1' yf2'])
159 plot(t,square_base,color='black');
160 legend("$N^*=3$","$N^*=7$",position=4);
161 xtitle('Fig 17.6 (d)', 'Time(min)', 'Output');
162
163
164
165 mprintf("Please note that for gaussian noise\n
        results ...
166     will be different from book owing to randomness\n
        n...
167     we do not know the seed for the random noise")

```

---

Scilab code Exa 17.2 Response of first order difference equation

```

1 clear
2 clc
3

```

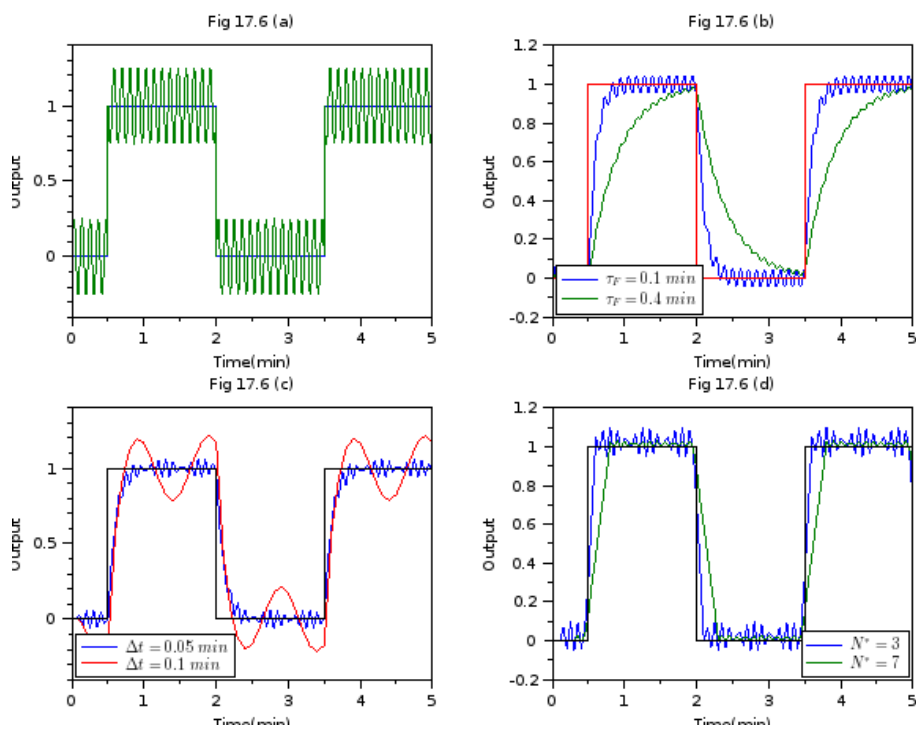


Figure 17.1: Performance of alternative filters

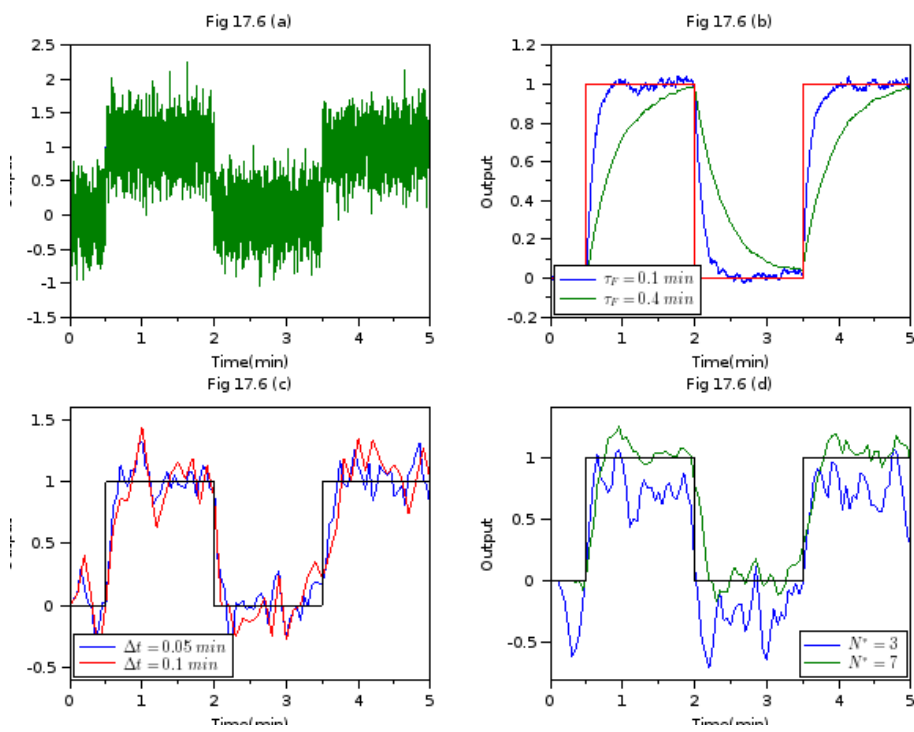


Figure 17.2: Performance of alternative filters



```

4 //Example 17.2
5 disp('Example 17.2 ')
6
7 Ts=1; //sampling time
8 K=2;
9 tau=1;
10 alpha=exp(-Ts/tau)
11 n=10;
12 y=zeros(n,1)
13 u=1; //input
14
15 for i=1:n
16     y(i+1)=alpha*y(i)+K*(1-alpha)*u;
17 end
18
19 disp(y, 'yk=')
20
21 mprintf("\n Note that in the book K=20 is wrong, it
22     should be K=2\n...
23     that is a first order function with gain 2 is given
24     an input step")

```

---

### Scilab code Exa 17.3 Recursive relation with inputs

```

1 clear
2 clc
3
4 //Example 17.3
5 disp('Example 17.3 ')
6
7 z=%z;
8 Gz=(-0.3225*z^-2+0.5712*z^-3)/(1-0.9744*z^-1+0.2231*
9     z^-2);
10 G=tf2ss(Gz)
11 n=10;

```

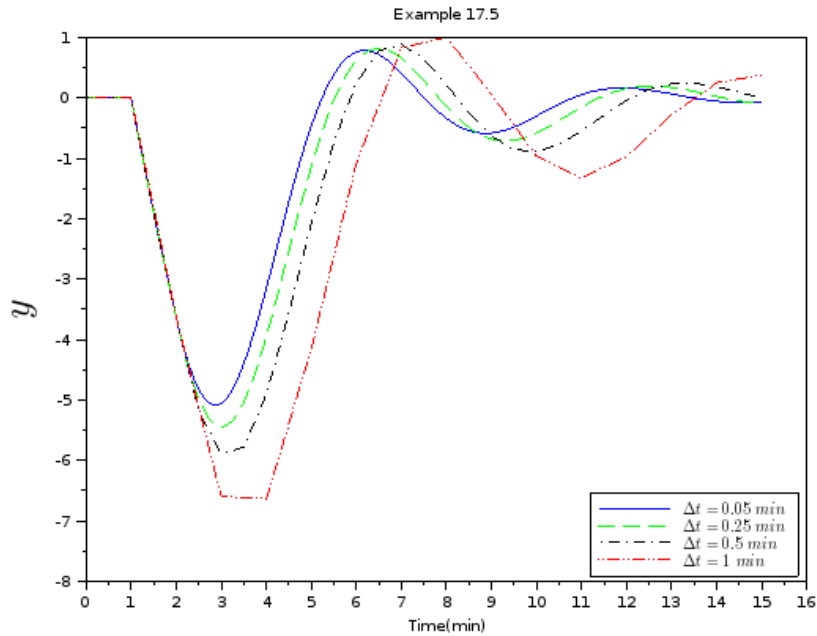


Figure 17.3: Digital control of pressure in a tank

```

11 u=ones(1,n);
12 y=dsimul(G,u);
13 disp(y','y=')
14
15 mprintf('\n\nAlternatively the simulation can also
    be done\n...
16 using syslin(d,Gz) and flts(u,Gz)\n\n')
17
18 Gz2=syslin('d',Gz);
19 y2=flts(u,Gz2)
20 disp(y2','y2=')
```

---

Scilab code Exa 17.5 Digital control of pressure in a tank

```
1 clear
2 clc
3
4 //Example 17.5
5 disp('Example 17.5')
6
7 deltaT=[0.05 0.25 0.5 1]';//sampling time
8 K=-20;
9 theta=1+(deltaT/2);//Add half of sampling time to
   delay for finding PI settings
10 tau=5;
11
12 //Table 11.3 ITAE disturbance settings
13 //Note that there is an error in book solution
   saying Table 11.2
14 //It should be table 11.3
15
16 Y=0.859*(theta/tau)^(-0.977);Kc=Y/K;
17 tauI=tau*(0.674*(theta/tau)^-0.680).^-1;
18
19 mprintf('\n      ITAE(disturbance)      \n')
20 mprintf('      deltaT      Kc      tauI')
21 mprintf('\n %f      %f      %f',deltaT,Kc,tauI)
22
23 //Finding digital controller settings
24 //Eqn 17-55
25 a0=1+deltaT./tauI;
26 a1=-(1); //since tauD=0
27 a2=0;
28 z=%z;
29
30 Gcz=Kc.*(a0+a1*z^-1)./(1-z^-1);
31
32 //Refer to table 17.1 to convert continuous transfer
   function to digital form
33 Gp=K*(1-exp(-1/tau*deltaT)).*z^(-1+(-1)./deltaT)
```

```

        ./((1-exp((-1)/tau*deltaT)*z^-1)); //z^(-1/deltaT)
    for delay
34
35 G_CL=syslin('d',((Gp)./(Gcz.*Gp+1)));
36
37 t=0:deltaT(1):15
38 u=ones(1,length(t));
39 yt=flds(u,G_CL(1,1));
40 plot(t,yt,'-')
41
42 t=0:deltaT(2):15
43 u=ones(1,length(t));
44 yt=flds(u,G_CL(2,1));
45 plot(t,yt,'green—')
46
47 t=0:deltaT(3):15
48 u=ones(1,length(t));
49 yt=flds(u,G_CL(3,1));
50 plot(t,yt,'black-.')
51
52 t=0:deltaT(4):15
53 u=ones(1,length(t));
54 yt=flds(u,G_CL(4,1));
55 plot(t,yt,'red:');
56
57 set(gca(),"data_bounds",[0 15 -8 1]); //putting
    bounds on display
58 l=legend("$\Delta t=0.05\ min$", "$\Delta t=0.25\
    min$", "$\Delta t=0.5\ min$", "$\Delta t=1\ min$",
    position=4);
59 xtitle("Example 17.5", "Time(min)", "$y$");
60 a=get("current_axes");
61 c=a.y_label;c.font_size=5;
62
63 mprintf("\nNote that there is a mismatch between the
    book simulation and what\n...
64 what we get from SCILAB. The book is wrong. This has
    been crosschecked using\n...

```

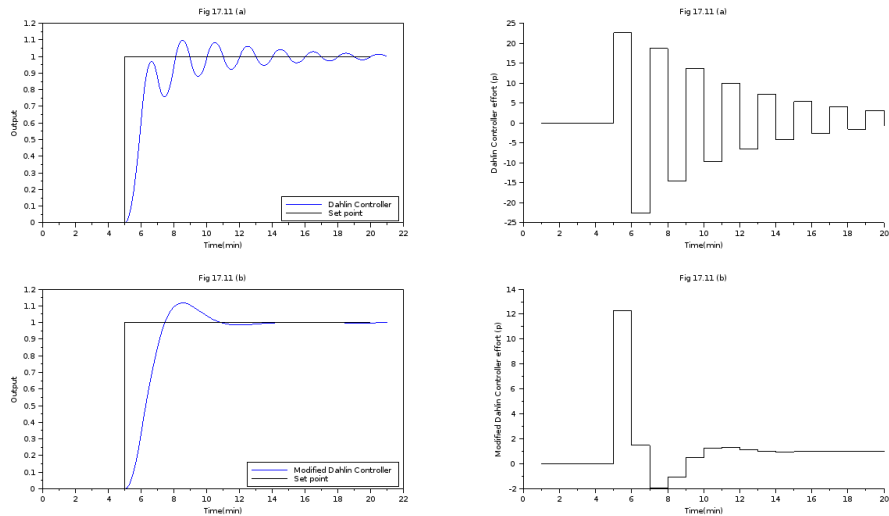


Figure 17.4: Dahlin controller

65 simulation in SIMULINK (MATLAB)”)

---

### Scilab code Exa 17.6 Dahlin controller

```

1 clear
2 clc
3
4 //Example 17.6
5 disp('Example 17.6')
6
7 //Note that for solving this example there are two
  ways
8 //One is to do this in xcos which is very easy to do
9 //and one can learn the same from example 17.5's
  solution
10 //To get the controller outputs at every point in

```

```

        xcos
11 //just add a scope to the leg connecting controller
    and
12 //zero order hold unit before the continuous time
    block
13
14 //The other method is given here so that the reader
    learns more
15 //of what all can be done in scilab
16 //Here we deal with the controller in time domain
    rather than z domain
17
18 z=%z;
19 N=0;
20 a1=-1.5353;
21 a2=0.5866;
22 b1=0.0280;
23 b2=0.0234;
24 G=(b1+b2*z^-1)*z^(-N-1)/(1+a1*z^-1+a2*z^-2);
25
26 h=0;//no process delay
27 s=%s;
28 lamda=1;
29 Y_Ysp=1/(lamda*s+1);//exp(-h*s) is one because h=0
    Eqn 17-62
30
31 Ts=1;//sampling time
32 A=exp(-Ts/lamda);
33 //Eqn 17-63
34 Y_Ysp_d=(1-A)*z^(-N-1)/(1-A*z^-1);
35
36 G_DC=1/G*(Y_Ysp_d)/(1-Y_Ysp_d); //Eqn 17-61
37
38
39
40 ysp=[zeros(1,4) ones(1,16)]
41 Gz_CL=syslin('d',G*G_DC/(G*G_DC+1)); //Closed loop
    discrete system

```

```

42 yd=flds(yssp,Gz_CL) //Discrete Output due to set
    point change
43 //plot(yd)
44
45 e=yssp-yd; //Since we know set point and the output
    of the system we can use
46 //this info to find out the errors at the discrete
    time points
47 //note that here we have exploited in a very subtle
    way the property of a
48 //discrete system that only the values at discrete
    points matter for
49 //any sort of calculation
50
51 //Now this error can be used to find out the
    controller effort
52 e_coef=coeff( numer(G_DC));
53 p_coef=coeff( denom(G_DC));
54
55 n=20; //Time in minutes discretized with Ts=1 min
56 p=zeros(1,n); //Controller effort
57
58 for k=3:n
59     p(k)=(-p_coef(2)*p(k-1)-p_coef(1)*p(k-2)+
        e_coef*[e(k-2) e(k-1) e(k)]')/p_coef(3);
60 end
61 subplot(2,2,2)
62 plot2d2(p)
63 xtitle('Fig 17.11 (a)', 'Time(min)', 'Dahlin
    Controller effort (p)');
64
65 //Now we simulate the continuous version of the
    plant to get output in between
66 //the discrete point. This will help us ascertain
    the efficacy of the controller
67 //at points other than the discrete points
68 //Note that this is required to be checked because
    deltaT=1. had it been much

```

```

69 //smaller like 0.01 it would have been a good approx
    to a continuous system
70 //thus making this interpolation check redundant
71
72 s=%s;
73 Gp=syslin('c',1/(5*s+1)/(3*s+1)); //continuous time
    version of process
74 Ts_c=0.01; //sampling time for continuous system
75 t=Ts_c:Ts_c:length([0 p])*Ts;
76 p_c=matrix(repmat([0 p],Ts/Ts_c,1),1,Ts/Ts_c*length
    ([0 p])) //hack for zero order hold
77 //p_c means controller effort which is continous
78 yc=csim(p_c,t,Gp);
79 subplot(2,2,1)
80 plot(t,yc)
81 plot2d2(ysp)
82 legend("Dahlin Controller","Set point",position=4)
83 xtitle('Fig 17.11 (a)', 'Time(min)', 'Output');
84
85
86
87 //=====Now we do calculations for modified
    Dahlin controller=====//
88 //


---


89 //Y_Ysp_d=(1-A)*z^(-N-1)/(1-A*z^-1)*(b1+b2*z^-1)/(b1
    +b2); //Vogel Edgar
90
91 //Page 362 just after solved example
92 G_DC_bar=(1-1.5353*z^-1+0.5866*z^-2)/(0.0280+0.0234)
    *0.632/(1-z^-1);
93 //G_DC2=1/G*((1-A)*z^(-N-1))/(1-A*z^-1-(1-A)*z^(-N
    -1)); //Eqn 17-61
94 //G_DC=(1-1.5353*z^-1+0.5866*z^-2)/(0.0280+0.0234*z
    ^-1)*0.632/(1-z^-1);
95
96 ysp=[zeros(1,4) ones(1,16)]

```



```

97 Gz_CL=syslin('d',G*G_DC_bar/(G*G_DC_bar+1)); //Closed
    loop discrete system
98 yd=flts(yssp,Gz_CL) //Discrete Output due to set
    point change
99 //plot(yd)
100
101 e=yssp-yd; //Since we know set point and the output
    of the system we can use
102 //this info to find out the errors at the discrete
    time points
103 //note that here we have exploited in a very subtle
    way the property of a
104 //discrete system that only the values at discrete
    points matter for
105 //any sort of calculation
106
107 //Now this error can be used to find out the
    controller effort
108 e_coef=coeff( numer(G_DC_bar));
109 p_coef=coeff( denom(G_DC_bar));
110
111 n=20; //Time in minutes discretized with Ts=1 min
112 p=zeros(1,n); //Controller effort
113
114 for k=3:n
115     p(k)=(-p_coef(2)*p(k-1)-p_coef(1)*p(k-2)+
            e_coef*[e(k-2) e(k-1) e(k)]')/p_coef(3);
116 end
117 subplot(2,2,4)
118 plot2d2(p)
119 xtitle('Fig 17.11 (b)', 'Time(min)', 'Modified Dahlin
    Controller effort (p)');
120
121 //Now we simulate the continuous version of the
    plant to get output in between
122 //the discrete point. This will help us ascertain
    the efficacy of the controller
123 //at points other than the discrete points

```

```

124 //Note that this is required to be checked because
      deltaT=1. had it been much
125 //smaller like 0.01 it would have been a good approx
      to a continuous system
126 //thus making this interpolation check redundant
127
128 s=%s;
129 Gp=syslin('c',1/(5*s+1)/(3*s+1)); //continuous time
      version of process
130 Ts_c=0.01; //sampling time for continuous system
131 t=Ts_c:Ts_c:length([0 p])*Ts;
132 p_c=matrix(repmat([0 p],Ts/Ts_c,1),1,Ts/Ts_c*length
      ([0 p])) //hack for zero order hold
133 //p_c means controller effort which is continuous
134 yc=csim(p_c,t,Gp);
135 subplot(2,2,3)
136 plot(t,yc)
137 plot2d2(y_c)
138 legend("Modified Dahlin Controller","Set point",
      position=4)
139 xtitle('Fig 17.11 (b)', 'Time(min)', 'Output');

```

---

#### Scilab code Exa 17.7 Non ringing Dahlin controller

```

1 clear
2 clc
3
4 //Example 17.7
5 disp('Example 17.7')
6
7 //Note that for solving this example there are two
      ways
8 //One is to do this in xcos which is very easy to do

```

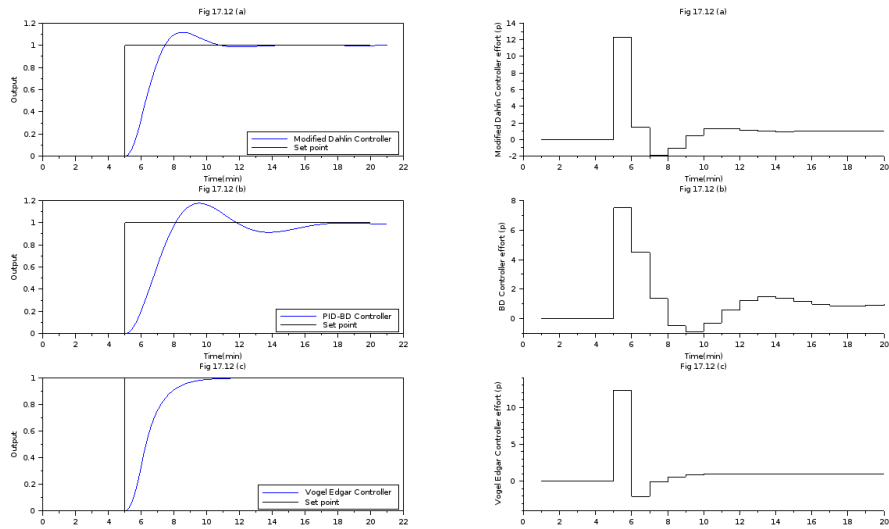


Figure 17.5: Non ringing Dahlin controller

```

9 //and one can learn the same from example 17.5's
  solution
10 //To get the controller outputs at every point in
  xcos
11 //just add a scope to the leg connecting controller
  and
12 //zero order hold unit before the continuous time
  block
13
14 //The other method is given here so that the reader
  learns more
15 //of what all can be done in scilab
16 //Here we deal with the controller in time domain
  rather than z domain
17
18 z=%z;
19 N=0;
20 a1=-1.5353;
21 a2=0.5866;
22 b1=0.0280;

```

```

23 b2=0.0234;
24 G=(b1+b2*z^-1)*z^(-N-1)/(1+a1*z^-1+a2*z^-2);
25
26 h=0; //no process delay
27 s=%s;
28 lamda=1;
29 Y_Ysp=1/(lamda*s+1); //exp(-h*s) is one because h=0
    Eqn 17-62
30
31 Ts=1; //sampling time
32 A=exp(-Ts/lamda);
33
34
35 //=====Now we do calculations for modified
    Dahlin controller=====//
36 //

```

---

```

37
38 //Page 362 just after solved example
39 G_DC_bar=(1-1.5353*z^-1+0.5866*z^-2)/(0.0280+0.0234)
    *0.632/(1-z^-1);
40
41 ysp=[zeros(1,4) ones(1,16)]
42 Gz_CL=syslin('d',G*G_DC_bar/(G*G_DC_bar+1)); //Closed
    loop discrete system
43 yd=flts(ysp,Gz_CL) //Discrete Output due to set
    point change
44 //plot(yd)
45
46 e=ysp-yd; //Since we know set point and the output
    of the system we can use
47 //this info to find out the errors at the discrete
    time points
48 //note that here we have exploited in a very subtle
    way the property of a
49 //discrete system that only the values at discrete
    points matter for

```

```

50 //any sort of calculation
51
52 //Now this error can be used to find out the
    controller effort
53 e_coeff=coeff(numer(G_DC_bar));
54 p_coeff=coeff(denom(G_DC_bar));
55
56 n=20; //Time in minutes discretized with Ts=1 min
57 p=zeros(1,n); //Controller effort
58
59 for k=3:n
60     p(k)=(-p_coeff(2)*p(k-1)-p_coeff(1)*p(k-2)+
        e_coeff*[e(k-2) e(k-1) e(k)]')/p_coeff(3);
61 end
62 subplot(3,2,2)
63 plot2d2(p)
64 xtitle('Fig 17.12 (a)', 'Time(min)', 'Modified Dahlin
    Controller effort (p)');
65
66 //Now we simulate the continuous version of the
    plant to get output in between
67 //the discrete point. This will help us ascertain
    the efficacy of the controller
68 //at points other than the discrete points
69 //Note that this is required to be checked because
    deltaT=1. had it been much
70 //smaller like 0.01 it would have been a good approx
    to a continuous system
71 //thus making this interpolation check redundant
72
73 s=%s;
74 Gp=syslin('c',1/(5*s+1)/(3*s+1)); //continuous time
    version of process
75 Ts_c=0.01; //sampling time for continuous system
76 t=Ts_c:Ts_c:length([0 p])*Ts;
77 p_c=matrix(repmat([0 p],Ts/Ts_c,1),1,Ts/Ts_c*length
    ([0 p])) //hack for zero order hold
78 //p_c means controller effort which is continuous

```

```

79 yc=csim(p_c,t,Gp);
80 subplot(3,2,1)
81 plot(t,yc)
82 plot2d2(yssp)
83 legend("Modified Dahlin Controller","Set point",
      position=4)
84 xtitle('Fig 17.12 (a)', 'Time(min)', 'Output');
85
86
87
88
89 //=====Now we do calculations for PID-BD
      controller=====//
90 //
      =====

91 G_BD=4.1111*(3.1486-5.0541*z^-1+2.0270*z^-2)
      /(1.7272-2.4444*z^-1+0.7222*z^-2)
92
93
94 yssp=[zeros(1,4) ones(1,16)]
95 Gz_CL=syslin('d',G*G_BD/(G*G_BD+1)); //Closed loop
      discrete system
96 yd=flts(yssp,Gz_CL) //Discrete Output due to set
      point change
97 //plot(yd)
98
99 e=yssp-yd; //Since we know set point and the output
      of the system we can use
100 //this info to find out the errors at the discrete
      time points
101 //note that here we have exploited in a very subtle
      way the property of a
102 //discrete system that only the values at discrete
      points matter for
103 //any sort of calculation
104
105 //Now this error can be used to find out the

```

```

        controller effort
106 e_coeff=coeff( numer(G_BD));
107 p_coeff=coeff( denom(G_BD));
108
109 n=20; //Time in minutes discretized with Ts=1 min
110 p=zeros(1,n); //Controller effort
111
112 for k=3:n
113     p(k)=(-p_coeff(2)*p(k-1)-p_coeff(1)*p(k-2)+
            e_coeff*[e(k-2) e(k-1) e(k)]')/p_coeff(3);
114 end
115 subplot(3,2,4)
116 plot2d2(p)
117 xtitle('Fig 17.12 (b)', 'Time(min)', 'BD Controller
        effort (p)');
118
119 //Now we simulate the continuous version of the
        plant to get output in between
120 //the discrete point. This will help us ascertain
        the efficacy of the controller
121 //at points other than the discrete points
122 //Note that this is required to be checked because
        deltaT=1. had it been much
123 //smaller like 0.01 it would have been a good approx
        to a continuous system
124 //thus making this interpolation check redundant
125
126 s=%s;
127 Gp=syslin('c',1/(5*s+1)/(3*s+1)); //continuous time
        version of process
128 Ts_c=0.01; //sampling time for continuous system
129 t=Ts_c:Ts_c:length([0 p])*Ts;
130 p_c=matrix(repmat([0 p],Ts/Ts_c,1),1,Ts/Ts_c*length
        ([0 p])) //hack for zero order hold
131 //p_c means controller effort which is continous
132 yc=csim(p_c,t,Gp);
133 subplot(3,2,3)
134 plot(t,yc)

```

```

135 plot2d2(ysp)
136 legend("PID-BD Controller", "Set point", position=4)
137 xtitle('Fig 17.12 (b)', 'Time(min)', 'Output');
138
139
140
141 //=====Now we do calculations for Vogel
    Edgar Dahlin controller=====//
142 //
    _____

143 Y_Ysp_d=(1-A)*z^(-N-1)/(1-A*z^-1)*(b1+b2*z^-1)/(b1+
    b2); //Vogel Edgar Eqn 17-70
144
145 G_VE=1/G*(Y_Ysp_d)/(1-Y_Ysp_d); //Eqn 17-61
146
147
148 ysp=[zeros(1,4) ones(1,16)]
149 Gz_CL=syslin('d',G*G_VE/(G*G_VE+1)); //Closed loop
    discrete system
150 yd=flds(ysp,Gz_CL) //Discrete Output due to set
    point change
151 //plot(yd)
152
153 e=ysp-yd; //Since we know set point and the output
    of the system we can use
154 //this info to find out the errors at the discrete
    time points
155 //note that here we have exploited in a very subtle
    way the property of a
156 //discrete system that only the values at discrete
    points matter for
157 //any sort of calculation
158
159 //Now this error can be used to find out the
    controller effort
160 e_coeff=coeff(numer(G_VE));
161 p_coeff=coeff(denom(G_VE));

```



```

162
163 n=20; //Time in minutes discretized with Ts=1 min
164 p=zeros(1,n); //Controller effort
165
166 for k=3:n
167     p(k)=(-p_coeff(2)*p(k-1)-p_coeff(1)*p(k-2)+
            e_coeff*[e(k-2) e(k-1) e(k)]')/p_coeff(3);
168 end
169 subplot(3,2,6)
170 plot2d2(p)
171 xtitle('Fig 17.12 (c)', 'Time(min)', 'Vogel Edgar
        Controller effort (p)');
172
173 //Now we simulate the continuous version of the
        plant to get output in between
174 //the discrete point. This will help us ascertain
        the efficacy of the controller
175 //at points other than the discrete points
176 //Note that this is required to be checked because
        deltaT=1. had it been much
177 //smaller like 0.01 it would have been a good approx
        to a continuous system
178 //thus making this interpolation check redundant
179
180 s=%s;
181 Gp=syslin('c',1/(5*s+1)/(3*s+1)); //continuous time
        version of process
182 Ts_c=0.01; //sampling time for continuous system
183 t=Ts_c:Ts_c:length([0 p])*Ts;
184 p_c=matrix(repmat([0 p],Ts/Ts_c,1),1,Ts/Ts_c*length
        ([0 p])) //hack for zero order hold
185 //p_c means controller effort which is continous
186 yc=csim(p_c,t,Gp);
187 subplot(3,2,5)
188 plot(t,yc)
189 plot2d2(ysp)
190 legend("Vogel Edgar Controller","Set point",position
        =4)

```

```
191 xtitle('Fig 17.12 (c)', 'Time(min)', 'Output');
192
193
194 mprintf("Note that there is some very slight
           difference between the \n...
195 curves shown in book and that obtained from scilab\n
           ...
196 this is simply because of more detailed calculation
           in scilab ")
```

---

# Chapter 19

## Real Time Optimization

Scilab code Exa 19.2 Nitration of Decane

```
1 clear
2 clc
3
4 //Example 19.2
5 disp('Example 19.2 ')
6
7 function y=f_DN03(r1)
8     D1=0.5;D2=0.5;
9     r2=0.4-0.5*r1;
10    y=r1*D1/(1+r1)^2/(1+r2)+r2*D2/(1+r1)/(1+r2)^2
11 endfunction
12
13 function [f, g, ind] = costf(x, ind)
14     f=-f_DN03(x); //cost is negative of function to
15                 //be maximised
16     g=-derivative(f_DN03,x); //derivative of the cost
17                             //function
18 endfunction
19
20 [fopt, xopt] = optim(costf,0.5);
21
```

```

20 disp(xopt,"Optimum value of r1=")
21 disp(-fopt,"Max value of DNO3=")
22
23 mprintf('Note that the answer in book is not as
         accurate as the one\n...
24 calculated from scilab')

```

---

### Scilab code Exa 19.3 Refinery blending and production

```

1 clear
2 clc
3
4 //Example 19.3
5 disp('Example 19.3')
6
7 //function for minimization
8 c=-[-24.5 -16 36 24 21 10]';
9 //Equality Constraints
10 Aeq=[0.80 0.44 -1 0 0 0;0.05 0.1 0 -1 0 0;0.1 0.36 0
      0 -1 0;0.05 0.1 0 0 0 -1];
11 beq=zeros(4,1);
12 //Inequality Constraints
13 A=[0 0 1 0 0 0;0 0 0 1 0 0;0 0 0 0 1 0];
14 b=[24000 2000 6000]';
15 //Lower bound on x
16 lb=zeros(6,1);
17 //Initial guess: such that it satisfies Aeq*x0=beq
18 x0=zeros(6,1);
19 x0(1:2)=[5000 3000]';//Initial guess for x1 and x2
20 x0(3:6)=Aeq(:,3:6)\(beq-Aeq(:,1:2)*x0(1:2));//
      solution of linear equations
21 //Note that x0 should also satisfy A*x0<b and lb
22
23
24 [xopt ,fopt]=karmarkar(Aeq,beq,c,x0,[],[],[],[],A,b,

```

```

1b)
25
26 disp(xopt,"Optimum value of x=")
27 mprintf("\nMax value of f=$ %f /day\n",-fopt)
28
29 mprintf('\n Note that the answer in book is not as
    accurate as the one\n...
30 calculated from scilab ')

```

---

#### Scilab code Exa 19.4 Fuel cost in boiler house

```

1 clear
2 clc
3
4 //Example 19.4
5 disp('Example 19.4 ')
6
7 //Here we have Nonlinear programming problem hence
    we use optim function
8 //Since optim does not have the ability to handle
    constraints
9 //we use the penalty method for optimization
10 //ie we make the constraints a part of the cost
    function such that
11 //cost function increases severly for any violation
    of constraints
12 //MATLAB users must be familiar with fmincon
    function in MATLAB
13 //Unfortunately a similar function in Scilab is not
    yet available
14 //Fmincon toolbox development for scilab is under
    development/testing
15
16 x0=[2 4 4 1]'; //Initial guess
17

```

```

18 function y=func(x) //x is 4*1 vector
19     P1=4.5*x(1)+0.1*x(1)^2+4*x(2)+0.06*x(2)^2;
20     P2=4*x(3)+0.05*x(3)^2+3.5*x(4)+0.2*x(4)^2;
21     if (P1>30) then
22         c1=abs(P1-30)^2;
23     elseif P1<18
24         c1=abs(P1-18)^2;
25     else c1=0;
26     end
27     if (P2>25) then
28         c2=abs(P2-30)^2;
29     elseif P2<14
30         c2=abs(P2-18)^2;
31     else c2=0;
32     end
33     c3=abs(P1+P2-50)^2;
34     c4=abs(x(2)+x(4)-5)^2;
35     y=(x(1)+x(3))+100*(c1+c2+c3+c4);
36 endfunction
37
38 function [f, g, ind] = costf(x, ind)
39     f=func(x); //cost is negative of function to be
40         maximised
41     g=derivative(func,x); //derivative of the cost
42         function
43 endfunction
44
45 [fopt, xopt] = optim(costf,"b",zeros(4,1), 10*ones
46     (4,1),x0);
47 // "b",zeros(4,1), 10*ones(4,1) stands for lower and
48     upper bounds on x
49
50 disp(xopt,"Optimum value of x=")
51 disp(fopt,"Min value of f=")
52
53 mprintf('Note that the answer in book is not as
54     accurate as the one\n...
55     calculated from scilab')

```



# Chapter 20

## Model Predictive Control

Scilab code Exa 20.1 Step response coefficients

```
1 clear
2 clc
3
4 //Example 20.1
5 disp('Example 20.1 ')
6
7
8 K=5;
9 tau=15; //min
10 theta=2; //min
11 Ts=1; //Sampling period
12 k=[0:79]'; //samples
13 N=80;
14
15
16 //From eqn 20-5
17 S=zeros(N,1);
18 S=K*(1-exp(-(k*Ts-theta)/tau));
19 S(1:(theta+1),1)=0; //delay
```



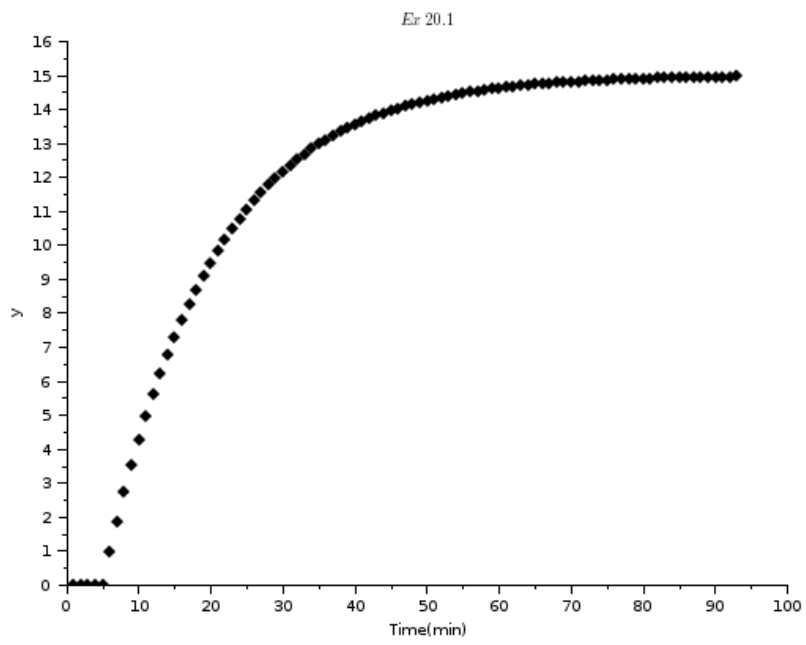


Figure 20.1: Step response coefficients

```

20
21
22 //Step change
23 M=3;
24
25 //Calculating step response from Eqn 20-4
26 step=3;//step change occurs at t=3 min
27 i=[(theta+1):90]';
28 yi=[zeros(theta+step,1);K*M*(1-exp(-(i*Ts-theta)/tau
    ));]
29
30 plot2d(yi,style=-4);
31 xtitle("$Ex\ 20.1$", "Time(min)", "y")

```

---

### Scilab code Exa 20.3 J step ahead prediction

```

1 clear
2 clc
3
4 //Example 20.3
5 disp('Example 20.3')
6
7
8 for J=[3 4 6 8] //Tuning parameter
9
10 Ts=5;//Sampling time ie delta_T
11 N=16;//Model Order
12
13 s=%s;
14 G=syslin('c',1/(5*s+1)^5);//Transfer function
15 t=0:Ts:N*Ts;
16 S=csim('step',t,G)';//generating step response model
17 //plot(t,S);

```

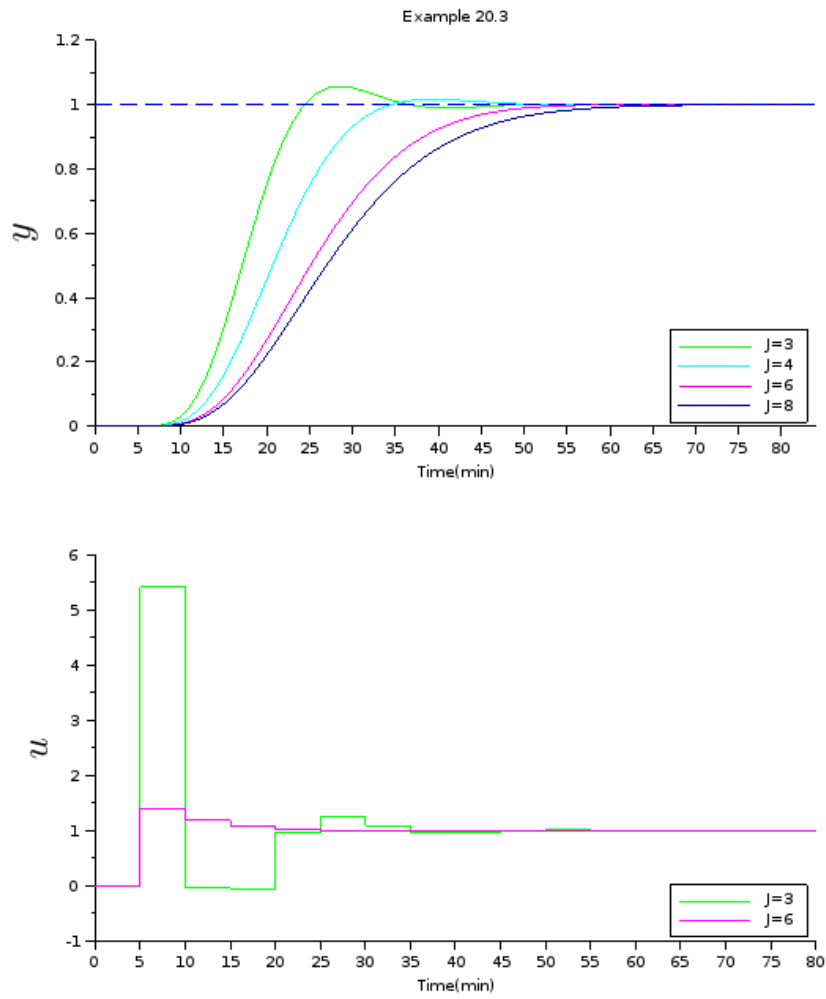


Figure 20.2: J step ahead prediction

```

18 S(1)=[];
19
20 T=80; //simulation time
21 n=T/Ts*2+1;
22 u=zeros(1,n);
23 //Input initialization 80 min is the Time for
    simulation
24 //We take a few u values in negative time to
    facilitate usage of step response
25 //model
26 delta_u=[0 diff(u)];
27 yhat_u=zeros(n,1);
28 ysp=1;
29 for k=(n-1)/2+1+1:n-J //an additional +1 is because
    MPC starts after set point change
30     yhat_u(k+J)=delta_u(k+J-N+1:k-1)*flipdim(S(J+1:N
        -1),1)+S(N)*u(k+J-N); //unforced predicted y
31     disp(yhat_u(k+J))
32     delta_u(k)=(ysp-yhat_u(k+J))/S(J);
33     u(k)=u(k-1)+delta_u(k);
34 end
35 u(n-J+1:$)=u(k)*ones(1,J); //Carry forward the u as
    constant
36
37 t=-(n-1)/2*Ts:Ts:(n-1)*Ts/2;
38 subplot(2,1,2);
39 if J==3 | J==6 then
40     plot2d2(t((n-1)/2+1:n),u((n-1)/2+1:n),style=J);
41 end
42 legend("J=3","J=6",position=4)
43 xtitle("", "Time(min)", "$u$");
44 a=get("current_axes");
45 c=a.y_label;c.font_size=5;
46
47
48 res=Ts; //resolution
49 //u_cont=matrix(repmat([0 u],res,1),1,res*length([0
    u]));

```

```

50 u_cont=matrix(repmat([u],res,1),1,res*length([u]));
51 entries=length(u_cont);
52 t_cont=linspace(-T,T+Ts-1,entries);
53 yt=csim(u_cont,t_cont,G);
54 subplot(2,1,1);
55 if J=8 then //for color of plot2d
56     J=9
57 end
58 plot2d(t_cont((entries-Ts)/2+1:$),yt((entries-Ts)
    /2+1:$),style=J,rect=[0,0,80,1.2]);
59 end
60
61 //Other niceties for plots
62 subplot(2,1,1);
63 plot(t_cont((entries-Ts)/2+1:$),ones(length(t_cont((
    entries-Ts)/2+1:$)),1),'--');
64 legend("J=3","J=4","J=6","J=8",position=4)
65 xtitle("Example 20.3","Time(min)","$y$");
66 a=get("current_axes");
67 c=a.y_label;c.font_size=5;

```

---

#### Scilab code Exa 20.4 Output feedback and bias correction

```

1 clear
2 clc
3
4 //Example 20.4
5 disp('Example 20.4')
6
7 J=15;
8 Ts=1;//Sampling time ie delta_T
9 N=80;//Model Order
10 s=%s;
11 G=syslin('c',5/(15*s+1));//Transfer function
12 t=0:Ts:N*Ts;

```

```

13 u_sim=ones(1,length(t));
14 u_sim(1:3)=[0 0 0]; //input delay to account for 2
    min delay in G
15 S=csim(u_sim,t,G)'; //generating step response model
    for real plant
16 //plot(t,S);
17 S(1)=[];
18 T=100; //simulation time
19
20 n=T/Ts*2+1; //no. of discrete points in our domain
    of analysis
21 //Input initialization T min is the Time for
    simulation
22 //We take a few u values in negative time to
    facilitate
23 //usage of step response model
24 u=zeros(n,1);
25 d=zeros(n,1);
26 delta_u=zeros(n,1);
27 delta_u(101+2)=1; //Step change at t=2 min
28 u=cumsum(delta_u);
29 delta_d=zeros(n,1);
30 delta_d(101+8)=0.15; //disturbance t=8 min
31 d=cumsum(delta_d);
32
33 yhat=zeros(n,1); //J step ahead predictions
34 ytilde=zeros(n,1); //J step ahead predictions
    corrected
35 b=zeros(n,1); //corrections
36
37 t=-(n-1)/2:Ts:(n-1)/2;
38
39 for k=(n-1)/2+1-J:n-J
40     yhat(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J
        -1),1)+S(N)*u(k+J-N);
41     //Predicted y Eqn 20-10
42     y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1)
        ,1)+S(N)*u(k+J-N)+...

```

```

43         S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1)
           ,1)+S(N)*d(k+J-N);
44     //Actual values of the process
45     b(k+J)=y(k)-yhat(k); //Note that there is a
           delay in corrections
46     //which is opposite of prediction
47 end
48 ytilde=b+yhat; //calculation of corrected y
49 plot(t,y,'-',t,yhat,'-.',t,ytilde,'--');
50 set(gca(),"data_bounds",[0 100 0 6]); //putting
           bounds on display
51 l=legend("y","$\hat{y}$","$\tilde{y}$",position=4);
52 l.font_size=5;
53 xtitle("Example 20.4","Time(min)","$y$");
54 a=get("current_axes");
55 c=a.y_label;c.font_size=5;
56
57
58 //====Part b====//
59 G2=syslin('c',4/(20*s+1)); //Transfer function
60 t2=0:Ts:N*Ts;
61 u_sim=ones(1,length(t2));
62 u_sim(1:3)=[0 0 0]; //input delay to account for 2
           min delay in G
63 S2=csim(u_sim,t2,G2)'; //generating step response
           model for model
64 //plot(t2,S);
65 S2(1)=[];
66
67 yhat=zeros(n,1); //J step ahead predictions
68 ytilde=zeros(n,1); //J step ahead predictions
           corrected
69 b=zeros(n,1); //corrections
70
71 for k=(n-1)/2+1-J:n-J
72     yhat(k+J)=S2(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J
           -1),1)+S2(N)*u(k+J-N);
73     //Predicted y Eqn 20-10

```

```

74     y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1)
        ,1)+S(N)*u(k+J-N)+...
75         S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1)
        ,1)+S(N)*d(k+J-N);
76     //Actual values of the process
77     b(k+J)=y(k)-yhat(k); //Note that there is a
        delay in corrections
78     //which is opposite of prediction
79 end
80 ytilde=b+yhat;//calculation of corrected y
81 scf();
82 plot(t,y,'-',t,yhat,'-.',t,ytilde,'--');
83 set(gca(),"data_bounds",[0 100 0 6]); //putting
        bounds on display
84 l=legend("y","$\hat{y}$","$\tilde{y}$",position=4);
85 l.font_size=5;
86 xtitle("Example 20.4","Time(min)","$y$");
87 a=get("current_axes");
88 c=a.y_label;c.font_size=5;

```

---

### Scilab code Exa 20.5 Comparison of MPCs and PID

```

1 clear
2 clc
3
4 //Example 20.5
5 disp('Example 20.5')
6
7 //=====Part (a)=====//
8 //=====Part (a)=====//
9 //=====Part (a)=====//

```



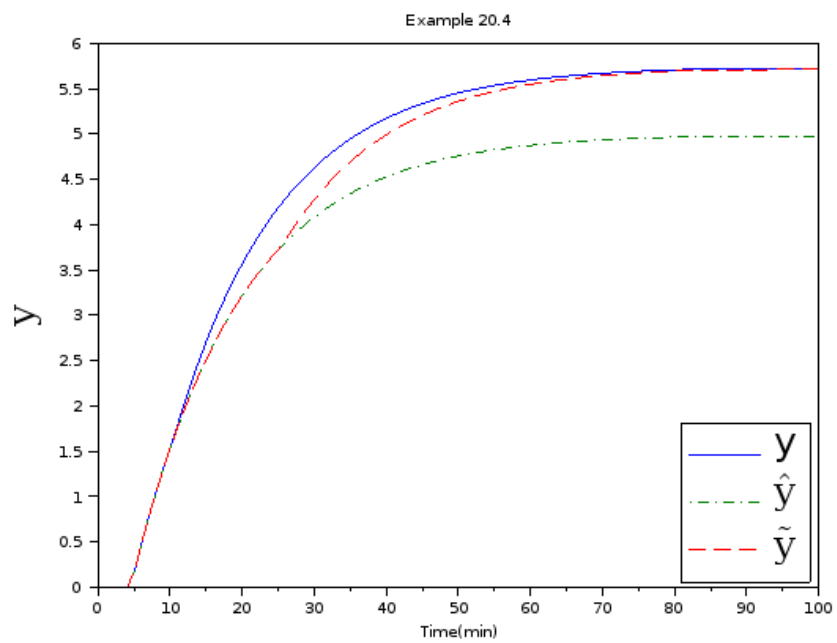


Figure 20.3: Output feedback and bias correction

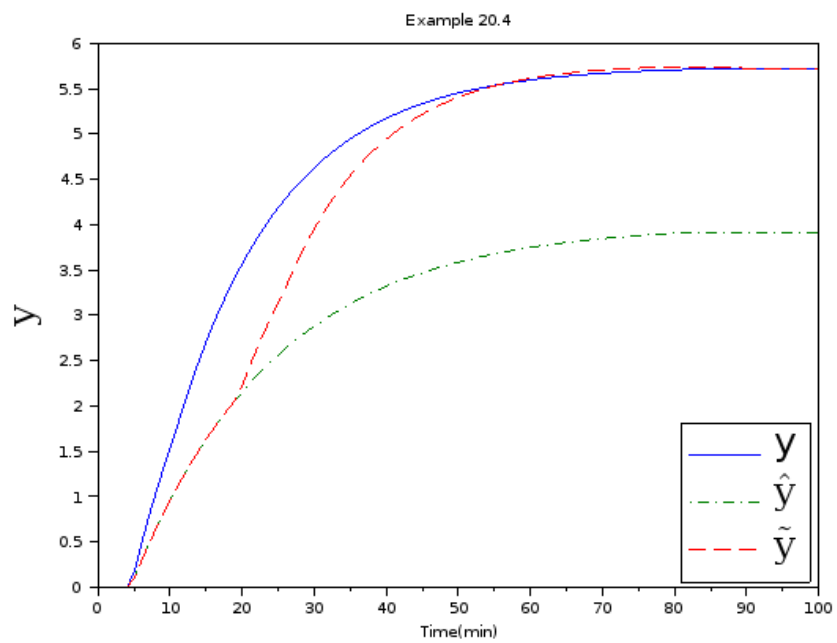


Figure 20.4: Output feedback and bias correction

```

10 //=====Part (a)=====//
11 //=====Part (a)=====//
12
13 Ts=1; //Sampling time ie delta_T
14 N=70; //Model Order
15 s=%s;
16 G=sslin('c',1/(5*s+1)/(10*s+1)); //Transfer function
17 t=0:Ts:(N-1)*Ts;
18 u_sim=ones(1,length(t));
19 //There is automatically an input delay of one unit
    in csim function
20 S=csim(u_sim,t,G)'; //generating step response model
    for real plant
21 //plot(t,S);
22 T=80; //simulation time
23
24 //Let the three simulations correspond to
25 //MPC1==> P=3,M=1
26 //MPC2==> P=4,M=2
27 //PID==> The PID controller
28
29 P1=3; M1=1;
30 P2=4; M2=2;
31 S1=S(1:P1); //MPC-1
32 S2=[S(1:P2) [0;S(1:P2-1)]]; //MPC-2
33
34 //SISO system
35 Q=1;
36 R=0; //No move suppression
37
38 Kc1=inv(S1'*Q*S1+R*eye(M1,M1))*S1'*Q; //Eqn20-57
    MPC1
39 Kc2=inv(S2'*Q*S2+R*eye(M2,M2))*S2'*Q; //Eqn20-57
    MPC2
40
41 mprintf('\nFor P=3 and M=1, \nKc=\n      [%f %f %f]\n
    n',Kc1)
42 mprintf('\nFor P=4 and M=2,\nKc='')

```

```

43 disp(Kc2)
44
45 //=====Part (b)=====//
46 //=====Part (b)=====//
47 //=====Part (b)=====//
48 //=====Part (b)=====//
49 //=====Part (b)=====//
50 //=====Part (b)=====//
51
52 //=====Part (b) MPC-1=====//
53 n=T/Ts*2+1; //no. of discrete points in our domain
    of analysis
54 //Input initialization T is the Time for simulation
55 //We take a few u values in negative time to
    facilitate
56 //usage of step response model
57 u=zeros(n,1);
58 d=zeros(n,1);
59 delta_u=zeros(n,1);
60 //delta_u(101+2)=1; //Step change at t=2 min
61 u=cumsum(delta_u);
62 delta_d=zeros(n,1);
63 //delta_d(101+8)=0.15;//disturbance t=8 min
64 d=cumsum(delta_d);
65
66 y=zeros(1,n); //Actual values
67 yhat=zeros(1,n); //predicted value
68 ydot=zeros(P1,n); //Unforced predictions
69 ydottilde=zeros(P1,n); //Corrected unforced
    predictions
70 yr=ones(P1,n); //reference trajectory (same as
    setpoint)
71 edot=zeros(P1,n); //predicted unforced error
72
73 t=-(n-1)/2:Ts:(n-1)/2;
74
75 for k=(n-1)/2+1:n-P1
76

```

```

77 //Unforced predictions
78 for J=1:P1
79     ydot(J,k+1)=S(J+1:N-1)'*flipdim(delta_u(k+J-
        N+1:k-1),1)+S(N)*u(k+J-N);
80 end
81
82 //Actual values of the process
83 J=0;
84 y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1)
        ,1)+S(N)*u(k+J-N)+...
85     S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1)
        ,1)+S(N)*d(k+J-N);
86
87 //Predicted value of the process
88 J=0;
89 yhat(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J
        -1),1)+S(N)*u(k+J-N);
90
91 //Corrected prediction for unforced case
92 ydottilde(:,k+1)=ydot(:,k+1)+ones(P1,1)*(y(k)-
        yhat(k));
93
94 //Predicted unforced error Eqn20-52
95 edot(:,k+1)=yr(:,k+1)-ydottilde(:,k+1);
96
97 //Control move
98 delta_u(k)=Kc1*edot(:,k+1);
99 u(k)=u(k-1)+delta_u(k);
100
101 end
102 subplot(1,2,1);
103 plot(t,y,'black-');
104 subplot(1,2,2);
105 plot2d2(t,u);
106
107 //=====Part(b) MPC-2=====//
108 n=T/Ts*2+1; //no. of discrete points in our domain
        of analysis

```

```

109 //Input initialization T is the Time for simulation
110 //We take a few u values in negative time to
    facilitate
111 //usage of step response model
112 u=zeros(n,1);
113 d=zeros(n,1);
114 delta_u=zeros(n,1);
115 //delta_u(101+2)=1; //Step change at t=2 min
116 u=cumsum(delta_u);
117 delta_d=zeros(n,1);
118 //delta_d(101+8)=0.15;//disturbance t=8 min
119 d=cumsum(delta_d);
120
121 y=zeros(1,n); //Actual values
122 yhat=zeros(1,n); //predicted value
123 ydot=zeros(P2,n); //Unforced predictions
124 ydottilde=zeros(P2,n); //Corrected unforced
    predictions
125 yr=ones(P2,n); //reference trajectory(same as
    setpoint)
126 edot=zeros(P2,n); //predicted unforced error
127
128 t=-(n-1)/2:Ts:(n-1)/2;
129
130 for k=(n-1)/2+1:n-P2
131
132     //Unforced predictions
133     for J=1:P2
134         ydot(J,k+1)=S(J+1:N-1)'*flipdim(delta_u(k+J-
            N+1:k-1),1)+S(N)*u(k+J-N);
135     end
136
137     //Actual values of the process
138     J=0;
139     y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1)
        ,1)+S(N)*u(k+J-N)+...
140         S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1)
            ,1)+S(N)*d(k+J-N);

```

```

141
142 //Predicted value of the process
143 J=0;
144 yhat(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J
      -1),1)+S(N)*u(k+J-N);
145
146 //Corrected prediction for unforced case
147 ydottilde(:,k+1)=ydot(:,k+1)+ones(P2,1)*(y(k)-
      yhat(k));
148
149 //Predicted unforced error Eqn20-52
150 edot(:,k+1)=yr(:,k+1)-ydottilde(:,k+1);
151
152 //Control move
153 delta_u(k)=Kc2(1,:)*edot(:,k+1);
154 u(k)=u(k-1)+delta_u(k);
155
156 end
157 subplot(1,2,1);
158 plot(t,y,'-.');
159 set(gca(),"data_bounds",[0 60 0 1.25]); //putting
      bounds on display
160 l=legend("MPC(P=3,M=1)", "MPC(P=4,M=2)",position=4);
161 xtitle("Process Output", "Time(min)", "$y$");
162 a=get("current_axes");
163 c=a.y_label;c.font_size=5;
164
165 subplot(1,2,2);
166 plot2d2(t,u,style=2);
167
168
169
170 //=====Part(b) PID=====//
171 n=T/Ts*2+1; //no. of discrete points in our domain
      of analysis
172 //Input initialization T is the Time for simulation
173 //We take a few u values in negative time to
      facilitate

```

```

174 //usage of step response model
175 u=zeros(n,1);
176 d=zeros(n,1);
177 delta_u=zeros(n,1);
178 delta_d=zeros(n,1);
179 //delta_d(101+8)=0.15;//disturbance t=8 min
180 d=cumsum(delta_d);
181
182 y=zeros(n,1); //Actual values
183 ysp=1; //setpoint
184 e=zeros(n,1); //error
185 delta_e=zeros(n,1); //error
186
187 t=-(n-1)/2:Ts:(n-1)/2;
188
189 //PID settings
190 Kc=2.27;tauI=16.6;tauD=1.49;
191
192 for k=(n-1)/2+1:n-1
193     //Actual values of the process
194     J=0;
195     y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1)
196         ,1)+S(N)*u(k+J-N)+...
197         S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1)
198         ,1)+S(N)*d(k+J-N);
199
200     //error
201     e(k)=ysp-y(k);
202     delta_e(k)=e(k)-e(k-1);
203
204     //Controller move——Digital PID——Eqn 7-28 Pg
205     //136 (Velocity form)
206     u(k)=u(k-1)+Kc*([delta_e(k,1)+e(k,1)*Ts/tauI+
207         tauD/Ts*(e(k)-2*e(k-1)+e(k-2))]);
208     delta_u(k)=u(k)-u(k-1);
209 end
210 subplot(1,2,1);
211 plot(t,y,'red—');
212 set(gca(),'data_bounds',[0 60 0 1.25]); //putting

```



```

        bounds on display
208 l=legend("MPC (P=3,M=1)", "MPC (P=4,M=2)", "PID
        controller", position=4);
209 xtitle("Process Output", "Time(min)", "$y$");
210 a=get("current_axes");
211 c=a.y_label; c.font_size=5;
212
213 subplot(1,2,2);
214 plot2d2(t,u, style=5);
215 set(gca(), "data_bounds", [0 30 -100 100]); //putting
        bounds on display
216 l=legend("MPC (P=3,M=1)", "MPC (P=4,M=2)", "PID
        controller", position=4);
217 xtitle("Controller Output", "Time(min)", "$u$");
218 a=get("current_axes");
219 c=a.y_label; c.font_size=5;
220
221
222 //=====Part(c)=====//
223 //=====Part(c)=====//
224 //=====Part(c)=====//
225 //=====Part(c)=====//
226 //=====Part(c)=====//
227 //=====Part(c)=====//
228 //=====Part(c)=====//
229
230 //=====Part(c) MPC-1=====//
231 n=T/Ts*2+1; //no. of discrete points in our domain
        of analysis
232 //Input initialization T is the Time for simulation
233 //We take a few u values in negative time to
        facilitate
234 //usage of step response model
235 u=zeros(n,1);
236 d=zeros(n,1);
237 delta_u=zeros(n,1);
238 u=cumsum(delta_u);
239 delta_d=zeros(n,1);

```

```

240 delta_d((n-1)/2+1)=1; //disturbance t=0 min
241 d=cumsum(delta_d);
242
243 y=zeros(1,n); //Actual values
244 yhat=zeros(1,n); //predicted value
245 ydot=zeros(P1,n); //Unforced predictions
246 ydottilde=zeros(P1,n); //Corrected unforced
    predictions
247 yr=zeros(P1,n); //reference trajectory (same as
    setpoint)
248 edot=zeros(P1,n); //predicted unforced error
249
250 t=-(n-1)/2:Ts:(n-1)/2;
251
252 for k=(n-1)/2+1:n-P1
253
254     //Unforced predictions
255     for J=1:P1
256         ydot(J,k+1)=S(J+1:N-1)'*flipdim(delta_u(k+J-
            N+1:k-1),1)+S(N)*u(k+J-N);
257     end
258
259     //Actual values of the process
260     J=0;
261     y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1)
        ,1)+S(N)*u(k+J-N)+...
262         S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1)
            ,1)+S(N)*d(k+J-N);
263
264     //Predicted value of the process
265     J=0;
266     yhat(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J
        -1),1)+S(N)*u(k+J-N);
267
268     //Corrected prediction for unforced case
269     ydottilde(:,k+1)=ydot(:,k+1)+ones(P1,1)*(y(k)-
        yhat(k));
270

```

```

271     //Predicted unforced error   Eqn20-52
272     edot(:,k+1)=yr(:,k+1)-ydottilde(:,k+1);
273
274     //Control move
275     delta_u(k)=Kc1*edot(:,k+1);
276     u(k)=u(k-1)+delta_u(k);
277
278 end
279
280 scf();
281 subplot(1,2,1);
282 plot(t,y,'black-');
283 subplot(1,2,2);
284 plot2d2(t,u);
285
286 //=====Part(c) MPC-2=====//
287 n=T/Ts*2+1; //no. of discrete points in our domain
        of analysis
288 //Input initialization T is the Time for simulation
289 //We take a few u values in negative time to
        facilitate
290 //usage of step response model
291 u=zeros(n,1);
292 d=zeros(n,1);
293 delta_u=zeros(n,1);
294 u=cumsum(delta_u);
295 delta_d=zeros(n,1);
296 delta_d((n-1)/2+1)=1; //disturbance t=0 min
297 d=cumsum(delta_d);
298
299 y=zeros(1,n); //Actual values
300 yhat=zeros(1,n); //predicted value
301 ydot=zeros(P2,n); //Unforced predictions
302 ydottilde=zeros(P2,n); //Corrected unforced
        predictions
303 yr=zeros(P2,n); //reference trajectory (same as
        setpoint)
304 edot=zeros(P2,n); //predicted unforced error

```

```

305
306 t=-(n-1)/2:Ts:(n-1)/2;
307
308 for k=(n-1)/2+1:n-P2
309
310     //Unforced predictions
311     for J=1:P2
312         ydot(J,k+1)=S(J+1:N-1)'*flipdim(delta_u(k+J-
313             N+1:k-1),1)+S(N)*u(k+J-N);
314     end
315
316     //Actual values of the process
317     J=0;
318     y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1)
319         ,1)+S(N)*u(k+J-N)+...
320         S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1)
321         ,1)+S(N)*d(k+J-N);
322
323     //Predicted value of the process
324     J=0;
325     yhat(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-
326         -1),1)+S(N)*u(k+J-N);
327
328     //Corrected prediction for unforced case
329     ydottilde(:,k+1)=ydot(:,k+1)+ones(P2,1)*(y(k)-
330         yhat(k));
331
332     //Predicted unforced error Eqn20-52
333     edot(:,k+1)=yr(:,k+1)-ydottilde(:,k+1);
334
335     //Control move
336     delta_u(k)=Kc2(1,:)*edot(:,k+1);
337     u(k)=u(k-1)+delta_u(k);
338
339 end
340 subplot(1,2,1);
341 plot(t,y,'-');
342 subplot(1,2,2);

```

```

338 plot2d2(t,u,style=2);
339
340
341
342 //=====Part(C) PID=====//
343 n=T/Ts*2+1; //no. of discrete points in our domain
      of analysis
344 //Input initialization T is the Time for simulation
345 //We take a few u values in negative time to
      facilitate
346 //usage of step response model
347
348 u=zeros(n,1);
349 d=zeros(n,1);
350 delta_u=zeros(n,1);
351 u=cumsum(delta_u);
352 delta_d=zeros(n,1);
353 delta_d((n-1)/2+1)=1; //disturbance t=0 min
354 d=cumsum(delta_d);
355
356 y=zeros(n,1); //Actual values
357 ysp=0; //setpoint
358 e=zeros(n,1); //error
359 delta_e=zeros(n,1); //error
360
361 t=-(n-1)/2:Ts:(n-1)/2;
362
363 //PID settings
364 Kc=3.52; tauI=6.98; tauD=1.73;
365
366 for k=(n-1)/2+1:n-1
367     //Actual values of the process
368     J=0;
369     y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1)
      ,1)+S(N)*u(k+J-N)+...
370     S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1)
      ,1)+S(N)*d(k+J-N);
371     //error

```

```

372     e(k)=ysp-y(k);
373     delta_e(k)=e(k)-e(k-1);
374
375     // Controller move——Digital PID——Eqn 7-28 Pg
        136 (Velocity form)
376     u(k)=u(k-1)+Kc*([delta_e(k,1)+e(k,1)*Ts/taui+
        tauD/Ts*(e(k)-2*e(k-1)+e(k-2))]);
377     delta_u(k)=u(k)-u(k-1);
378 end
379 subplot(1,2,1);
380 plot(t,y,'red—');
381 set(gca(),'data_bounds',[0 60 -0.1 0.3]); //putting
        bounds on display
382 l=legend("MPC (P=3,M=1)", "MPC (P=4,M=2)", "PID
        controller",position=1);
383 xtitle("Process Output", "Time(min)", "$y$");
384 a=get("current_axes");
385 c=a.y_label;c.font_size=5;
386
387 subplot(1,2,2);
388 plot2d2(t,u,style=5);
389 set(gca(),'data_bounds',[0 30 -1.5 0]); //putting
        bounds on display
390 l=legend("MPC (P=3,M=1)", "MPC (P=4,M=2)", "PID
        controller",position=1);
391 xtitle("Controller Output", "Time(min)", "$u$");
392 a=get("current_axes");
393 c=a.y_label;c.font_size=5;

```

---

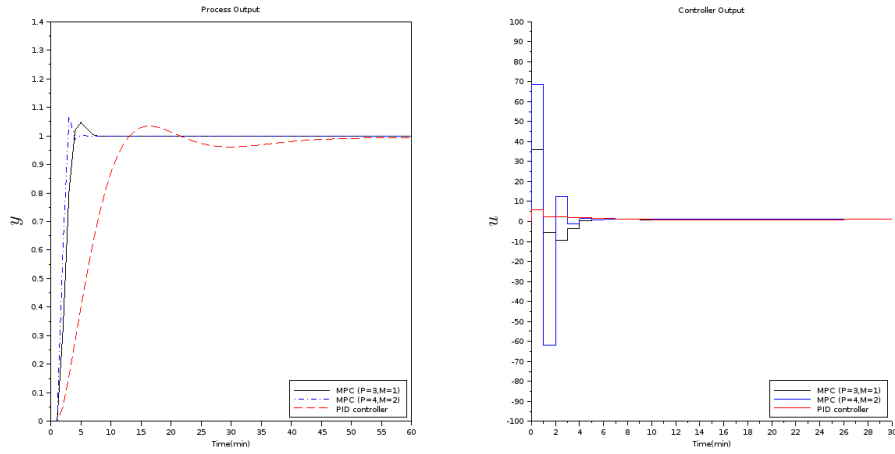


Figure 20.5: Comparison of MPCs and PID

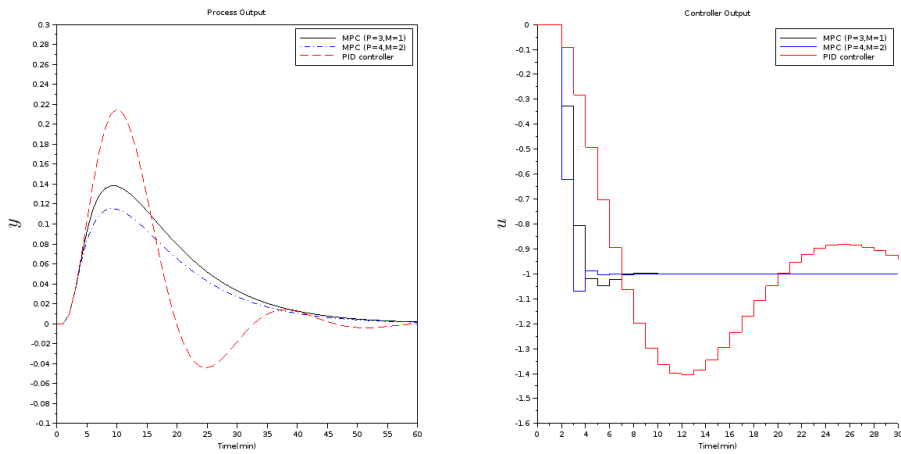


Figure 20.6: Comparison of MPCs and PID

# Chapter 21

## Process Monitoring

Scilab code Exa 21.2 Semiconductor processing control charts

```
1 clear
2 clc
3
4 //Example 21.2
5 disp('Example 21.2 ')
6
7 //data
8 x=[209.6      207.6      211.1
9      183.5      193.1      202.4
10     190.1      206.8      201.6
11     206.9      189.3      204.1
12     260.      209.      212.2
13     193.9      178.8      214.5
14     206.9      202.8      189.7
15     200.2      192.7      202.1
16     210.6      192.3      205.9
17     186.6      201.5      197.4
18     204.8      196.6      225.
19     183.7      209.7      208.6
```



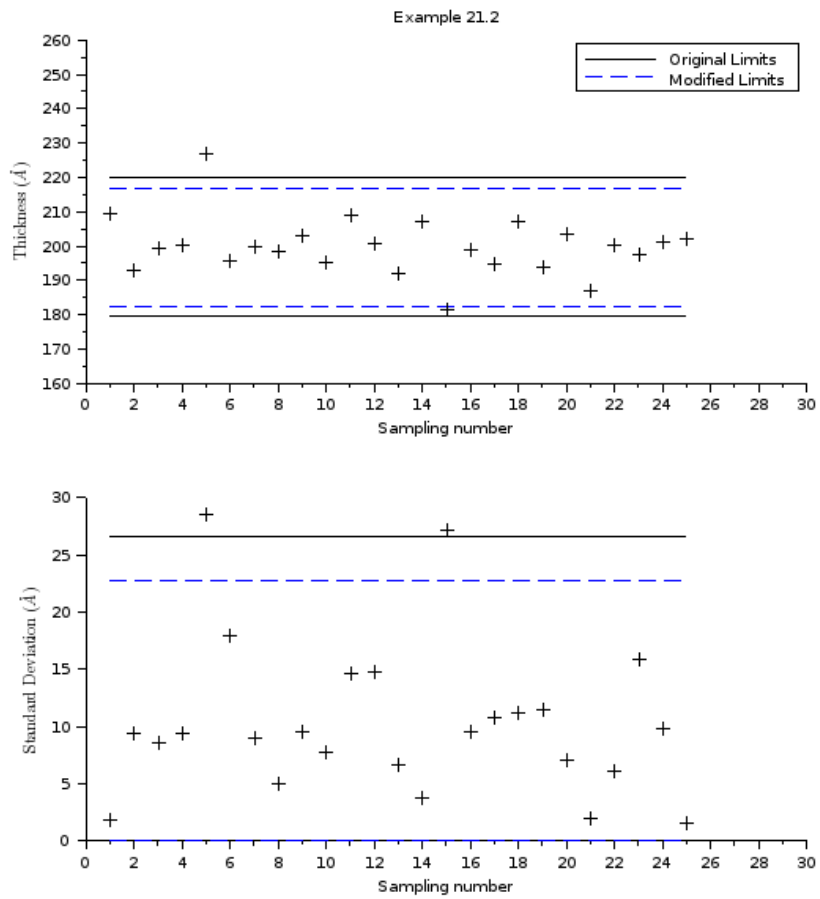


Figure 21.1: Semiconductor processing control charts

```

20      185.6      198.9      191.5
21      202.9      210.1      208.1
22      198.6      195.2      150.
23      188.7      200.7      207.6
24      197.1      204.      182.9
25      194.2      211.2      215.4
26      191.      206.2      183.9
27      202.5      197.1      211.1
28      185.1      186.3      188.9
29      203.1      193.1      203.9
30      179.7      203.3      209.7
31      205.3      190.      208.2
32      203.4      202.9      200.4 ]
33
34
35 //Original Limits
36 n=3;
37 xbar=sum(x,2)/n; //mean calculation
38 s=sqrt(1/(n-1)*sum((x-repmat(xbar,1,3)).^2,2)); //
    standard deviation calculation
39 p=length(xbar); //no. of subgroups
40 xbarbar=sum(xbar,1)/p;
41 sbar=sum(s,1)/p;
42
43 c4=0.8862; B3=0; B4=2.568; c=3;
44 sigma=1/c4*sbar/sqrt(n);
45 //original limits
46 UCL_x=xbarbar+c*sigma; //Eqn21-9
47 LCL_x=xbarbar-c*sigma; //Eqn 21-10
48
49 UCL_s=B4*sbar; //Eqn21-14
50 LCL_s=B3*sbar; //Eqn21-15
51
52 //Modified Limits
53 x_mod=x;
54 x_mod([5 15],:)=[];
55 n=3;
56 xbar_mod=sum(x_mod,2)/n; //mean calculation

```

```

57 s_mod=sqrt(1/(n-1)*sum((x_mod-repmat(xbar_mod,1,3))
    .^2,2)); //standard deviation calculation
58 p_mod=length(xbar_mod);//no. of subgroups
59 xbarbar_mod=sum(xbar_mod,1)/p_mod;
60 sbar_mod=sum(s_mod,1)/p_mod;
61
62 c4=0.8862;B3=0;B4=2.568;c=3;
63 sigma_mod=1/c4*sbar_mod/sqrt(n);
64 //modified limits
65 UCL_x_mod=xbarbar_mod+c*sigma_mod;//Eqn21-9
66 LCL_x_mod=xbarbar_mod-c*sigma_mod;//Eqn 21-10
67
68 UCL_s_mod=B4*sbar_mod;//Eqn21-14
69 LCL_s_mod=B3*sbar_mod;//Eqn21-15
70
71
72
73 mprintf('\n                Original Limits
    Modified Limits')
74 mprintf('\n xbar Chart Control Limits ')
75 mprintf('\n UCL                %f                %f',UCL_x,
    UCL_x_mod)
76 mprintf('\n LCL                %f                %f',LCL_x,
    LCL_x_mod)
77 mprintf('\n s Chart Control Limits ')
78 mprintf('\n UCL                %f                %f',UCL_s,
    UCL_s_mod)
79 mprintf('\n LCL                %f                %f',LCL_s
    ,LCL_s_mod)
80
81 subplot(2,1,1);
82 plot2d(repmat(UCL_x,1,p));
83 plot(repmat(UCL_x_mod,1,p),'--');
84 plot2d(repmat(LCL_x,1,p));
85 plot(repmat(LCL_x_mod,1,p),'--');
86 plot2d(xbar,style=-1,rect=[0,160,30,260])
87 xttitle('Example 21.2','Sampling number','$\text{
    Thickness}\ (\AA)$')

```

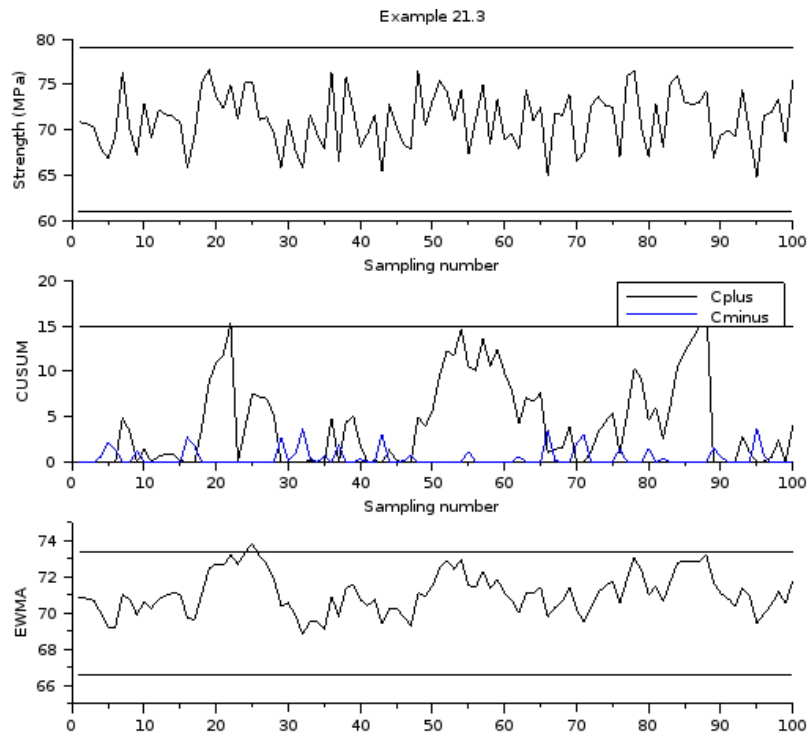


Figure 21.2: Shewart CUSUM EWMA comparison

```

88 legend('Original Limits','Modified Limits')
89
90 subplot(2,1,2);
91 plot2d(repmat(UCL_s,1,p));
92 plot2d(repmat(LCL_s,1,p));
93 plot(repmat(UCL_s_mod,1,p),'--');
94 plot(repmat(LCL_s_mod,1,p),'--');
95 plot2d(s,style=-1,rect=[0,0,30,30])
96 xtitle('','Sampling number','$\text{Standard Deviation}\ (\AA)$')

```

---

### Scilab code Exa 21.3 Shewart CUSUM EWMA comparison

```
1 clear
2 clc
3
4 //Example 21.3
5 disp('Example 21.3')
6
7
8 T=70;sigma=3;p=100;//p is the no. of samples
9 x=grand(p,1,"nor",T,sigma);
10 delta=0.5*sigma;
11 x(11:$)=x(11:$)+delta;
12
13 //Limits for Shewart charts
14 UCL_1=T+sigma*3;
15 LCL_1=T-sigma*3;
16
17 subplot(3,1,1);
18 plot2d(repmat(UCL_1,1,p));
19 plot2d(repmat(LCL_1,1,p));
20 plot2d(x,style=1,rect=[0,60,100,80])
21 xtitle('Example 21.3','Sampling number','Strength (
    MPa)')
22
23 //CUSUM
24 Cplus=zeros(100,1);Cminus=zeros(100,1);
25 K=0.5*sigma;H=5*sigma;
26 UCL_2=H;
27
28 for k=2:100
29     Cplus(k)=max(0,x(k)-(T+K)+Cplus(k-1));
30     Cminus(k)=max(0,(T-K)-x(k)+Cminus(k-1));
31     if Cplus(k-1)>H then
32         Cplus(k)=0;
33     end
34     if Cminus(k-1)>H then
35         Cminus(k)=0;
```

```

36     end
37
38 end
39
40
41 subplot(3,1,2);
42 plot2d(Cplus,style=1,rect=[0,0,100,20]);
43 plot2d(Cminus,style=2,rect=[0,0,100,20]);
44 plot2d(repmat(UCL_2,1,p));
45 xtitle('', 'Sampling number', 'CUSUM')
46 legend('Cplus', 'Cminus')
47
48 //EWMA
49 lamda=0.25;
50 z=x;
51 for k=2:100
52     z(k)=lamda*x(k)+(1-lamda)*z(k-1);
53 end
54 UCL_3=T+3*sigma*sqrt(lamda/(2-lamda));
55 LCL_3=T-3*sigma*sqrt(lamda/(2-lamda));
56
57 subplot(3,1,3);
58 plot2d(repmat(UCL_3,1,p));
59 plot2d(repmat(LCL_3,1,p));
60 plot2d(z,style=1,rect=[0,65,100,75])
61 xtitle('', 'Sampling number', 'EWMA')
62
63
64 mprintf('The charts in the example and in the book
        differ due\n...
65 a different realization of data everytime the code
        is run\n...
66 due to the grand command. If we had the exact data
        as that given\n...
67 in the book our charts would have matched.')

```

---

### Scilab code Exa 21.4 Process Capability Indices

```
1
2 clear
3 clc
4
5 //Example 21.4
6 disp('Example 21.4')
7
8 xbar=199.5; //Note that this is the correct value and
   not 199
9 sbar=8.83;
10 USL=235; //Note that this is diff from UCL
11 LSL=185;
12 c4=0.8862;
13 n=3;
14 sigma=5.75;
15 sigma_x=sbar/c4/sqrt(n);
16
17 mprintf('\nValue of sigma_x=%f',sigma_x);
18
19 Cp=(USL-LSL)/6/sigma;
20 Cpk=min(xbar-LSL,USL-xbar)/3/sigma;
21
22 mprintf('\nCp=%f and Cpk=%f',Cp,Cpk)
```

---

### Scilab code Exa 21.5 Effluent Stream from wastewater treatment

```
1 clear
2 clc
3
4 //Example 21.5
```

```

5 disp('Example 21.5 ')
6
7 //data
8 x=[ 17.7      1380.
9      23.6      1458.
10     13.2      1322.
11     25.2      1448.
12     13.1      1334.
13     27.8      1485.
14     29.8      1503.
15     9.        1540.
16     14.3      1341.
17     26.        1448.
18     23.2      1426.
19     22.8      1417.
20     20.4      1384.
21     17.5      1380.
22     18.4      1396.
23     16.8      1345.
24     13.8      1349.
25     19.4      1398.
26     24.7      1426.
27     16.8      1361.
28     14.9      1347.
29     27.6      1476.
30     26.1      1454.
31     20.        1393.
32     22.9      1427.
33     22.4      1431.
34     19.6      1405.
35     31.5      1521.
36     19.9      1409.
37     20.3      1392.];
38
39
40 T=mean(x, 'r ');
41 s=sqrt(variance(x, 'r '));
42

```



```

43 UCL=T+3*s;
44 LCL=T-3*s;
45
46 p=size(x,1)
47
48 subplot(2,1,1);
49 plot2d(repmat(UCL(1),1,p));
50 plot2d(repmat(LCL(1),1,p));
51 plot2d(repmat(T(1),1,p));
52 plot2d(x(:,1),style=-1,rect=[0,0,32,40])
53 xtitle('Example 21.4','Sampling number','BOD (mg/L)')
    )
54
55
56 subplot(2,1,2);
57 plot2d(repmat(UCL(2),1,p));
58 plot2d(repmat(LCL(2),1,p));
59 plot2d(repmat(T(2),1,p));
60 plot2d(x(:,2),style=-1,rect=[0,1200,32,1600])
61 xtitle('', 'Sampling number', 'Solids (mg/L)')
62
63 //subplot(3,1,3);
64 scf()
65 plot2d(x(8,1),x(8,2),style=-3,rect=[0,1200,40,1600])
66 plot2d(x(:,1),x(:,2),style=-1,rect=[0,1200,40,1600])
67 legend("Sample #8",position=4)
68 xtitle('', 'BOD (mg/L)', 'Solids (mg/L)')
69
70 mprintf('\n\nThe confidence interval for third case is
    not drawn\n\n...
    because it is beyond the scope of this book')

```

---

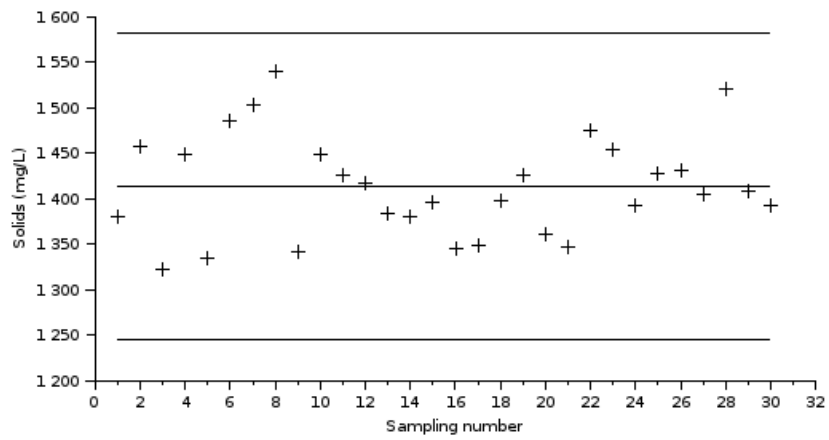
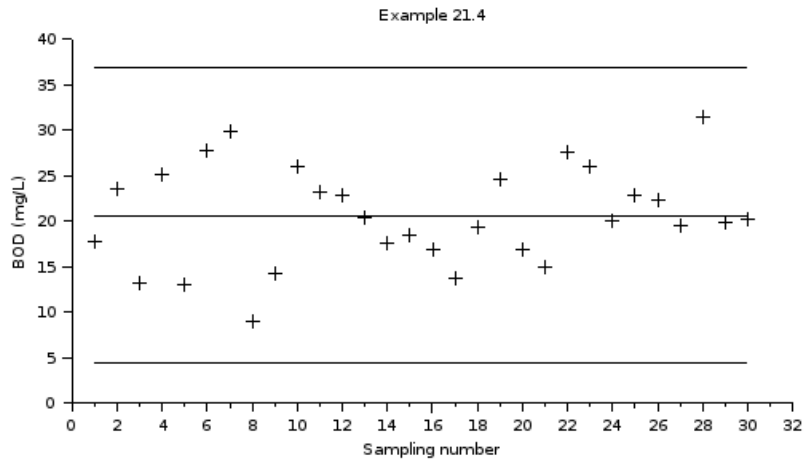


Figure 21.3: Effluent Stream from wastewater treatment

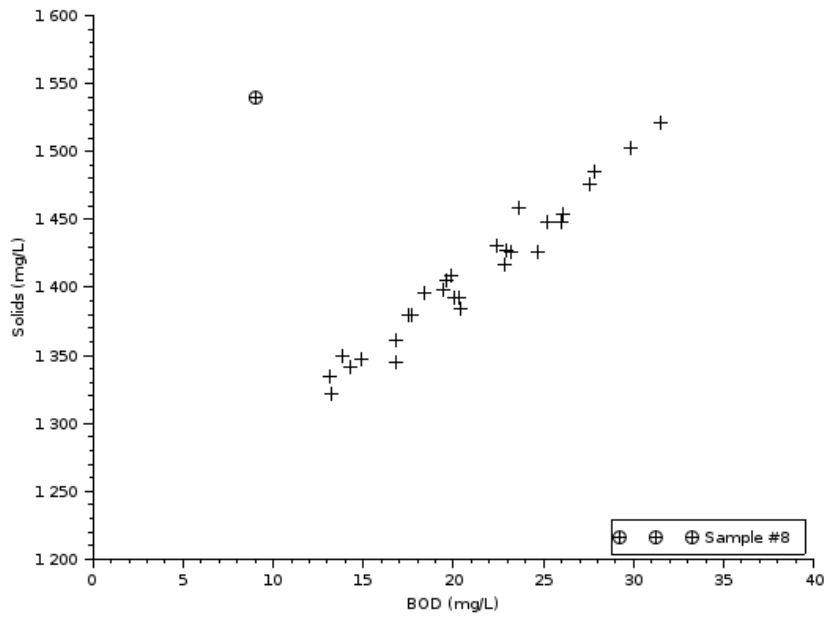


Figure 21.4: Effluent Stream from wastewater treatment

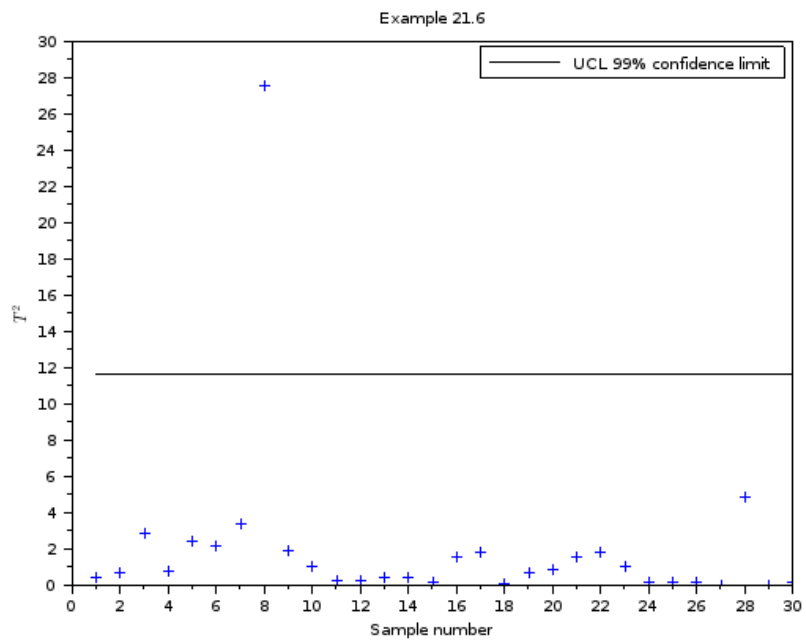


Figure 21.5: Hotellings T square statistic

### Scilab code Exa 21.6 Hotellings T square statistic

```
1 clear
2 clc
3
4 //Example 21.6
5 disp('Example 21.6 ')
6
7 //data
8 x=[ 17.7      1380.
9      23.6      1458.
10     13.2      1322.
11     25.2      1448.
12     13.1      1334.
13     27.8      1485.
14     29.8      1503.
15     9.        1540.
16     14.3      1341.
17     26.       1448.
18     23.2      1426.
19     22.8      1417.
20     20.4      1384.
21     17.5      1380.
22     18.4      1396.
23     16.8      1345.
24     13.8      1349.
25     19.4      1398.
26     24.7      1426.
27     16.8      1361.
28     14.9      1347.
29     27.6      1476.
30     26.1      1454.
31     20.       1393.
32     22.9      1427.
```

```

33     22.4     1431.
34     19.6     1405.
35     31.5     1521.
36     19.9     1409.
37     20.3     1392.];
38
39
40 n=1;
41 N=size(x,1);
42 T=mean(x,'r');
43 //For our example n=1 because each measurement is a
    subgroup
44 S=mvcov(x);
45 //Note that mvcov calculates covariance with
    denominator N, while
46 //variance calculates with denominator N-1, hence
    diagonal entry of mvcov does not
47 //match with variance calculated manually for each
    vector
48 //As per wikipedia the book is wrong and for
    covariance matrix we should
49 //use N-1 but here we follow the book
50 Tsquare=zeros(N,1);
51 for k=1:N
52     Tsquare(k)=n*(x(k,:)-T)*inv(S)*(x(k,:)-T)';
53 end
54
55 UCL=11.63;
56
57 plot(repmat(UCL,1,N),color='black');
58 plot(Tsquare,'+')
59 legend("UCL 99% confidence limit")
60 xtitle("Example 21.6","Sample number","$T^2$")

```

---

# Chapter 22

## Biosystems Control Design

Scilab code Exa 22.1 Fermentor

```
1 clear
2 clc
3
4 //Example 22.1
5 disp('Example 22.1 ')
6
7 //Parameters
8 Yxs=0.4; B=0.2; Pm=50; Ki=22;
9 a=2.2; mu_m=0.48; Km=1.2; Sf=20;
10
11
12 //ODE model
13 function ydot=model(t,y,D)
14     X=y(1); S=y(2); P=y(3);
15
16     Xdot=-D*X+mu(S,P)*X;
17     Sdot=D*(Sf-S)-1/Yxs*mu(S,P)*X;
18     Pdot=-D*P+[a*mu(S,P)+B]*X
19
```

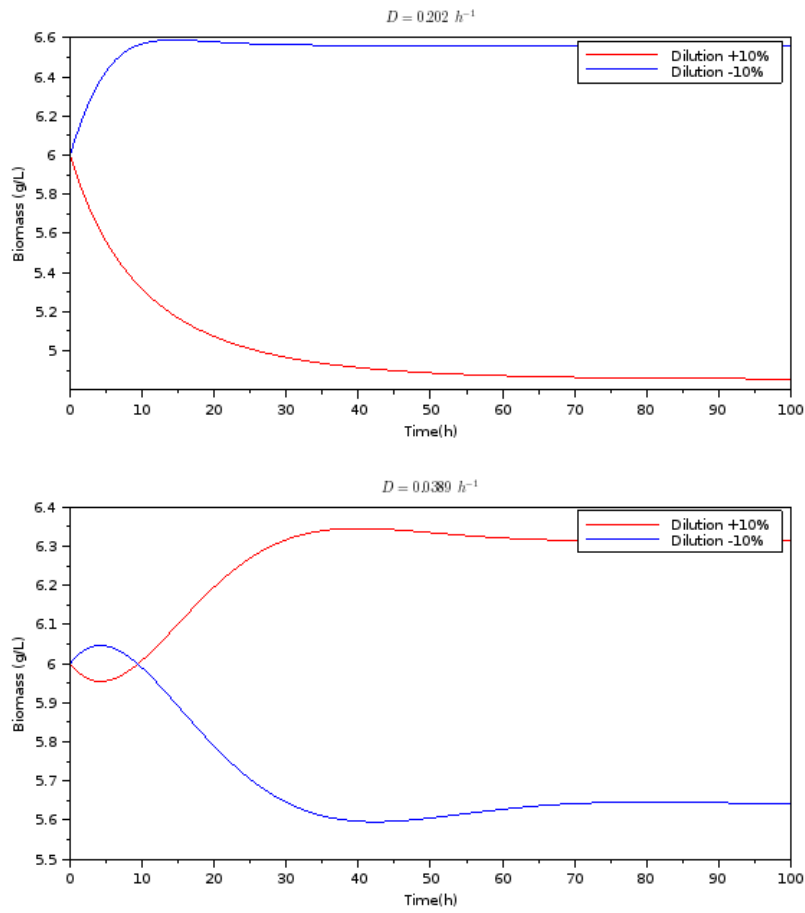


Figure 22.1: Fermentor



```

20     ydot=[Xdot Sdot Pdot]';
21 endfunction
22
23 //Rate law
24 function mu=mu(S,P)
25     mu=mu_m*(1-P/Pm)*S/(Km+S+S^2/Ki);
26 endfunction
27
28 t=0:0.1:100;t0=0;
29 y0=[6 5 19.14]';//Initial stable condition
30
31 D=0.202*1.1;//10% increase
32 y_up = ode(y0, t0, t, list(model,D))
33 D=0.202*0.9;//10% decrease
34 y_down = ode(y0, t0, t, list(model,D))
35
36 subplot(2,1,1);
37 plot(t,y_up(1,:),color='red');
38 plot(t,y_down(1,:));
39 xtitle("$D=0.202\ h^{-1}$", "Time(h)", "Biomass (g/L)"
40 )
41 legend("Dilution +10%", "Dilution -10%")
42
43 subplot(2,1,2);
44 t=0:0.1:100;t0=0;
45 y0=[6 5 44.05]';//Initial stable condition
46 D=0.0389*1.1;//10% increase
47 y_up = ode(y0, t0, t, list(model,D))
48 D=0.0389*0.9;//10% decrease
49 y_down = ode(y0, t0, t, list(model,D))
50
51 plot(t,y_up(1,:),color='red');
52 plot(t,y_down(1,:));
53 xtitle("$D=0.0389\ h^{-1}$", "Time(h)", "Biomass (g/L)"
54 );
55 legend("Dilution +10%", "Dilution -10%")

```

---

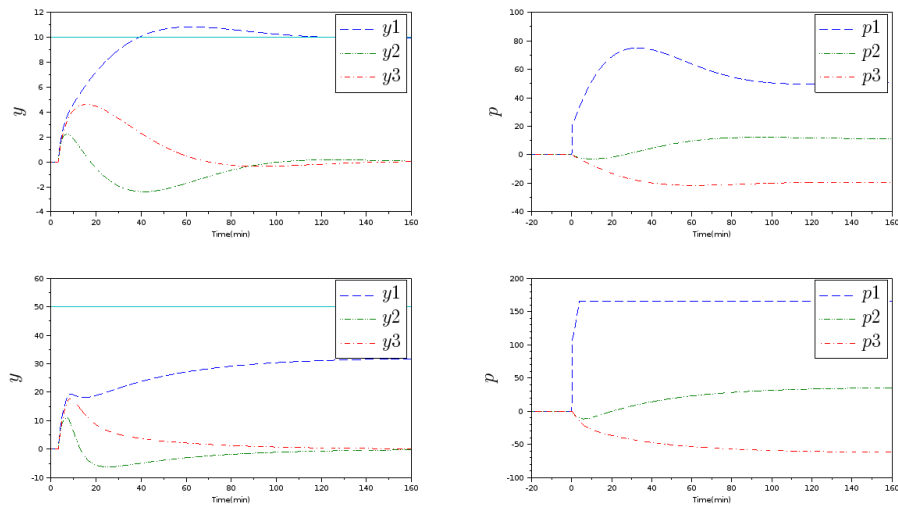


Figure 22.2: Granulation process control

### Scilab code Exa 22.2 Granulation process control

```

1 clear
2 clc
3
4 //Example 22.2
5 disp('Example 22.2')
6 //Author: Dhruv Gupta....Aug 4, 2013
7 //<dgupta6@wisc.edu>
8
9 K=[0.2 0.58 0.35;0.25 1.10 1.3;0.3 0.7 1.2];
10 tau=[2 2 2;3 3 3;4 4 4];
11 s=%s;
12
13 G=K./(1+tau*s);

```

```

14
15 RGA=K.*inv(K');
16 disp(RGA,"RGA=")
17
18 //IMC based tuning
19 tauC=5;
20 Kc=diag(tau/tauC./K);
21 mprintf("\n\nThe tauI given in book are wrong\n...
22 refer to Table 11.1 for correct formula\n\n")
23 tauI=diag(tau)+1;
24 mprintf('\nWe still however use the ones given in
      book\n');
25
26
27 disp(Kc,"Kc=")
28 disp(tauI,"tauI=")
29 //Refer to Eqns 15-23 and 15-24
30 Gc=Kc.*(1+(1)./tauI/s);
31 //For the sake of brevity we write Gstar as G
32 //We will account for delays in the for loop that we
      will write
33 //Refer to Figure 15.9 Page 295 for details of Smith
      Predictor
34
35
36 //====Making step response models of the continuous
      transfer functions====//
37 Ts=0.1;//Sampling time ie delta_T
38 delay=3/Ts;
39 N=150/Ts;//Model Order
40 s=%s;
41 G=syslin('c',diag(matrix(G,1,9)));//Transfer
      function
42 t=0:Ts:N*Ts;
43 u_sim=ones(9,length(t));
44 //u_sim(:,1:4)=zeros(9,4); //input delay to account
      for 3 min delay in G
45 S=csim(u_sim,t,G)';//generating step response model

```

```

    for real plant
46 //plot(t,S);
47 S(1,:)=[];
48 //Now we have these step response models for each of
    the transfer functions
49 //[S1 S4 S7
50 //S2 S5 S8
51 //S3 S6 S9]
52
53 T=150+delay;//Simulation Run Time in minutes(we add
    delay because our for loop runs till n-delay)
54 n=T/Ts*2+1; //no. of discrete points in our domain
    of analysis
55 //Input initialization T is the Time for simulation
56
57 //=====Set point as 10=====//
58 //p is the controller output
59 p=zeros(n,3);
60 delta_p=zeros(n,3);
61 ytilde=zeros(n,3); //Prediction of Smith Fig 15.9
62 e=zeros(n,3); //corrections
63 edash=zeros(n,3);
64 delta_edash=zeros(n,3);
65 ysp=zeros(n,3);
66 ysp((n-1)/2+1:n,1)=10*ones(n-((n-1)/2+1)+1,1);
67
68 t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
69 y=zeros(n,3);
70
71 for k=(n-1)/2+1:n-delay
72
73     //Error e
74     e(k,:)=ysp(k-1,:)-y(k-1,:);
75
76     //Error edash
77     edash(k,:)=e(k-1,:)-ytilde(k-1,:)+ytilde(k-1-
        delay,:);
78     //Edash=E-(Y1-Y2)... where Y2 is delayed Y1

```

```

79     delta_edash(k,:) = edash(k,:) - edash(k-1,:);
80
81     // Controller calculation — Digital PID — Eqn
      7-28 Pg 136 (Velocity form)
82     p(k,:) = p(k-1,:) + [delta_edash(k,:) + edash(k,:) *
      diag(Ts./tauI)] * diag(Kc);
83
84     // Limits on manipulated variables
85     p(k,:) = min([(345-180)*ones(1,3); p(k,:)], 'r');
86     p(k,:) = max([(105-180)*ones(1,3); p(k,:)], 'r');
87
88     delta_p(k,:) = p(k,:) - p(k-1,:);
89
90
91     // Prediction
92     ytilde(k,1) = [S(1:N-1,1); S(1:N-1,4); S(1:N-1,7)
      ]' ...
93     * [flipdim(delta_p(k-N+1:k-1,1),1); flipdim(
      delta_p(k-N+1:k-1,2),1); flipdim(delta_p(k
      -N+1:k-1,3),1)] ...
94     + [S(N,1) S(N,4) S(N,7)] * [p(k-N,1); p(k-N,2); p
      (k-N,3)];
95     ytilde(k,2) = [S(1:N-1,2); S(1:N-1,5); S(1:N-1,8)
      ]' ...
96     * [flipdim(delta_p(k-N+1:k-1,1),1); flipdim(
      delta_p(k-N+1:k-1,2),1); flipdim(delta_p(k
      -N+1:k-1,3),1)] ...
97     + [S(N,2) S(N,5) S(N,8)] * [p(k-N,1); p(k-N,2); p
      (k-N,3)];
98     ytilde(k,3) = [S(1:N-1,3); S(1:N-1,6); S(1:N-1,9)
      ]' ...
99     * [flipdim(delta_p(k-N+1:k-1,1),1); flipdim(
      delta_p(k-N+1:k-1,2),1); flipdim(delta_p(k
      -N+1:k-1,3),1)] ...
100    + [S(N,3) S(N,6) S(N,9)] * [p(k-N,1); p(k-N,2); p
      (k-N,3)];
101
102    // Output

```

```

103     y(k+delay ,1)=[S(1:N-1,1);S(1:N-1,4);S(1:N-1,7)
104         ]'...
105         * [flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
106             delta_p(k-N+1:k-1,2),1);flipdim(delta_p(k
107                 -N+1:k-1,3),1)]...
108         +[S(N,1) S(N,4) S(N,7)]*[p(k-N,1);p(k-N,2);p
109             (k-N,3)];
110     y(k+delay ,2)=[S(1:N-1,2);S(1:N-1,5);S(1:N-1,8)
111         ]'...
112         * [flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
113             delta_p(k-N+1:k-1,2),1);flipdim(delta_p(k
114                 -N+1:k-1,3),1)]...
115         +[S(N,2) S(N,5) S(N,8)]*[p(k-N,1);p(k-N,2);p
116             (k-N,3)];
117     y(k+delay ,3)=[S(1:N-1,3);S(1:N-1,6);S(1:N-1,9)
118         ]'...
119         * [flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
120             delta_p(k-N+1:k-1,2),1);flipdim(delta_p(k
121                 -N+1:k-1,3),1)]...
122         +[S(N,3) S(N,6) S(N,9)]*[p(k-N,1);p(k-N,2);p
123             (k-N,3)];
124     end
125
126     subplot(2,2,1);
127     plot(t',y(:,1),'--',t',y(:,2),':',t',y(:,3),'-.',t',
128         ysp(:,1),'-');
129     set(gca(),"data_bounds",[0 150 -4 12]); //putting
130         bounds on display
131     l=legend("$y1$", "$y2$", "$y3$",position=1);
132     l.font_size=5;
133     xtitle("","Time(min)","$y$");
134     a=get("current_axes");
135     c=a.y_label;c.font_size=5;
136
137     subplot(2,2,2);
138     plot(t',p(:,1),'--',t',p(:,2),':',t',p(:,3),'-.'');

```

```

127 set(gca(),"data_bounds",[-1 150 -40 100]); //putting
    bounds on display
128 l=legend("$p1$","$p2$","$p3$",position=1);
129 l.font_size=5;
130 xtitle("", "Time(min)", "$p$");
131 a=get("current_axes");
132 c=a.y_label;c.font_size=5;
133
134 mprintf("Note that there is no overshoot around time
    =25 mins \n...
135 which is in contrast to what is shown in book")
136
137
138 //=====Now for set point as 50=====//
139
140 //p is the controller output
141 p=zeros(n,3);
142 delta_p=zeros(n,3);
143 ytilde=zeros(n,3); //Prediction of Smith Fig 15.9
144 e=zeros(n,3); //corrections
145 edash=zeros(n,3);
146 delta_edash=zeros(n,3);
147 ysp=zeros(n,3);
148 ysp((n-1)/2+1:n,1)=50*ones(n-((n-1)/2+1)+1,1);
149
150 t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
151 y=zeros(n,3);
152
153 for k=(n-1)/2+1:n-delay
154
155     //Error e
156     e(k,:)=ysp(k-1,:)-y(k-1,:);
157
158     //Error edash
159     edash(k,:)=e(k-1,:)-ytilde(k-1,:)+ytilde(k-1-
        delay,:);
160     //Edash=E-(Y1-Y2) ... where Y2 is delayed Y1
161     delta_edash(k,:)=edash(k,:)-edash(k-1,:);

```

```

162
163 // Controller calculation —— Digital PID —— Eqn
      7-28 Pg 136 (Velocity form)
164 p(k,:) = p(k-1,:) + [delta_edash(k,:) + edash(k,:)] *
      diag(Ts./tauI)] * diag(Kc);
165
166 // Limits on manipulated variables
167 p(k,:) = min([(345-180)*ones(1,3); p(k,:)], 'r');
168 p(k,:) = max([(105-180)*ones(1,3); p(k,:)], 'r');
169
170 delta_p(k,:) = p(k,:) - p(k-1,:);
171
172
173 // Prediction
174 ytilde(k,1) = [S(1:N-1,1); S(1:N-1,4); S(1:N-1,7)
      ]', ...
175 * [flipdim(delta_p(k-N+1:k-1,1),1); flipdim(
      delta_p(k-N+1:k-1,2),1); flipdim(delta_p(k
      -N+1:k-1,3),1)] ...
176 + [S(N,1) S(N,4) S(N,7)] * [p(k-N,1); p(k-N,2); p
      (k-N,3)];
177 ytilde(k,2) = [S(1:N-1,2); S(1:N-1,5); S(1:N-1,8)
      ]', ...
178 * [flipdim(delta_p(k-N+1:k-1,1),1); flipdim(
      delta_p(k-N+1:k-1,2),1); flipdim(delta_p(k
      -N+1:k-1,3),1)] ...
179 + [S(N,2) S(N,5) S(N,8)] * [p(k-N,1); p(k-N,2); p
      (k-N,3)];
180 ytilde(k,3) = [S(1:N-1,3); S(1:N-1,6); S(1:N-1,9)
      ]', ...
181 * [flipdim(delta_p(k-N+1:k-1,1),1); flipdim(
      delta_p(k-N+1:k-1,2),1); flipdim(delta_p(k
      -N+1:k-1,3),1)] ...
182 + [S(N,3) S(N,6) S(N,9)] * [p(k-N,1); p(k-N,2); p
      (k-N,3)];
183
184 // Output
185 y(k+delay,1) = [S(1:N-1,1); S(1:N-1,4); S(1:N-1,7)

```



```

    ]',...
186     *flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
        delta_p(k-N+1:k-1,2),1);flipdim(delta_p(k
        -N+1:k-1,3),1)]...
187     +[S(N,1) S(N,4) S(N,7)]*[p(k-N,1);p(k-N,2);p
        (k-N,3)];
188     y(k+delay,2)=[S(1:N-1,2);S(1:N-1,5);S(1:N-1,8)
        ]',...
189     *flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
        delta_p(k-N+1:k-1,2),1);flipdim(delta_p(k
        -N+1:k-1,3),1)]...
190     +[S(N,2) S(N,5) S(N,8)]*[p(k-N,1);p(k-N,2);p
        (k-N,3)];
191     y(k+delay,3)=[S(1:N-1,3);S(1:N-1,6);S(1:N-1,9)
        ]',...
192     *flipdim(delta_p(k-N+1:k-1,1),1);flipdim(
        delta_p(k-N+1:k-1,2),1);flipdim(delta_p(k
        -N+1:k-1,3),1)]...
193     +[S(N,3) S(N,6) S(N,9)]*[p(k-N,1);p(k-N,2);p
        (k-N,3)];
194 end
195
196
197 subplot(2,2,3);
198 plot(t',y(:,1),'--',t',y(:,2),':',t',y(:,3),'-.',t',
        ysp(:,1),'-');
199 set(gca(),"data_bounds",[0 150 -10 60]); //putting
        bounds on display
200 l=legend("$y1$","$y2$","$y3$",position=1);
201 l.font_size=5;
202 xtitle("", "Time(min)", "$y$");
203 a=get("current_axes");
204 c=a.y_label;c.font_size=5;
205
206
207 subplot(2,2,4);
208 plot(t',p(:,1),'--',t',p(:,2),':',t',p(:,3),'-.'');
209 set(gca(),"data_bounds",[-1 150 -100 200]); //

```

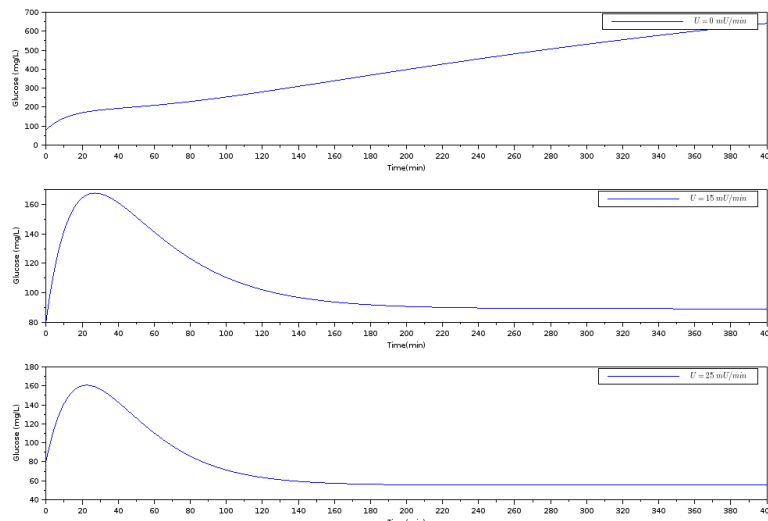


Figure 22.3: Type 1 Diabetes

```

    putting bounds on display
210 l=legend(" $p1$ ", " $p2$ ", " $p3$ ", position=1);
211 l.font_size=5;
212 xtitle(" ", " Time(min) ", " $p$ ");
213 a=get(" current_axes ");
214 c=a.y_label;c.font_size=5;

```

---

### Scilab code Exa 22.3 Type 1 Diabetes

```

1
2 clear
3 clc
4
5 //Example 22.3
6 disp('Example 22.3 ')
7

```

```

8 //Parameters
9 p1=0.028735;p2=0.028344;p3=5.035E-5;V1=12;n=0.0926;
10 Ib=15;//basal
11 Gb=81;
12
13 //Diet function
14 function D=D(t)
15     D=9*exp(-0.05*t);
16 endfunction
17
18
19 //ODE model
20 function ydot=model(t,y,U)
21     G=y(1);X=y(2);I=y(3);
22     Gdot=-p1*G-X*(G+Gb)+D(t);
23     Xdot=-p2*X+p3*I;
24     Idot=-n*(I+Ib)+U/V1;
25     ydot=[Gdot Xdot Idot]';
26 endfunction
27
28
29 t=0:0.1:400;t0=0;
30 y0=[0 0 0]';//G,X,I are deviation variables
31
32 U=0;
33 y = Gb+ode(y0, t0, t, list(model,U))
34 subplot(3,1,1);
35 plot(t,y(1,:));
36 xtitle("", "Time(min)", "Glucose (mg/L)")
37 legend("$U=0\ mU/min$")
38
39 U=15;
40 y =Gb+ ode(y0, t0, t, list(model,U))
41 subplot(3,1,2);
42 plot(t,y(1,:));
43 xtitle("", "Time(min)", "Glucose (mg/L)")
44 legend("$U=15\ mU/min$")
45

```

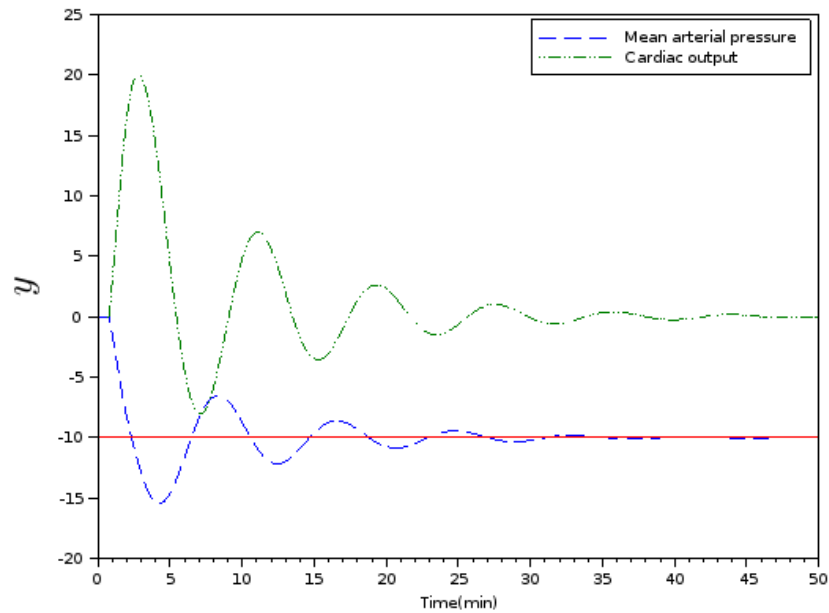


Figure 22.4: Influence of drugs

```

46 U=25;
47 y = Gb+ode(y0, t0, t, list(model,U))
48 subplot(3,1,3);
49 plot(t,y(1,:));
50 xtitle("","Time(min)","Glucose (mg/L)")
51 legend("$U=25\ mU/min$")

```

---

Scilab code Exa 22.4 Influence of drugs

```

1 clear
2 clc

```

```

3
4 //Example 22.4
5 disp('Example 22.4')
6 K=[-6 3;12 5];
7 tau=[0.67 2;0.67 5];
8 s=%s;
9 G=K./(1+tau*s);
10 delay75=0.75;
11 delay1=1;
12 RGA=K.*inv(K');
13 disp(RGA,"RGA=")
14
15 //IMC based tuning
16 tauC=[tau(1,1) tau(2,2)];
17 Kc=diag(tau./(repmat(tauC,2,1)+[delay75 delay1;
    delay75 delay1])./K);
18 tauI=diag(tau);
19 disp(Kc,"Kc=")
20 disp(tauI,"tauI=")
21
22 //====Making step response models of the continuos
    transfer functions====//
23 Ts=0.05;//Sampling time ie delta_T
24 delay75=0.75/Ts;
25 delay1=1/Ts;
26 N=30/Ts;//Model Order
27 s=%s;
28 G=syslin('c',diag(matrix(G,1,4)));//Transfer
    function
29 t=0:Ts:N*Ts;
30 u_sim=ones(4,length(t));
31 u_sim([1 2],1:(delay75))=zeros(2,delay75); //input
    delay to account for delay in SNP
32 u_sim([3 4],1:(delay1))=zeros(2,delay1); //input
    delay to account for delay in DPM
33 S=csim(u_sim,t,G)';//generating step response model
    for real plant
34 //plot(t,S);

```

```

35 S(1,:)=[];
36 //Now we have these step response models for each of
    the transfer functions
37 //[S1 S3
38 //S2 S4
39
40
41
42
43 T=50; //Simulation Run Time in minutes
44 n=T/Ts*2+1; //no. of discrete points in our domain
    of analysis
45
46
47 //=====Set point as -10=====//
48 //p is the controller output
49 p=zeros(n,2);
50 delta_p=zeros(n,2);
51 e=zeros(n,2); //errors=(ysp-y) on which PI acts
52 ysp=zeros(n,2);
53 ysp((n-1)/2+1:n,1)=-10*ones(n-((n-1)/2+1)+1,1);
54
55 t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
56 y=zeros(n,2);
57
58
59 for k=(n-1)/2+1:n
60
61     //Error e
62     e(k,:)=ysp(k-1,:)-y(k-1,:);
63     delta_e(k,:)=e(k,:)-e(k-1,:);
64
65     //Controller calculation ——Digital PID——Eqn
        7-28 Pg 136 (Velocity form)
66 //     p(k,:)=p(k-1,:)+flipdim([delta_e(k,:)+e(k,:)*
diag(Ts./tauI)]*diag(Kc),2);
67     p(k,:)=p(k-1,:)+([delta_e(k,:)+e(k,:)*diag(Ts./
tauI)]*diag(Kc));

```

```

68     //1-1/2-2 pairing
69
70     delta_p(k,:) = p(k,:) - p(k-1,:);
71
72     //Output
73     y(k,1) = [S(1:N-1,1); S(1:N-1,3)]' ...
74             * [flipdim(delta_p(k-N+1:k-1,1),1); flipdim(
75                 delta_p(k-N+1:k-1,2),1)] ...
76             + [S(N,1) S(N,3)] * [p(k-N,1); p(k-N,2)];
77     y(k,2) = [S(1:N-1,2); S(1:N-1,4)]' ...
78             * [flipdim(delta_p(k-N+1:k-1,1),1); flipdim(
79                 delta_p(k-N+1:k-1,2),1);] ...
80             + [S(N,2) S(N,4)] * [p(k-N,1); p(k-N,2)];
81
82     plot(t', y(:,1), '--', t', y(:,2), ':', t', ysp(:,1), '-');
83     set(gca(), "data_bounds", [0 50 -20 25]); //putting
84         bounds on display
85 l = legend("Mean arterial pressure", "Cardiac output",
86         position=1);
87 //l.font_size=5;
88 xtitle("", "Time(min)", "$y$");
89 a = get("current_axes");
90 c = a.y_label; c.font_size=5;
91
92 mprintf('\nThere is more interaction in the
93     variables\n...
94     than the book claims, hence a mismatch between the
95     result\n...
96     and the book\n')

```

---